

# 完整 RAG 实施方案 2- 适配现有项目

## 📋 实施计划总览

阶段 1: 数据库准备 (Day 1-2)
阶段 2: 依赖安装与配置 (Day 2)
阶段 3: 核心工具函数 (Day 3-4)
阶段 4: API 路由实现 (Day 5-6)
阶段 5: 前端界面集成 (Day 7)
阶段 6: 测试与优化 (Day 8)

## 📁 阶段 1: 数据库准备

### 步骤 1.1: 安装 pgvector 扩展

#### 方案 A: 使用 Docker (推荐)

```
# 使用带 pgvector 的 PostgreSQL 镜像
docker pull pgvector/pgvector:pg16

# 运行容器
docker run -d \
  --name postgres-vector \
  -e POSTGRES_PASSWORD=your_password \
  -e POSTGRES_DB=ai_chat \
  -p 5432:5432 \
  pgvector/pgvector:pg16
```

#### 方案 B: 云服务 (最简单)

如果使用 Supabase 或 Neon, pgvector 已预装, 直接在 SQL 编辑器执行:

```
-- 启用 pgvector 扩展
CREATE EXTENSION IF NOT EXISTS vector;

-- 验证安装
SELECT * FROM pg_extension WHERE extname = 'vector';
```

## 方案 C: 本地安装 (Linux/Mac)

```
# Ubuntu/Debian
sudo apt install postgresql-16-pgvector

# macOS (Homebrew)
brew install pgvector

# 然后在 PostgreSQL 中启用
psql -U postgres -d ai_chat -c "CREATE EXTENSION vector;"
```

## 步骤 1.2: 修改 Prisma Schema

✅ 修改位置 1: `prisma/schema.prisma`

```
generator client {
  provider      = "prisma-client-js"
  previewFeatures = ["postgresqlExtensions"]
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
  extensions = [vector]
}

model User {
  id          String      @id @default(uuid())
  email       String      @unique
  passwordHash String      @map("password_hash")
  name        String?
  avatarUrl   String?      @map("avatar_url")
  role        String      @default("user")
  createdAt   DateTime     @default(now()) @map("created_at")
  updatedAt   DateTime     @updatedAt @map("updated_at")
  conversations Conversation[]
  pdfs        PDF[]
  usageLogs   UsageLog[]

  @@map("users")
}

model Conversation {
  id          String      @id @default(uuid())
  userId      String      @map("user_id")
  title       String      @default("新对话")
  model       String      @default("gpt-4o")
  createdAt   DateTime     @default(now()) @map("created_at")
  updatedAt   DateTime     @updatedAt @map("updated_at")
  user        User         @relation(fields: [userId], references: [id], onDelete: cascade)
  messages    Message[]
}
```

```

    @@index([userId])
    @@index([updatedAt])
    @@map("conversations")
}

model Message {
  id          String      @id @default(uuid())
  conversationId String    @map("conversation_id")
  role        String
  content      String
  images       Json?
  model        String?
  tokensUsed   Int?       @map("tokens_used")
  createdAt    DateTime    @default(now()) @map("created_at")
  citations     Json?
  iswebSearch  Boolean     @default(false) @map("is_web_search")
  conversation Conversation @relation(fields: [conversationId], references: [id],
onDelete: Cascade)

  @@index([conversationId])
  @@index([createdAt])
  @@map("messages")
}

model ModelConfig {
  id          String      @id @default(uuid())
  modelId     String      @unique @map("model_id")
  displayName  String      @map("display_name")
  provider     String?
  supportsvision Boolean   @default(false) @map("supports_vision")
  maxTokens   Int         @default(4096) @map("max_tokens")
  isActive    Boolean     @default(true) @map("is_active")
  sortOrder   Int         @default(0) @map("sort_order")
  createdAt   DateTime    @default(now()) @map("created_at")

  @@map("model_configs")
}

model UsageLog {
  id          String      @id @default(uuid())
  userId      String      @map("user_id")
  conversationId String?   @map("conversation_id")
  model        String
  tokensUsed   Int         @map("tokens_used")
  cost         Decimal     @db.Decimal(10, 6)
  createdAt    DateTime    @default(now()) @map("created_at")
  user         User        @relation(fields: [userId], references: [id], onDelete: Cascade)

  @@index([userId])
  @@index([createdAt])
  @@map("usage_logs")
}

```

```

model PDF {
  id          String          @id @default(cuid())
  userId      String
  name        String
  fileName    String
  filePath    String
  size        Int
  createdAt   DateTime        @default(now())
  updatedAt   DateTime        @updatedAt
  errorMessage String?
  processedAt DateTime?
  status      String          @default("processing")
  totalChunks Int             @default(0)
  totalPages  Int?
  chunks      DocumentChunk[]
  user        User            @relation(fields: [userId], references: [id], onDelete:
Cascade)

  @@index([userId])
  @@index([status])
  @@map("pdfs")
}

model DocumentChunk {
  id          String          @id @default(cuid())
  pdfId       String          @map("pdf_id")
  chunkIndex  Int             @map("chunk_index")
  content     String
  embedding   Unsupported("vector(1024)")?
  pageNumber  Int?            @map("page_number")
  startChar   Int?            @map("start_char")
  endChar     Int?            @map("end_char")
  tokenCount  Int?            @map("token_count")
  metadata    Json?
  createdAt   DateTime        @default(now()) @map("created_at")
  pdf         PDF             @relation(fields: [pdfId], references: [id], onDelete:
Cascade)

  @@index([pdfId])
  @@index([chunkIndex])
  @@map("document_chunks")
}

```

### 关键说明:

- `Unsupported("vector(1536)")`: Prisma 对 pgvector 的支持方式
- `1536`: OpenAI text-embedding-3-small 的向量维度
- `@@index`: 加速查询性能

## 步骤 1.3: 创建数据库迁移

```
# 生成迁移文件
npx prisma migrate dev --name add_rag_support

# 如果遇到问题，可以先重置
# npx prisma migrate reset

# 生成 Prisma Client
npx prisma generate
```

## 步骤 1.4: 创建向量索引 (SQL)

✅ 修改位置 2: 创建新文件 `prisma/migrations/create_vector_index.sql`

```
-- =====
-- 创建向量相似度搜索索引
-- =====

-- 使用 IVFFlat 索引（适合中小规模数据）
CREATE INDEX IF NOT EXISTS document_chunks_embedding_idx
ON document_chunks
USING ivfflat (embedding vector_cosine_ops)
WITH (lists = 100);

-- 说明：
-- - ivfflat：倒排文件索引，适合 10K-1M 向量
-- - vector_cosine_ops：余弦相似度（推荐）
-- - lists = 100：聚类数量（建议为行数的平方根）

-- 如果数据量更大（> 1M），使用 HNSW 索引：
-- CREATE INDEX document_chunks_embedding_idx
-- ON document_chunks
-- USING hnsw (embedding vector_cosine_ops);
```

手动执行：

```
# 连接数据库执行
psql -U your_user -d ai_chat -f prisma/migrations/create_vector_index.sql
```

## 阶段 2: 依赖安装与配置

### 步骤 2.1: 安装 NPM 包

```
# 进入项目目录
cd ai-chat-app

# 安装 LangChain 相关包
```

```
npm install langchain @langchain/openai @langchain/community
```

```
# 安装文本处理工具
```

```
npm install tiktoken # Token 计数
```

```
# 安装 PDF 解析（已有，确认版本）
```

```
npm install pdf-parse
```

```
# 验证安装
```

```
npm list langchain
```

## 步骤 2.2: 更新环境变量

✅ 修改位置 3: `.env.local`

```
OPENAI_API_KEY=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
```

```
NEXTAUTH_URL=http://localhost:3000
```

```
DATABASE_URL="postgresql://postgres:li2523323@172.24.47.103:5432/ai_chat"
```

```
NEXTAUTH_SECRET=your_nextauth_secret_here
```

```
# 新增: 博查搜索 API Key
```

```
BOCHA_API_KEY=sk-a5a4cb5a769e44b0806b93bca6934c91
```

```
OPENAI_BASE_URL = https://openrouter.ai/api/v1
```

```
# 应用信息（OpenRouter 需要）
```

```
NEXT_PUBLIC_APP_URL=http://localhost:3000
```

```
NEXT_PUBLIC_APP_NAME=AI Chat App
```

```
# =====
```

```
# ✅ 新增: RAG 配置
```

```
# =====
```

```
# Embedding 模型配置
```

```
OPENAI_EMBEDDING_DIMENSIONS=1024
```

```
# Embedding 模型
```

```
OPENAI_EMBEDDING_MODEL=qwen/qwen3-embedding-0.6b
```

```
# 批次大小
```

```
EMBEDDING_BATCH_SIZE=10
```

```
# 文档分块配置
```

```
CHUNK_SIZE=1000 # 每块字符数
```

```
CHUNK_OVERLAP=200 # 重叠字符数
```

```
MAX_CHUNKS_PER_PDF=500 # 单个 PDF 最大块数
```

```
# 检索配置
```

```
RETRIEVAL_TOP_K=5 # 返回最相关的 K 个块
```

```
SIMILARITY_THRESHOLD=0.7 # 相似度阈值（0-1）
```

```
# 处理配置
```

```
ENABLE_ASYNC_PROCESSING=true # 启用异步处理
```

```
MAX_PDF_SIZE_MB=20 # 最大 PDF 大小
```

## 🔧 阶段 3: 核心工具函数

### 步骤 3.1: 创建 Embedding 工具

✅ 修改位置 4: 创建新文件 `lib/rag/embeddings.js`

```
/**
 * =====
 * Embedding 工具 (lib/rag/embeddings.js)
 * =====
 *
 * 功能:
 *   1. 文本向量化 (单个/批量)
 *   2. Token 计数
 *   3. 成本估算
 *
 * 使用:
 *   import { embedText, embedBatch } from '@lib/rag/embeddings';
 * =====
 */

/**
 * =====
 * Embedding 工具 (lib/rag/embeddings.js)
 * =====
 * 使用原生 Fetch 调用 OpenRouter Embeddings API
 */

import { encoding_for_model } from 'tiktoken';

// =====
// 配置
// =====
const OPENAI_API_KEY = process.env.OPENAI_API_KEY;
const OPENAI_BASE_URL = process.env.OPENAI_BASE_URL || 'https://openrouter.ai/api/v1';
const EMBEDDING_MODEL = process.env.OPENAI_EMBEDDING_MODEL || 'qwen/qwen3-embedding-0.6b';
const APP_URL = process.env.NEXT_PUBLIC_APP_URL || 'http://localhost:3000';
const APP_NAME = process.env.NEXT_PUBLIC_APP_NAME || 'AI Chat App';

// =====
// Token 计数器
// =====
let tokenizer;
try {
  tokenizer = encoding_for_model('gpt-3.5-turbo');
} catch (error) {
  console.warn('⚠️ Tiktoken 初始化失败, 使用估算方法');
}
```

```

export function countTokens(text) {
  if (!text) return 0;

  if (tokenizer) {
    try {
      const tokens = tokenizer.encode(text);
      return tokens.length;
    } catch (error) {
      console.error('Token 计数失败:', error);
    }
  }

  return Math.ceil(text.length / 4);
}

// =====
// ✅ 原生 Fetch 实现：单个文本向量化
// =====
export async function embedText(text) {
  if (!text || !text.trim()) {
    throw new Error('文本不能为空');
  }

  if (!OPENAI_API_KEY) {
    throw new Error('❌ OPENAI_API_KEY 未配置');
  }

  try {
    console.log('🔄 开始向量化，文本长度:', text.length);
    const startTime = Date.now();

    // ✅ 按照 OpenRouter 官方文档格式调用
    const response = await fetch(`${OPENAI_BASE_URL}/embeddings`, {
      method: 'POST',
      headers: {
        'Authorization': `Bearer ${OPENAI_API_KEY}`,
        'Content-Type': 'application/json',
        'HTTP-Referer': APP_URL, // ✅ 必需
        'X-Title': APP_NAME, // ✅ 必需
      },
      body: JSON.stringify({
        model: EMBEDDING_MODEL,
        input: text,
      }),
    });

    // ✅ 详细错误处理
    if (!response.ok) {
      const errorText = await response.text();
      console.error('❌ API 错误响应:', {
        status: response.status,
        statusText: response.statusText,

```

```

        body: errorText,
      });
      throw new Error(`API 错误 (${response.status}): ${errorText}`);
    }

    const data = await response.json();

    // ✅ 验证返回格式（按照官方文档）
    if (!data.data || !Array.isArray(data.data) || data.data.length === 0) {
      console.error('❌ API 返回格式错误:', data);
      throw new Error(`API 返回格式错误: ${JSON.stringify(data)}`);
    }

    if (!data.data[0].embedding || !Array.isArray(data.data[0].embedding)) {
      console.error('❌ 缺少 embedding 字段:', data.data[0]);
      throw new Error('API 返回缺少 embedding 字段');
    }

    const vector = data.data[0].embedding;
    const duration = Date.now() - startTime;

    console.log(`✅ 向量化完成, 耗时: ${duration}ms, 维度: ${vector.length}`);
    console.log(`💰 成本: $$${data.usage?.cost || '未知'}`);

    return vector;
  } catch (error) {
    console.error('❌ 向量化失败:', error);
    console.error('配置信息:', {
      baseUrl: OPENAI_BASE_URL,
      model: EMBEDDING_MODEL,
      apiKey: OPENAI_API_KEY ? `${OPENAI_API_KEY.slice(0, 10)}...` : '未配置',
      textLength: text.length,
    });
    throw new Error(`向量化失败: ${error.message}`);
  }
}

// =====
// ✅ 原生 Fetch 实现: 批量文本向量化
// =====
export async function embedBatch(texts, options = {}) {
  const {
    batchSize = 50, // OpenRouter 建议批次大小
    showProgress = true,
  } = options;

  if (!texts || texts.length === 0) {
    return [];
  }

  if (!OPENAI_API_KEY) {
    throw new Error('❌ OPENAI_API_KEY 未配置');
  }

```

```

}

console.log(`🔄 批量向量化开始，总数：${texts.length}`);
console.log(` - 模型：${EMBEDDING_MODEL}`);
console.log(` - 批次大小：${batchSize}`);

const startTime = Date.now();
const results = [];
let totalCost = 0;

// 分批处理
for (let i = 0; i < texts.length; i += batchSize) {
  const batch = texts.slice(i, i + batchSize);
  const batchNum = Math.floor(i / batchSize) + 1;
  const totalBatches = Math.ceil(texts.length / batchSize);

  if (showProgress) {
    console.log(`📊 处理批次 ${batchNum}/${totalBatches} (${batch.length} 个文本)`);
  }

  try {
    // ✅ 按照 OpenRouter 官方文档格式调用（批量）
    const response = await fetch(`${OPENAI_BASE_URL}/embeddings`, {
      method: 'POST',
      headers: {
        'Authorization': `Bearer ${OPENAI_API_KEY}`,
        'Content-Type': 'application/json',
        'HTTP-Referer': APP_URL,
        'X-Title': APP_NAME,
      },
      body: JSON.stringify({
        model: EMBEDDING_MODEL,
        input: batch, // ✅ 数组形式
      }),
    });

    if (!response.ok) {
      const errorText = await response.text();
      throw new Error(`API 错误 (${response.status}): ${errorText}`);
    }

    const data = await response.json();

    // ✅ 验证返回数据
    if (!data.data || data.data.length !== batch.length) {
      throw new Error(`返回数量不匹配：期望 ${batch.length}，实际 ${data.data?.length || 0}`);
    }

    // ✅ 提取 embeddings
    const vectors = data.data.map(item => item.embedding);
    results.push(...vectors);
  }
}

```

```

// 累计成本
if (data.usage?.cost) {
    totalCost += parseFloat(data.usage.cost);
}

console.log(`✅ 批次 ${batchNum} 完成`);

} catch (error) {
    console.error(`❌ 批次 ${batchNum} 失败:`, error.message);

// ✅ 失败时逐个重试
console.log(`🔄 逐个重试批次 ${batchNum}...`);
for (let j = 0; j < batch.length; j++) {
    try {
        const vector = await embedText(batch[j]);
        results.push(vector);

        // 避免频繁请求
        if (j < batch.length - 1) {
            await new Promise(resolve => setTimeout(resolve, 300));
        }
    } catch (retryError) {
        console.error(`❌ 文本 ${i + j} 重试失败:`, retryError.message);
        // 返回零向量（避免数据库错误）
        results.push(new Array(1024).fill(0));
    }
}
}

// 批次间延迟，避免限流
if (i + batchSize < texts.length) {
    await new Promise(resolve => setTimeout(resolve, 500));
}
}

const duration = Date.now() - startTime;
console.log(`✅ 批量向量化完成，耗时: ${duration}ms`);
console.log(`💰 总成本: ${totalCost.toFixed(6)}`);

return results;
}

// =====
// 成本估算
// =====
export function estimateCost(tokenCount) {
    // OpenAI text-embedding-3-small: $0.02 / 1M tokens
    const costPerMillion = 0.00001;
    const cost = (tokenCount / 1000000) * costPerMillion;

    return {
        tokens: tokenCount,
        cost: cost.toFixed(6),
    };
}

```

```

    costUSD: `${cost.toFixed(6)}`,
    costCNY: `¥${(cost * 7.2).toFixed(4)}`,
  };
}

// =====
// 导出
// =====
export default {
  embedText,
  embedBatch,
  countTokens,
  estimateCost,
};

```

## 步骤 3.2: 创建文本分块工具

✓ 修改位置 5: 创建新文件 `lib/rag/chunking.js`

```

/**
 * =====
 * 文本分块工具 (lib/rag/chunking.js)
 * =====
 *
 * 功能:
 *   1. 递归字符分割
 *   2. 保持语义完整
 *   3. 添加元数据
 *
 * 使用:
 *   import { chunkText } from '@lib/rag/chunking';
 * =====
 */

import { RecursiveCharacterTextSplitter } from '@langchain/textsplitters';
import { countTokens } from './embeddings';

// =====
// 配置
// =====
const DEFAULT_CHUNK_SIZE = parseInt(process.env.CHUNK_SIZE || '1000');
const DEFAULT_CHUNK_OVERLAP = parseInt(process.env.CHUNK_OVERLAP || '200');

// =====
// 创建分块器
// =====
/**
 * 创建文本分块器
 * @param {Object} options - 配置选项
 * @returns {RecursiveCharacterTextSplitter}

```

```

*/
export function createSplitter(options = {}) {
  const {
    chunkSize = DEFAULT_CHUNK_SIZE,
    chunkOverlap = DEFAULT_CHUNK_OVERLAP,
    separators = ['\n\n', '\n', '。', '!', '?', ';', ':', ',', ' ', ' ', ' ', ' '],
  } = options;

  return new RecursiveCharacterTextSplitter({
    chunkSize,
    chunkOverlap,
    separators,
    lengthFunction: (text) => text.length, // 按字符计数
  });
}

// =====
// 文本分块（核心函数）
// =====
/**
 * 将文本分块
 * @param {string} text - 输入文本
 * @param {Object} metadata - 元数据
 * @param {Object} options - 配置选项
 * @returns {Promise<Array>} 分块结果
 */
export async function chunkText(text, metadata = {}, options = {}) {
  if (!text || !text.trim()) {
    console.warn('⚠️ 输入文本为空');
    return [];
  }

  console.log('🔪 开始文本分块...');
  console.log('📏 原始文本长度:', text.length);

  const startTime = Date.now();

  try {
    // 创建分块器
    const splitter = createSplitter(options);

    // 执行分块
    const docs = await splitter.createDocuments([text], [metadata]);

    // 处理结果
    const chunks = docs.map((doc, index) => {
      const content = doc.pageContent;
      const tokens = countTokens(content);

      return {
        chunkIndex: index,
        content: content,
        tokenCount: tokens,
      };
    });
  } catch (error) {
    console.error('❌ 文本分块失败:', error);
    return [];
  }
}

```

```

        charCount: content.length,
        metadata: {
            ...doc.metadata,
            ...metadata,
        },
    },
};
});

const duration = Date.now() - startTime;

console.log('✅ 分块完成');
console.log('📊 统计信息:', {
    totalChunks: chunks.length,
    avgChunkSize: Math.round(text.length / chunks.length),
    totalTokens: chunks.reduce((sum, c) => sum + c.tokenCount, 0),
    duration: `${duration}ms`,
});

return chunks;

} catch (error) {
    console.error('❌ 分块失败:', error);
    throw new Error(`文本分块失败: ${error.message}`);
}
}

// =====
// 智能分块（按页码）
// =====
/**
 * 按页码分块（适合 PDF）
 * @param {Object} pdfData - PDF 解析数据
 * @param {Object} options - 配置选项
 * @returns {Promise<Array>} 分块结果
 */
export async function chunkByPages(pdfData, options = {}) {
    const { text, numpages, metadata } = pdfData;

    console.log('📄 按页码分块, 总页数:', numpages);

    // 如果有页码信息, 按页分块
    if (metadata?.pageTexts && Array.isArray(metadata.pageTexts)) {
        const allChunks = [];

        for (const page of metadata.pageTexts) {
            const pageChunks = await chunkText(
                page.text,
                {
                    pageNumber: page.page,
                    source: 'pdf',
                },
                options
            );

```

```

    allChunks.push(...pageChunks);
  }

  return allChunks;
}

// 否则整体分块
return chunkText(text, { source: 'pdf', totalPages: numpages }, options);
}

// =====
// 导出
// =====
export default {
  chunkText,
  chunkByPages,
  createSplitter,
};

```

### 步骤 3.3: 创建向量检索工具

✅ 修改位置 6: 创建新文件 `lib/rag/retrieval.js`

```

/**
 * =====
 * 向量检索工具 (lib/rag/retrieval.js) - 修复列名问题
 * =====
 */

import { prisma } from '@lib/prisma';
import { embedText } from './embeddings';
import { Prisma } from '@prisma/client';

const DEFAULT_TOP_K = parseInt(process.env.RETRIEVAL_TOP_K || '5');
const DEFAULT_THRESHOLD = parseFloat(process.env.SIMILARITY_THRESHOLD || '0.7');

export async function searchSimilarChunks(query, options = {}) {
  const {
    pdfId = null,
    topK = DEFAULT_TOP_K,
    threshold = DEFAULT_THRESHOLD,
    includeMetadata = true,
  } = options;

  console.log('🔍 开始相似度搜索...');
  console.log('📄 查询:', query);
  console.log('📊 查询参数:', { pdfId, topK, threshold });

  try {

```

```

const queryVector = await embedText(query);
console.log('✅ 查询向量化完成');

const vectorString = `[${queryVector.join(',')}]`;

console.log('📖 执行数据库查询...');
console.log('🔗 使用 Prisma.$queryRawUnsafe（混合方案）');
console.log('📝 参数:', {
  vectorLength: vectorString.length,
  pdfId: pdfId || 'all',
  threshold,
  topK
});

// ✅ 修改点：根据实际列名调整（三种可能的列名）
let sql = `
  SELECT
    dc.id,
    dc.pdf_id as "pdfId",
    dc.chunk_index as "chunkIndex",
    dc.content,
    dc.page_number as "pageNumber",
    dc.token_count as "tokenCount",
    dc.metadata,
    p.name as "pdfName",
    p."filePath" as "pdfPath",
    1 - (dc.embedding <=> $1::vector) as similarity
  FROM document_chunks dc
  JOIN pdfs p ON dc.pdf_id = p.id
  WHERE dc.embedding IS NOT NULL
`;

const params = [vectorString];
let paramIndex = 2;

if (pdfId) {
  sql += ` AND p.id = ${paramIndex}`;
  params.push(pdfId);
  paramIndex++;
}

sql += ` AND (1 - (dc.embedding <=> $1::vector)) >= ${paramIndex}`;
params.push(threshold);
paramIndex++;

sql += `
  ORDER BY dc.embedding <=> $1::vector
  LIMIT ${paramIndex}
`;
params.push(topK);

console.log('📝 最终 SQL:', sql.substring(0, 300) + '...');
console.log('📝 参数列表:', params.map((p, i) =>

```

```

    i === 0 ? `${i+1}: [向量,长度=${p.length}]` : `${i+1}: ${p}`
  ));

  const results = await prisma.$queryRawUnsafe(sql, ...params);

  console.log(`✅ 找到 ${results.length} 个相似块`);

  const formattedResults = results.map(row => ({
    id: row.id,
    pdfId: row.pdfId,
    pdfName: row.pdfName,
    pdfPath: row.pdfPath,
    chunkIndex: row.chunkIndex,
    content: row.content,
    pageNumber: row.pageNumber,
    tokenCount: row.tokenCount,
    similarity: parseFloat(row.similarity.toFixed(4)),
    metadata: includeMetadata ? row.metadata : undefined,
  }));

  if (formattedResults.length > 0) {
    console.log('📄 检索结果摘要:');
    formattedResults.forEach((r, i) => {
      console.log(` ${i + 1}. 相似度: ${r.similarity}, 页码: ${r.pageNumber || 'N/A'}, 长度: ${r.content.length}`);
    });
  } else {
    console.log('⚠️ 未找到满足条件的结果');
  }

  return formattedResults;

} catch (error) {
  console.error('❌ 检索失败:', error);
  console.error('错误详情:', error.message);
  console.error('错误代码:', error.code);
  console.error('错误堆栈:', error.stack);

  throw new Error(`向量检索失败: ${error.message}`);
}

}

export async function hybridSearch(query, options = {}) {
  const {
    pdfId = null,
    topK = DEFAULT_TOP_K,
    vectorWeight = 0.7,
    keywordWeight = 0.3,
  } = options;

  console.log('🔍 混合检索开始...');

  try {

```

```

const vectorResults = await searchSimilarChunks(query, {
  pdfId,
  topK: topK * 2,
  threshold: 0.5,
});

const keywords = query.split(/\s+/).filter(k => k.length > 1);

let keywordResults = [];
if (keywords.length > 0) {
  const whereCondition = {
    content: {
      contains: keywords.join(' '),
      mode: 'insensitive',
    },
  },
};

if (pdfId) {
  whereCondition.pdfId = pdfId;
}

keywordResults = await prisma.documentChunk.findMany({
  where: whereCondition,
  include: {
    pdf: {
      select: {
        name: true,
        filePath: true,
      },
    },
  },
  take: topK * 2,
});

const combinedMap = new Map();

vectorResults.forEach(result => {
  combinedMap.set(result.id, {
    ...result,
    vectorScore: result.similarity,
    keywordScore: 0,
    finalScore: result.similarity * vectorWeight,
  });
});

keywordResults.forEach(result => {
  const keywordScore = calculateKeywordScore(result.content, keywords);

  if (combinedMap.has(result.id)) {
    const existing = combinedMap.get(result.id);
    existing.keywordScore = keywordScore;
    existing.finalScore =

```

```

        existing.vectorScore * vectorWeight +
        keywordScore * keywordWeight;
    } else {
        combinedMap.set(result.id, {
            id: result.id,
            pdfId: result.pdfId,
            pdfName: result.pdf.name,
            pdfPath: result.pdf.filePath,
            chunkIndex: result.chunkIndex,
            content: result.content,
            pageNumber: result.pageNumber,
            tokenCount: result.tokenCount,
            vectorScore: 0,
            keywordScore: keywordScore,
            finalScore: keywordScore * keywordWeight,
        });
    }
});

const finalResults = Array.from(combinedMap.values())
    .sort((a, b) => b.finalScore - a.finalScore)
    .slice(0, topK);

console.log(`✅ 混合检索完成, 返回 ${finalResults.length} 个结果`);

return finalResults;

} catch (error) {
    console.error('❌ 混合检索失败:', error);
    throw error;
}
}

function calculateKeywordScore(content, keywords) {
    if (!keywords || keywords.length === 0) return 0;

    const lowerContent = content.toLowerCase();
    let matchCount = 0;

    keywords.forEach(keyword => {
        const regex = new RegExp(keyword.toLowerCase(), 'g');
        const matches = lowerContent.match(regex);
        if (matches) {
            matchCount += matches.length;
        }
    });

    return Math.min(matchCount / (keywords.length * 3), 1);
}

export default {
    searchSimilarChunks,
    hybridSearch,

```

```
};
```

## 阶段 4: API 路由实现

### 步骤 4.1: 修改 PDF 上传 API (添加异步处理)

✅ 修改位置 7: 修改 `app/api/pdf/upload/route.js`

在现有代码的基础上, 添加以下修改:

```
// ✅ 修改点 1: 在文件顶部添加导入
import { chunkText } from '@lib/rag/chunking';
import { embedBatch } from '@lib/rag/embeddings';
import { NextResponse } from 'next/server';
import { auth } from '@app/api/auth/[...nextauth]/route'; // ✅ 从 NextAuth 路由导入
import { authOptions } from '@app/api/auth/[...nextauth]/route';
import { prisma } from '@lib/prisma';
import { promises as fs } from 'fs';
import path from 'path';

// ✅ 在文件顶部添加配置
const VECTOR_DIMENSION = 1024; // 根据模型调整
// GET 方法用于测试
export async function GET() {
  return NextResponse.json({
    success: true,
    message: 'PDF 上传 API 正常运行',
    timestamp: new Date().toISOString()
  });
}

// POST 方法处理文件上传
export async function POST(req) {
  console.log('📄 收到上传请求');

  try {
    // 1. 身份验证
    const session = await auth();

    if (!session?.user?.id) {
      console.log('❌ 用户未登录');
      return NextResponse.json(
        { success: false, error: '未登录, 请先登录' },
        { status: 401 }
      );
    }

    console.log('✅ 用户已登录:', session.user.email);

    // 2. 解析表单数据
```

```
const formData = await req.formData();
const file = formData.get('file');

if (!file) {
  console.log('❌ 未选择文件');
  return NextResponse.json(
    { success: false, error: '未选择文件' },
    { status: 400 }
  );
}

console.log('📄 文件信息:', {
  name: file.name,
  size: file.size,
  type: file.type
});

// 3. 验证文件类型
if (file.type !== 'application/pdf') {
  return NextResponse.json(
    { success: false, error: '仅支持 PDF 文件' },
    { status: 400 }
  );
}

// 4. 验证文件大小 (20MB)
if (file.size > 20 * 1024 * 1024) {
  return NextResponse.json(
    { success: false, error: '文件大小不能超过 20MB' },
    { status: 400 }
  );
}

// 5. 生成文件名
const timestamp = Date.now();
const originalName = file.name;
const ext = path.extname(originalName);
const baseName = path.basename(originalName, ext);
const fileName = `${baseName}_${timestamp}${ext}`;

// 6. 创建上传目录
const year = new Date().getFullYear().toString();
const month = (new Date().getMonth() + 1).toString();
const uploadDir = path.join(
  process.cwd(),
  'public',
  'uploads',
  'pdfs',
  year,
  month
);

console.log('📁 创建目录:', uploadDir);
```

```

await fs.mkdir(uploadDir, { recursive: true });

// 7. 保存文件
const filePath = path.join(uploadDir, fileName);
const buffer = Buffer.from(await file.arrayBuffer());
await fs.writeFile(filePath, buffer);
console.log('✅ 文件已保存:', filePath);

// 8. 生成访问 URL
const fileUrl = `/uploads/pdfs/${year}/${month}/${fileName}`;

// 9. 保存到数据库
const pdfRecord = await prisma.pdf.create({
  data: {
    userId: session.user.id,
    name: originalName,
    fileName: fileName,
    filePath: fileUrl,
    size: file.size,
    status: 'processing', // ✅ 修改: 设置为处理中
  },
});

console.log('✅ 数据库保存成功:', pdfRecord.id);
// ✅ 修改点 3: 启动异步 RAG 处理（不阻塞响应）
processPdfInBackground(pdfRecord.id, filePath).catch(error => {
  console.error('❌ 后台处理失败:', error);
});

// 10. 返回成功响应
return NextResponse.json({
  success: true,
  message: '上传成功, 正在处理中...',
  data: pdfRecord,
});

} catch (error) {
  console.error('❌ 上传失败:', error);
  console.error('错误堆栈:', error.stack);

  return NextResponse.json(
    {
      success: false,
      error: '上传失败: ' + error.message,
    },
    { status: 500 }
  );
}
}

// ✅ 修改点 4: 新增后台处理函数
// =====

```

```
// ✅ 修改后的后台处理函数
// =====
// ✅ 完整修复代码
async function processPdfInBackground(pdfId, filePath) {
  console.log('🔄 开始后台处理 PDF:', pdfId);

  try {
    // 1. 解析 PDF
    const pdfParse = require('pdf-parse');
    const dataBuffer = await fs.readFile(filePath);
    const pdfData = await pdfParse(dataBuffer);

    console.log('📄 PDF 解析完成, 页数:', pdfData.numpages);

    // 2. 文本分块
    const chunks = await chunkText(pdfData.text, {
      source: 'pdf',
      totalPages: pdfData.numpages,
    });

    console.log('📄 分块完成, 总数:', chunks.length);

    // 3. 批量向量化
    const texts = chunks.map(c => c.content);
    const vectors = await embedBatch(texts);

    console.log('📊 向量化完成');

    // ✅ 动态获取向量维度
    const actualDimension = vectors[0]?.length || VECTOR_DIMENSION;
    console.log('📏 向量维度: ${actualDimension}');

    console.log('📁 开始保存文档块...');

    let successCount = 0;
    let failCount = 0;

    for (let i = 0; i < chunks.length; i++) {
      const chunk = chunks[i];
      const vector = vectors[i];

      try {
        if (vector && vector.length > 0) {
          // ✅ 方案 1: 使用 $executeRawUnsafe
          const vectorStr = `[${vector.join(',')}]`;
          const metadataStr = JSON.stringify(chunk.metadata || {});

          await prisma.$executeRawUnsafe(
            `INSERT INTO document_chunks (
              id, pdf_id, chunk_index, content, embedding,
              token_count, page_number, metadata, created_at
            ) VALUES (
              $1, $2, $3, $4, $5::vector(${actualDimension}), $6, $7, $8::jsonb, NOW()
            )`
          );
        }
      } catch (error) {
        failCount++;
        console.error(`保存文档块失败: ${error.message}`);
      }
      successCount++;
    }

    console.log(`处理完成: 成功 ${successCount}, 失败 ${failCount}`);
  } catch (error) {
    console.error('后台处理 PDF 失败:', error);
  }
}
```

```

        )`,
        `chunk_${pdfId}_${i}`,
        pdfId,
        chunk.chunkIndex,
        chunk.content,
        vectorStr,
        chunk.tokenCount || 0,
        chunk.metadata?.pageNumber || null,
        metadataStr
    );

} else {
    // 没有向量时使用 ORM
    await prisma.documentChunk.create({
        data: {
            id: `chunk_${pdfId}_${i}`,
            pdfId: pdfId,
            chunkIndex: chunk.chunkIndex,
            content: chunk.content,
            tokenCount: chunk.tokenCount || 0,
            pageNumber: chunk.metadata?.pageNumber || null,
            metadata: chunk.metadata || {},
        }
    });
}

successCount++;

if ((i + 1) % 10 === 0) {
    console.log(`✅ 已保存 ${i + 1}/${chunks.length} 个块`);
}

} catch (insertError) {
    failCount++;
    console.error(`❌ 保存块 ${i} 失败:`, insertError.message);
    console.error('详细错误:', insertError);

    if (failCount > 5) {
        throw new Error(`保存失败过多 (${failCount} 个), 停止处理`);
    }
}
}

console.log(`✅ 文档块保存完成: 成功 ${successCount}, 失败 ${failCount}`);

// 5. 更新 PDF 状态
await prisma.pdf.update({
    where: { id: pdfId },
    data: {
        status: 'ready',
        totalChunks: successCount,
        totalPages: pdfData.numpages,
        processedAt: new Date(),
    }
});

```

```

    },
  });

  console.log('✅ PDF 处理完成:', pdfId);

} catch (error) {
  console.error('❌ 后台处理失败:', error);
  console.error('错误堆栈:', error.stack);

  try {
    await prisma.pdf.update({
      where: { id: pdfId },
      data: {
        status: 'failed',
        errorMessage: error.message || '未知错误',
      },
    });
  } catch (updateError) {
    console.error('❌ 更新失败状态失败:', updateError);
  }
}
}
}

```

## 步骤 4.2: 修改 ChatPDF API (使用 RAG 检索)

✅ 修改位置 8: 修改 `app/api/chat-pdf/route.js`

```

/**
 * =====
 * ChatPDF API 路由 (app/api/chat-pdf/route.js)
 * =====
 *
 * 文件作用:
 *   处理与 PDF 文件的 AI 对话请求
 *
 * 主要功能:
 *   1. 验证用户身份
 *   2. 解析 PDF 文件内容
 *   3. 调用 AI 模型进行对话
 *   4. 返回 AI 响应
 *
 * 技术栈:
 *   - pdf-parse: PDF 文本提取
 *   - OpenRouter API: AI 模型调用
 *
 * 修改记录:
 *   - 2025-11-16: 修复 pdf-parse 导入问题 ✅ 修复
 *
 * =====
 */

```

```

// ✅ 修改点 1: 在文件顶部添加导入
import { searchSimilarChunks } from '@lib/rag/retrieval';
// ✅ 修复点 1: 修改 PDF 解析库的导入方式
import { NextResponse } from 'next/server';
import { auth } from '@app/api/auth/[...nextauth]/route';
import fs from 'fs';
import path from 'path';
import { prisma } from '@lib/prisma'; // ✅ 保持 ES modules 导入

// // ✅ 修复点 2: 使用动态导入避免模块加载问题
// let pdfParse;
// try {
//   pdfParse = require('pdf-parse');
// } catch (error) {
//   console.error('❌ pdf-parse 加载失败:', error);
// }

// ✅ 修改点 2: 更新版本标识
// ✅ 修改位置 2: 更新版本标识
console.log('🔧 使用 RAG 检索版本 - 2025-11-17 ✅');

export async function POST(request) {
  console.log('🔧 使用 RAG 检索版本 - 2025-11-17 ✅');
  console.log('📄 收到 ChatPDF 请求');

  try {
    // =====
    // 1. 身份验证（保持不变）
    // =====
    const session = await auth();
    if (!session || !session.user) {
      console.log('❌ 用户未登录');
      return NextResponse.json({ error: '请先登录' }, { status: 401 });
    }


    console.log('✅ 用户已登录:', session.user.email);

    // =====
    // 2. 解析请求参数（保持不变）
    // =====
    const { message, pdfId } = await request.json();
    console.log('📄 消息:', message);
    console.log('📄 PDF ID:', pdfId);

    if (!message?.trim()) {
      return NextResponse.json({ error: '消息不能为空' }, { status: 400 });
    }

    if (!pdfId) {
      return NextResponse.json({ error: '请先选择 PDF 文件' }, { status: 400 });
    }
  }
}

```

```
// =====
// 3.  修复点 3: 增强文件查找逻辑
// =====
// =====
//  修改位置 3: 简化文件查找逻辑（只查数据库，不再读取文件）
// =====
console.log('🔍 查询 PDF 记录...');

//  正确代码
const pdfRecord = await prisma.PDF.findFirst({
  where: {
    id: pdfId, // ← 直接使用字符串 ID
    userId: session.user.id
  }
});

if (!pdfRecord) {
  console.log('❌ PDF 记录不存在');
  return NextResponse.json({
    error: 'PDF 文件不存在或无权访问'
  }, { status: 404 });
}

console.log('✅ 找到 PDF 记录:', pdfRecord.name);

// =====
//  修改位置 4: 检查 PDF 处理状态
// =====
console.log('📄 PDF 状态:', pdfRecord.status);

if (pdfRecord.status === 'processing') {
  return NextResponse.json({
    error: 'PDF 文件正在处理中, 请稍后再试',
    status: pdfRecord.status,
  }, { status: 400 });
}

if (pdfRecord.status === 'failed') {
  return NextResponse.json({
    error: 'PDF 文件处理失败',
    details: pdfRecord.errorMessage,
  }, { status: 400 });
}

if (pdfRecord.status !== 'ready') {
  return NextResponse.json({
    error: `PDF 文件状态异常: ${pdfRecord.status}`,
  }, { status: 400 });
}

// =====
//  修改位置 6: 新增 RAG 检索逻辑
```

```
// =====
console.log('🔍 开始 RAG 检索相关内容...');

let relevantChunks = [];
try {
  relevantChunks = await searchSimilarChunks(message, {
    pdfId: pdfId,
    topK: 5,           // 检索前 5 个最相关的块
    threshold: 0.6,    // 相似度阈值
  });

  console.log(`✅ 检索到 ${relevantChunks.length} 个相关文档块`);

  if (relevantChunks.length > 0) {
    console.log('📄 相关块预览:');
    relevantChunks.forEach((chunk, index) => {
      console.log(`  ${index + 1}. 相似度: ${chunk.similarity.toFixed(3)}, 页码: ${chunk.pageNumber || 'N/A'}, 长度: ${chunk.content.length}`);
    });
  }

} catch (retrievalError) {
  console.error('❌ RAG 检索失败:', retrievalError);

  // 检索失败时降级处理
  return NextResponse.json({
    error: 'RAG 检索服务暂时不可用',
    details: retrievalError.message,
    suggestion: '请稍后重试或联系管理员',
  }, { status: 500 });
}


// 检查是否检索到相关内容
if (relevantChunks.length === 0) {
  console.log('⚠️ 未检索到相关内容');
  return NextResponse.json({
    success: true,
    response: `抱歉, 我在文档《${pdfRecord.name}》中没有找到与您的问题直接相关的内容。

    可能的原因:
    1. 您的问题可能超出了文档的主题范围
    2. 文档中可能没有涉及相关信息
    3. 可以尝试换一种方式提问


    建议:
    - 尝试使用文档中的关键词提问
    - 询问文档的整体内容或结构
    - 提出更具体的问题`,
  });
}


metadata: {
  pdfName: pdfRecord.name,
  chunksFound: 0,
  model: 'openai/gpt-4o',
  timestamp: new Date().toISOString()
}
```


```
    }  
  });  
}
```


```
//  构建上下文
```

```
const context = relevantChunks  
  .map((chunk, index) => {  
    const pageInfo = chunk.pageNumber ? ` (第 ${chunk.pageNumber} 页)` : '';  
    const similarity = (chunk.similarity * 100).toFixed(1);  
    return `[来源 ${index + 1}]${pageInfo} | 相关度: ${similarity}%\n${chunk.content}`;  
  })  
  .join('\n\n---\n\n');
```

```
console.log( 上下文长度:', context.length);
```

```
// =====  
// 5.  修复点 5: 增强 AI 调用错误处理  
// =====  
// =====
```


```
//  修改位置 7: 更新 AI 提示词 (使用 RAG 上下文)
```

```
// =====  
console.log( 开始调用 AI 模型...');
```


```
// 检查 API 密钥
```

```
if (!process.env.OPENAI_API_KEY) {  
  console.error( OpenRouter API 密钥未配置');  
  return NextResponse.json({  
    error: 'AI 服务配置错误, 请联系管理员',  
    details: 'OpenRouter API key not configured'  
  }, { status: 500 });  
}
```

```
try {
```

```
  //  新的系统提示词 (基于 RAG 检索)
```

```
  const systemPrompt = `你是一个专业的 PDF 文档分析助手。用户上传了一个名为 "${pdfRecord.name}"  
的 PDF 文件。`
```

```
##  相关文档内容 (基于语义检索):  
${context}
```

```
##  回答要求:
```

1. **\*\*基于检索内容\*\***: 优先使用上述检索到的相关内容回答问题
2. **\*\*引用来源\*\***: 回答时可以标注来源编号, 如"根据来源1..."或"第X页提到..."
3. **\*\*准确性\*\***: 如果检索内容不足以完整回答问题, 明确告知用户
4. **\*\*格式美化\*\***: 使用 Markdown 格式, 提高可读性
5. **\*\*友好语气\*\***: 保持专业、准确、友好的语气
6. **\*\*中文回答\*\***: 使用简体中文回答

```
##  文档信息:
```

- 文件名: \${pdfRecord.name}
- 总页数: \${pdfRecord.totalPages || 'N/A'}
- 文档块数: \${pdfRecord.totalChunks}
- 检索到的相关块: \${relevantChunks.length}

## ⚠️ 注意事项:

- 不要编造文档中不存在的内容
- 如果问题超出检索内容范围, 诚实告知用户
- 可以建议用户换一种方式提问`;

```
const userPrompt = `用户问题: ${message}`;
```

```
// 调用 OpenRouter API
```

```
console.log('🔗 调用 OpenRouter API...');
```

```
console.log('🔑 API 密钥前缀:', process.env.OPENAI_API_KEY?.substring(0, 10) + '...');
```

```
const response = await fetch('https://openrouter.ai/api/v1/chat/completions', {  
  method: 'POST',  
  headers: {  
    'Authorization': `Bearer ${process.env.OPENAI_API_KEY}`,  
    'Content-Type': 'application/json',  
    'X-Title': 'AI Chat App - ChatPDF with RAG',  
    'HTTP-Referer': process.env.NEXT_PUBLIC_APP_URL || 'http://localhost:3000',  
  },  
  body: JSON.stringify({  
    model: 'openai/gpt-4o',  
    messages: [  
      { role: 'system', content: systemPrompt },  
      { role: 'user', content: userPrompt }  
    ],  
    temperature: 0.7,  
    max_tokens: 2000,  
    top_p: 1,  
    frequency_penalty: 0,  
    presence_penalty: 0  
  })  
});
```

```
console.log('🔗 API 响应状态:', response.status);
```

```
if (!response.ok) {  
  const errorData = await response.text();  
  console.error('❌ OpenRouter API 错误:', response.status, errorData);  
  
  let errorMessage = 'AI 服务暂时不可用';  
  if (response.status === 401) {  
    errorMessage = 'AI 服务认证失败, 请检查 API 密钥';  
  } else if (response.status === 429) {  
    errorMessage = 'AI 服务请求过于频繁, 请稍后重试';  
  } else if (response.status === 500) {  
    errorMessage = 'AI 服务内部错误, 请稍后重试';  
  }  
  
  throw new Error(`${errorMessage} (状态码: ${response.status})`);  
}
```

```
const aiResponse = await response.json();
```

```

console.log('✅ AI API 调用成功');

const aiMessage = aiResponse.choices?.[0]?.message?.content;

if (!aiMessage) {
  console.error('❌ AI 响应为空:', aiResponse);
  throw new Error('AI 响应为空, 请重试');
}

console.log('🤖 AI 回复长度:', aiMessage.length);
console.log('🤖 AI 回复预览:', aiMessage.substring(0, 100));

// =====
// ✅ 修改位置 8: 更新返回的元数据
// =====
return NextResponse.json({
  success: true,
  response: aiMessage,
  metadata: {
    pdfName: pdfRecord.name,
    totalPages: pdfRecord.totalPages,
    totalChunks: pdfRecord.totalChunks,
    chunksRetrieved: relevantChunks.length, // ✅ 新增
    sources: relevantChunks.map(chunk => ({ // ✅ 新增: 来源信息
      pageNumber: chunk.pageNumber,
      similarity: chunk.similarity,
      preview: chunk.content.substring(0, 100) + '...'
    })),
    model: 'openai/gpt-4o',
    ragEnabled: true, // ✅ 新增: 标识使用了 RAG
    timestamp: new Date().toISOString()
  }
});

} catch (aiError) {
  console.error('❌ AI 调用失败:', aiError);
  return NextResponse.json({
    error: `AI 服务调用失败: ${aiError.message}`,
    suggestion: '请检查网络连接或稍后重试',
    debugInfo: {
      errorType: aiError.name,
      errorMessage: aiError.message
    }
  }, { status: 500 });
}

} catch (error) {
  console.error('❌ ChatPDF API 总体错误:', error);
  return NextResponse.json({
    error: '服务器内部错误',
    details: error.message,
    timestamp: new Date().toISOString()
  }, { status: 500 });
}

```

```
}  
}
```

## 步骤 4.3: 创建 PDF 状态查询 API

✓ 修改位置 9: 创建新文件 `app/api/pdf/status/route.js`

```
/**  
 * =====  
 * PDF 处理状态查询 API app/api/pdf/status/route.js  
 * =====  
 */  
  
import { NextResponse } from 'next/server';  
import { auth } from '@app/api/auth/[...nextauth]/route';  
import { prisma } from '@lib/prisma';  
  
export async function GET(req) {  
  try {  
    // 身份验证  
    const session = await auth();  
    if (!session?.user?.id) {  
      return NextResponse.json({ error: '未登录' }, { status: 401 });  
    }  
  
    // 获取 PDF ID  
    const { searchParams } = new URL(req.url);  
    const pdfId = searchParams.get('id');  
  
    if (!pdfId) {  
      return NextResponse.json({ error: '缺少 PDF ID' }, { status: 400 });  
    }  
  
    // 查询 PDF 状态  
    const pdf = await prisma.pdf.findUnique({  
      where: { id: pdfId },  
      select: {  
        id: true,  
        name: true,  
        status: true,  
        totalChunks: true,  
        totalPages: true,  
        processedAt: true,  
        errorMessage: true,  
      },  
    });  
  
    if (!pdf) {  
      return NextResponse.json({ error: 'PDF 不存在' }, { status: 404 });  
    }  
  }  
}
```

```
// 验证权限
const pdfwithUser = await prisma.pdf.findUnique({
  where: { id: pdfId },
  select: { userId: true },
});

if (pdfwithUser.userId !== session.userId) {
  return NextResponse.json({ error: '无权访问' }, { status: 403 });
}

return NextResponse.json({
  success: true,
  data: pdf,
});

} catch (error) {
  console.error('查询状态失败:', error);
  return NextResponse.json({ error: '查询失败' }, { status: 500 });
}
}
```

## 阶段 5: 前端界面集成

### 步骤 5.1: 修改 ChatPDF 页面（添加状态轮询）

✅ 修改位置 10: 修改 `app/chatpdf/page.js`


在现有代码的基础上添加以下修改：

```
/**
 * =====
 * ChatPDF 页面组件 (app/chatpdf/page.js) - 添加 Markdown 和复制功能
 * =====
 *
 * 文件作用：
 * 提供 PDF 文件上传和智能对话功能
 *
 * 主要功能：
 * 1. PDF 文件上传
 * 2. PDF 文件列表管理（侧边栏）
 * 3. 与 PDF 内容进行 AI 对话
 * 4. 模型选择（GPT-4o、Claude 等）
 * 5. ✅ 新增：Markdown 渲染支持
 * 6. ✅ 新增：复制功能
 *
 * 技术栈：
 * - Next.js 14 App Router
 * - NextAuth.js（身份验证）
 * - Tailwind CSS（样式）
 * - ✅ 新增：React Markdown（Markdown 渲染）
 */
```

```

*
* 修改记录:
*   - 2025-11-16: 使用 pdfApi 统一处理文件上传
*   - 2025-11-16: 修复 handleSendMessage 函数参数问题  新增
*   - 2025-11-16: 添加获取 PDF 列表功能  新增
*   -  2025-11-16: 添加 Markdown 渲染和复制功能
*
* =====
*/
'use client';

import { useState, useRef, useEffect } from 'react';
import { useSession } from 'next-auth/react';
import { useRouter } from 'next/navigation';
import {
  Upload,
  FileText,
  Send,
  Loader2,
  X,
  ChevronDown,
  Link as LinkIcon,
  ArrowLeft,
  RefreshCw,
  //  修改位置 1: 新增复制相关图标
  Copy,
  Check
} from 'lucide-react';
// 修改点 1: 导入 pdfApi
import { pdfApi } from '@lib/api-client';

//  修改位置 2: 新增 Markdown 相关导入
import ReactMarkdown from 'react-markdown';
import remarkGfm from 'remark-gfm';

export default function ChatPDF() {
  const { data: session, status } = useSession();
  const router = useRouter();

  // =====
  // 状态管理
  // =====
  // PDF 文件相关
  const [pdfFiles, setPdfFiles] = useState([]); // PDF 文件列表
  const [currentPdf, setCurrentPdf] = useState(null); // 当前选中的 PDF
  const [isUploading, setIsUploading] = useState(false); // 上传中
  const [isDragging, setIsDragging] = useState(false); // 拖拽状态
  const [isLoadingList, setIsLoadingList] = useState(false); // 列表加载状态

  //  修改位置 1: 新增状态
  const [pdfStatus, setPdfStatus] = useState({}); // PDF 处理状态映射

  // 修改点 2: 添加错误和成功提示状态

```

```

const [uploadError, setUploadError] = useState(''); // 上传错误信息
const [uploadSuccess, setUploadSuccess] = useState(''); // 上传成功信息

// 聊天相关
const [messages, setMessages] = useState([]); // 聊天消息
const [inputMessage, setInputMessage] = useState(''); // 输入框内容
const [isGenerating, setIsGenerating] = useState(false); // AI 生成中

// ✅ 修改位置 3: 新增复制功能状态
const [copiedMessageId, setCopiedMessageId] = useState(null); // 复制功能状态

// 模型选择
const [selectedModel, setSelectedModel] = useState('openai/gpt-4o');
const [showModelDropdown, setShowModelDropdown] = useState(false);

// URL 上传
const [showUrlInput, setShowUrlInput] = useState(false);
const [urlInput, setUrlInput] = useState('');

// Refs
const fileInputRef = useRef(null);
const messagesEndRef = useRef(null);
const dropZoneRef = useRef(null);
const modelDropdownRef = useRef(null);

// =====
// 模型列表
// =====
const models = [
  { id: 'openai/gpt-4o', name: 'GPT-4o', icon: '🚀', description: '最强大的模型' },
  { id: 'openai/gpt-4o-mini', name: 'GPT-4o Mini', icon: '⚡', description: '快速且经济' },
  { id: 'anthropic/claude-3.5-sonnet', name: 'Claude 3.5 Sonnet', icon: '🧠',
description: '擅长分析' },
  { id: 'google/gemini-pro-1.5', name: 'Gemini Pro 1.5', icon: '💎', description: '长文本处理' },
];

// =====
// ✅ 修改位置 4: 新增复制到剪贴板功能
// =====
const copyToClipboard = async (text, messageId) => {
  try {
    await navigator.clipboard.writeText(text);
    setCopiedMessageId(messageId);
    setTimeout(() => setCopiedMessageId(null), 2000);
  } catch (error) {
    console.error('复制失败:', error);
  }
};

// =====
// ✅ 修改位置 2: 新增状态轮询函数
// =====

```

```

const checkPdfStatus = async (pdfId) => {
  try {
    console.log('🔍 查询 PDF 状态:', pdfId);

    const response = await fetch(`/api/pdf/status?id=${pdfId}`);
    const result = await response.json();

    console.log('📊 状态查询结果:', result);

    if (result.success) {
      setPdfStatus(prev => ({
        ...prev,
        [pdfId]: result.data.status,
      }));

      return result.data.status;
    }
  } catch (error) {
    console.error('❌ 查询状态失败:', error);
  }
  return null;
};

// =====
// 权限检查
// =====
useEffect(() => {
  if (status === 'unauthenticated') {
    router.push('/login');
  }
}, [status, router]);

// =====
// 修改点 3: 添加获取 PDF 列表的 useEffect
// =====
// 修改点 3: 添加获取 PDF 列表的 useEffect
// =====
useEffect(() => {
  const fetchPdfList = async () => {
    try {
      console.log('📁 开始获取 PDF 列表...');
      setIsLoadingList(true);

      const response = await fetch('/api/pdf/list', {
        method: 'GET',
        headers: {
          'Content-Type': 'application/json',
        },
      });

      console.log('📡 API 响应状态:', response.status);
    }
  };

```

```

    if (!response.ok) {
      throw new Error(`HTTP error! status: ${response.status}`);
    }

    const result = await response.json();
    console.log('📁 获取到的 PDF 列表:', result);

    if (result.success) {
      // ✅ 修改位置 3: 转换数据格式, 添加状态字段
      const formattedPdfFiles = result.data.map(pdf => ({
        id: pdf.id,
        name: pdf.name || pdf.fileName,
        size: pdf.size,
        url: pdf.filePath,
        uploadedAt: new Date(pdf.createdAt),
        status: pdf.status, // ✅ 新增: 保存状态
      }));

      setPdfFiles(formattedPdfFiles);
      console.log('✅ PDF 列表设置成功, 数量:', formattedPdfFiles.length);

      // ✅ 修改位置 3.1: 初始化状态映射
      const statusMap = {};
      formattedPdfFiles.forEach(pdf => {
        statusMap[pdf.id] = pdf.status;
      });
      setPdfStatus(statusMap);

      // 如果有文件且没有选中的 PDF, 自动选中第一个
      if (formattedPdfFiles.length > 0 && !currentPdf) {
        setCurrentPdf(formattedPdfFiles[0]);
        console.log('✅ 自动选中第一个 PDF:', formattedPdfFiles[0].name);
      }
    } else {
      console.error('❌ API 返回失败:', result.error);
      setPdfFiles([]);
    }
  } catch (error) {
    console.error('❌ 获取 PDF 列表失败:', error);
    setPdfFiles([]);
  } finally {
    setIsLoadingList(false);
  }
};

// 只在用户登录后获取列表
if (session?.user) {
  fetchPdfList();
}
}, [session]); // 依赖 session, 当用户登录状态改变时重新获取

// =====
// 修改点 4: 添加刷新列表的函数

```

```
// =====
const refreshPdfList = async () => {
  try {
    console.log('🔄 刷新 PDF 列表...');
    setIsLoadingList(true);

    const response = await fetch('/api/pdf/list');
    const result = await response.json();

    if (result.success) {
      const formattedPdfFiles = result.data.map(pdf => ({
        id: pdf.id,
        name: pdf.name || pdf.fileName,
        size: pdf.size,
        url: pdf.filePath,
        uploadedAt: new Date(pdf.createdAt)
      }));

      setPdfFiles(formattedPdfFiles);
      console.log('✅ 列表刷新成功, 数量:', formattedPdfFiles.length);
    }
  } catch (error) {
    console.error('❌ 刷新列表失败:', error);
  } finally {
    setIsLoadingList(false);
  }
};

// =====
// 自动滚动到底部
// =====
useEffect(() => {
  messagesEndRef.current?.scrollIntoView({ behavior: 'smooth' });
}, [messages]);

// =====
// 点击外部关闭下拉菜单
// =====
useEffect(() => {
  const handleClickOutside = (event) => {
    if (modelDropdownRef.current && !modelDropdownRef.current.contains(event.target)) {
      setShowModelDropdown(false);
    }
  };

  document.addEventListener('mousedown', handleClickOutside);
  return () => document.removeEventListener('mousedown', handleClickOutside);
}, []);

// =====
// 修改点 5: 修改文件上传处理函数, 上传成功后刷新列表
// =====
// =====
```

```
// 修改点 5: 修改文件上传处理函数, 上传成功后刷新列表
// =====
const handleFileUpload = async (file) => {
  console.log('📁 开始上传文件:', file.name);

  // 清空之前的提示信息
  setUploadError('');
  setUploadSuccess('');

  // 验证文件类型
  if (file.type !== 'application/pdf') {
    const errorMsg = '仅支持 PDF 文件上传';
    setUploadError(errorMsg);
    console.error('❌', errorMsg);
    return;
  }

  // 验证文件大小 (最大 20MB, 与后端一致)
  if (file.size > 20 * 1024 * 1024) {
    const errorMsg = '文件大小不能超过 20MB';
    setUploadError(errorMsg);
    console.error('❌', errorMsg);
    return;
  }

  setIsuploading(true);

  try {
    // 使用 pdfApi.upload() 上传文件
    console.log('📁 调用 pdfApi.upload()');
    const result = await pdfApi.upload(file);

    console.log('✅ 上传成功:', result);

    // ✅ 修改位置 4: 上传成功后的处理
    if (result.success) {
      const uploadedPdf = {
        id: result.data.id,
        name: result.data.name,
        size: result.data.size,
        url: result.data.filePath,
        uploadedAt: new Date(result.data.createdAt || Date.now()),
        status: result.data.status || 'processing', // ✅ 新增: 保存状态
      };

      // ✅ 修改位置 4.1: 初始化状态
      setPdfStatus(prev => ({
        ...prev,
        [uploadedPdf.id]: uploadedPdf.status,
      }));

      // 刷新列表
      await refreshPdfList();
    }
  }
}
```

```

// 设置为当前 PDF
setCurrentPdf(uploadedPdf);

// 清空聊天记录
setMessages([]);

//  修改位置 4.2: 启动状态轮询
if (uploadedPdf.status === 'processing') {
  setUploadSuccess('文件上传成功, 正在处理中...');

  const pollInterval = setInterval(async () => {
    const status = await checkPdfStatus(uploadedPdf.id);

    console.log('📄 轮询状态:', status);

    if (status === 'ready' || status === 'failed') {
      clearInterval(pollInterval);

      if (status === 'ready') {
        setUploadSuccess('文件处理完成, 可以开始对话! ');
        console.log('✅ PDF 处理完成');
      } else {
        setUploadError('文件处理失败, 请重新上传');
        console.error('❌ PDF 处理失败');
      }
    }

    // 刷新列表
    await refreshPdfList();
  }, 2000); // 每 2 秒检查一次

  // 5 分钟后停止轮询
  setTimeout(() => {
    clearInterval(pollInterval);
    console.log('⌚ 轮询超时, 停止检查');
  }, 300000);
} else {
  setUploadSuccess('文件上传成功! ');
}

console.log('✅ 文件上传成功:', uploadedPdf);

// 3秒后自动清除成功提示 (如果不是 processing 状态)
if (uploadedPdf.status !== 'processing') {
  setTimeout(() => {
    setUploadSuccess('');
  }, 3000);
}
}

} catch (error) {
  console.error('❌ 上传失败:', error);
}

```

```

        setUploadError(error.message || '文件上传失败，请稍后重试');
    } finally {
        setIsUploading(false);
    }
};

// =====
// 文件选择处理
// =====
const handleFileSelect = (e) => {
    const file = e.target.files[0];
    if (file) {
        handleFileUpload(file);
    }
};

// =====
// 拖拽上传处理
// =====
const handleDragOver = (e) => {
    e.preventDefault();
    setIsDragging(true);
};

const handleDragLeave = (e) => {
    e.preventDefault();
    setIsDragging(false);
};

const handleDrop = (e) => {
    e.preventDefault();
    setIsDragging(false);

    const file = e.dataTransfer.files[0];
    if (file) {
        handleFileUpload(file);
    }
};

// =====
// 修改点 6: 修改 URL 上传处理，成功后刷新列表
// =====
const handleUrlUpload = async () => {
    if (!urlInput.trim()) {
        setUploadError('请输入 PDF 链接');
        return;
    }

    setUploadError('');
    setUploadSuccess('');
    setIsUploading(true);

    try {

```

```

const response = await fetch('/api/upload/pdf-url', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({ url: urlInput }),
});

const result = await response.json();

if (result.success) {
  // 修改点: 上传成功后刷新列表
  await refreshPdfList();

  const newPdf = {
    id: result.data.id || Date.now().toString(),
    name: result.data.filename || result.data.name,
    size: result.data.size,
    url: result.data.url || result.data.filePath,
    uploadedAt: new Date()
  };

  setCurrentPdf(newPdf);
  setMessages([]);
  setShowUrlInput(false);
  setUrlInput('');

  setUploadSuccess('文件上传成功! ');
  setTimeout(() => setUploadSuccess(''), 3000);
} else {
  setUploadError(result.error || '文件上传失败');
}
} catch (error) {
  console.error('上传失败:', error);
  setUploadError('文件上传失败, 请稍后重试');
} finally {
  setIsUploading(false);
}
};

// =====
// 修改点 7: 完全重写 handleMessage 函数
// =====
const handleMessage = async () => {
  console.log('🚀 开始发送消息:', inputMessage);

  if (!inputMessage.trim()) return;

  if (!currentPdf) {
    setUploadError('请先上传 PDF 文件');
    return;
  }

  // ✅ 修改位置 5: 检查处理状态
  const status = pdfStatus[currentPdf.id] || currentPdf.status;

```

```

console.log('📄 当前 PDF 状态:', status);

if (status === 'processing') {
  setUploadError('PDF 正在处理中, 请稍后再试');
  return;
}

if (status === 'failed') {
  setUploadError('PDF 处理失败, 请重新上传');
  return;
}

if (status !== 'ready') {
  // ✅ 修改位置 5.1: 主动查询最新状态
  console.log('🔍 状态未知, 主动查询...');
  const latestStatus = await checkPdfStatus(currentPdf.id);

  if (latestStatus === 'processing') {
    setUploadError('PDF 正在处理中, 请稍后再试');
    return;
  }

  if (latestStatus === 'failed') {
    setUploadError('PDF 处理失败, 请重新上传');
    return;
  }

  if (latestStatus !== 'ready') {
    setUploadError('PDF 状态异常, 请刷新页面或重新上传');
    return;
  }
}

const userMessage = {
  id: Date.now().toString(),
  role: 'user',
  content: inputMessage,
  timestamp: new Date()
};

setMessages(prev => [...prev, userMessage]);
setInputMessage('');
setIsGenerating(true);

try {
  console.log('📤 发送请求到 /api/chat-pdf');
  console.log('📋 请求参数:', {
    message: inputMessage,
    pdfId: currentPdf.id
  });

  // 修复: 使用正确的参数格式

```

```

const response = await fetch('/api/chat-pdf', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    message: inputMessage,
    pdfId: currentPdf.id // 使用 pdfId 而不是 pdfurl
  }),
});

console.log('📡 收到响应状态:', response.status);

if (!response.ok) {
  throw new Error(`API 请求失败: ${response.status}`);
}

const data = await response.json();
console.log('📄 响应数据:', data);

// 处理响应
if (data.success) {
  const aiMessage = {
    id: (Date.now() + 1).toString(),
    role: 'assistant',
    content: data.response,
    timestamp: new Date()
  };

  setMessages(prev => [...prev, aiMessage]);
  console.log('✅ 消息添加成功');
} else {
  throw new Error(data.error || '未知错误');
}

} catch (error) {
  console.error('❌ 发送消息失败:', error);

  const errorMessage = {
    id: (Date.now() + 1).toString(),
    role: 'assistant',
    content: `抱歉, 发生了错误: ${error.message}`,
    timestamp: new Date()
  };

  setMessages(prev => [...prev, errorMessage]);
  setUploadError(`发送消息失败: ${error.message}`);
} finally {
  setIsGenerating(false);
}
};

// =====
// 切换 PDF
// =====

```

```

const handleSelectPdf = (pdf) => {
  setCurrentPdf(pdf);
  setMessages([]);
  setUploadError('');
  setUploadSuccess('');
};

// =====
// 修改点 8: 修改删除 PDF, 删除后刷新列表
// =====
const handleDeletePdf = async (pdfId) => {
  if (!confirm('确定要删除这个 PDF 文件吗? ')) {
    return;
  }

  try {
    console.log('🗑️ 开始删除 PDF:', pdfId);

    // 调用后端 API 删除文件 (如果有的话)
    const response = await fetch(`/api/pdf/delete`, {
      method: 'DELETE',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ id: pdfId }),
    });

    if (response.ok) {
      console.log('✅ 后端删除成功');
    } else {
      console.log('⚠️ 后端删除失败, 但继续前端删除');
    }

    // 刷新列表
    await refreshPdfList();

    // 如果删除的是当前 PDF, 清空状态
    if (currentPdf?.id === pdfId) {
      setCurrentPdf(null);
      setMessages([]);
    }

    console.log('✅ PDF 删除成功:', pdfId);
  } catch (error) {
    console.error('❌ 删除失败:', error);
    setUploadError('删除失败, 请稍后重试');
  }
};

// =====
// 格式化文件大小
// =====
const formatFileSize = (bytes) => {

```

```

    if (bytes < 1024) return bytes + ' B';
    if (bytes < 1024 * 1024) return (bytes / 1024).toFixed(2) + ' KB';
    return (bytes / (1024 * 1024)).toFixed(2) + ' MB';
  };

  // =====
  // 格式化日期
  // =====
  const formatDate = (date) => {
    const d = new Date(date);
    const month = String(d.getMonth() + 1).padStart(2, '0');
    const day = String(d.getDate()).padStart(2, '0');
    const year = d.getFullYear();
    return `${month}/${day}/${year}`;
  };

  // =====
  // 处理键盘事件 (Enter 发送)
  // =====
  const handleKeyPress = (e) => {
    if (e.key === 'Enter' && !e.shiftKey) {
      e.preventDefault();
      handleSendMessage();
    }
  };

  // =====
  // 渲染
  // =====
  if (status === 'loading') {
    return (
      <div className="flex items-center justify-center h-screen">
        <Loader2 className="w-8 h-8 animate-spin text-blue-600" />
      </div>
    );
  }

  return (
    <div className="flex h-screen bg-gray-50">
      {/* =====
        左侧边栏 - PDF 文件列表 (保持不变)
        ===== */}
      <div className="w-64 bg-white border-r border-gray-200 flex flex-col">
        {/* 修改点 9: 修改头部, 添加刷新按钮 */}
        <div className="p-4 border-b border-gray-200">
          <div className="flex items-center justify-between mb-4">
            <button
              onClick={() => router.push('/') }
              className="flex items-center text-gray-600 hover:text-gray-900 transition-
colors"
            >
              <ArrowLeft className="w-5 h-5 mr-2" />
              <span className="font-medium">返回主页</span>

```

```

</button>

{/* 添加刷新按钮 */}
<button
  onClick={refreshPdfList}
  disabled={isLoadingList}
  className="p-2 text-gray-600 hover:text-gray-900 transition-colors rounded-lg
hover:bg-gray-100 disabled:opacity-50"
  title="刷新列表"
>
  <RefreshCw className={`w-4 h-4 ${isLoadingList ? 'animate-spin' : ''}`} />
</button>
</div>

<h1 className="text-xl font-bold text-gray-900 flex items-center">
  <FileText className="w-6 h-6 mr-2 text-red-500" />
  PDF 工具
</h1>
</div>

{/* 修改点 10: 修改 PDF 文件列表, 添加加载状态 */}
<div className="flex-1 overflow-y-auto custom-scrollbar p-4">
  <div className="flex items-center justify-between mb-3">
    <h2 className="text-sm font-semibold text-gray-500">最近使用</h2>
    {isLoadingList && (
      <Loader2 className="w-4 h-4 animate-spin text-gray-400" />
    )}
  </div>

  {isLoadingList ? (
    /* 加载状态 */
    <div className="text-center text-gray-400 text-sm mt-8">
      <Loader2 className="w-8 h-8 mx-auto mb-2 animate-spin" />
      <p>加载中...</p>
    </div>
  ) : pdfFiles.length === 0 ? (
    <div className="text-center text-gray-400 text-sm mt-8">
      <FileText className="w-12 h-12 mx-auto mb-2 opacity-50" />
      <p>暂无 PDF 文件</p>
    </div>
  ) : (
    <div className="space-y-2">
      {pdfFiles.map((pdf) => (
        <div
          key={pdf.id}
          onClick={() => handleSelectPdf(pdf)}
          className={`group relative p-3 rounded-lg cursor-pointer transition-all ${
            currentPdf?.id === pdf.id
              ? 'bg-blue-50 border-2 border-blue-500'
              : 'bg-gray-50 hover:bg-gray-100 border-2 border-transparent'
          }`}
        >

```

{pdf.name}>

```
<div className="flex items-start">
  <FileText className={`w-5 h-5 mr-2 flex-shrink-0 ${
    currentPdf?.id === pdf.id ? 'text-blue-600' : 'text-red-500'
 }`} />
  <div className="flex-1 min-w-0">
    <p className="text-sm font-medium text-gray-900 truncate" title=

      {pdf.name}
    </p>
    <p className="text-xs text-gray-500 mt-1">
      {formatFileSize(pdf.size)}
    </p>
    <p className="text-xs text-gray-400 mt-0.5">
      {formatDate(pdf.uploadedAt)}
    </p>

    /*  修改位置 6: 新增状态标签 */
    {pdfStatus[pdf.id] === 'processing' && (
      <div className="flex items-center mt-1">
        <Loader2 className="w-3 h-3 mr-1 animate-spin text-blue-500" />
        <span className="text-xs text-blue-600">处理中...</span>
      </div>
    )}

    {pdfStatus[pdf.id] === 'failed' && (
      <div className="flex items-center mt-1">
        <X className="w-3 h-3 mr-1 text-red-500" />
        <span className="text-xs text-red-600">处理失败</span>
      </div>
    )}

    {pdfStatus[pdf.id] === 'ready' && (
      <div className="flex items-center mt-1">
        <Check className="w-3 h-3 mr-1 text-green-500" />
        <span className="text-xs text-green-600">就绪</span>
      </div>
    )}
  </div>

  /*  修改位置 6.1: 状态指示器（右上角） */
  {pdfStatus[pdf.id] === 'processing' && (
    <div className="absolute top-2 right-2">
      <Loader2 className="w-4 h-4 animate-spin text-blue-500" />
    </div>
  )}

  {pdfStatus[pdf.id] === 'failed' && (
    <div className="absolute top-2 right-2">
      <X className="w-4 h-4 text-red-500" />
    </div>
  )}

  /* 删除按钮 */
```

```

        <button
          onClick={e => {
            e.stopPropagation();
            handleDeletePdf(pdf.id);
          }}
          className="opacity-0 group-hover:opacity-100 transition-opacity ml-2"
        >
          <X className="w-4 h-4 text-gray-400 hover:text-red-500" />
        </button>
      </div>
    </div>
  )})
</div>
})
</div>
</div>

{/* =====
  右侧主内容区
===== */}
<div className="flex-1 flex flex-col">
  {/* 顶部工具栏（保持不变） */}
  <div className="bg-white border-b border-gray-200 p-4">
    <div className="flex items-center justify-between max-w-5xl mx-auto">
      {/* 左侧：标题和图标 */}
      <div className="flex items-center">
        <div className="w-12 h-12 bg-gradient-to-br from-red-500 to-pink-500 rounded-
xl flex items-center justify-center mr-3 shadow-lg">
          <FileText className="w-6 h-6 text-white" />
        </div>
        <div>
          <h1 className="text-2xl font-bold text-gray-900">在线 ChatPDF</h1>
          <p className="text-sm text-gray-500">使用 Chat AI 能力帮助你更好的阅读</p>
        </div>
      </div>

      {/* 右侧：模型选择下拉菜单 */}
      <div className="relative" ref={modelDropdownRef}>
        <button
          onClick={() => setShowModelDropdown(!showModelDropdown)}
          className="flex items-center space-x-2 px-4 py-2 bg-gray-100 hover:bg-gray-
200 rounded-lg transition-colors"
        >
          <span className="text-2xl">
            {models.find(m => m.id === selectedModel)?.icon}
          </span>
          <span className="font-medium text-gray-900">
            {models.find(m => m.id === selectedModel)?.name}
          </span>
          <ChevronDown className={`w-4 h-4 text-gray-500 transition-transform ${
            showModelDropdown ? 'rotate-180' : ''
          }`} />
        </button>

```

```

    {/* 下拉菜单 */}
    {showModelDropdown && (
      <div className="absolute right-0 mt-2 w-64 bg-white rounded-lg shadow-xl
border border-gray-200 py-2 z-50">
        {models.map((model) => (
          <button
            key={model.id}
            onClick={() => {
              setSelectedModel(model.id);
              setShowModelDropdown(false);
            }}
            className={`w-full px-4 py-3 text-left hover:bg-gray-50 transition-
colors flex items-start ${
              selectedModel === model.id ? 'bg-blue-50' : ''
            }`}
          >
            <span className="text-2xl mr-3">{model.icon}</span>
            <div className="flex-1">
              <div className="font-medium text-gray-900">{model.name}</div>
              <div className="text-xs text-gray-500 mt-0.5">{model.description}
</div>

            </div>
            {selectedModel === model.id && (
              <div className="w-2 h-2 bg-blue-600 rounded-full mt-2"></div>
            )}
          </button>
        )]}
      </div>
    )}
  </div>
</div>
</div>
</div>

{/* 主内容区 */}
<div className="flex-1 overflow-y-auto custom-scrollbar">
  <div className="max-w-5xl mx-auto p-6">
    {/* 修改点 11: 添加错误和成功提示 */}
    {uploadError && (
      <div className="mb-4 p-4 bg-red-50 border border-red-200 rounded-lg flex
items-start">
        <X className="w-5 h-5 text-red-500 mr-3 flex-shrink-0 mt-0.5" />
        <div className="flex-1">
          <p className="text-sm text-red-800 font-medium">上传失败</p>
          <p className="text-sm text-red-600 mt-1">{uploadError}</p>
        </div>
        <button
          onClick={() => setUploadError('')}
          className="text-red-400 hover:text-red-600 transition-colors"
        >
          <X className="w-4 h-4" />
        </button>
      </div>
    )}
  </div>

```

```

    })

    {uploadSuccess && (
      <div className="mb-4 p-4 bg-green-50 border border-green-200 rounded-lg flex
items-start">
        <div className="w-5 h-5 bg-green-500 rounded-full flex items-center justify-
center mr-3 flex-shrink-0 mt-0.5">
          <svg className="w-3 h-3 text-white" fill="none" viewBox="0 0 24 24"
stroke="currentColor">
            <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M5
13 14 4 19 7" />
          </svg>
        </div>
        <div className="flex-1">
          <p className="text-sm text-green-800 font-medium">上传成功</p>
          <p className="text-sm text-green-600 mt-1">{uploadSuccess}</p>
        </div>
        <button
          onClick={() => setUploadSuccess('')}
          className="text-green-400 hover:text-green-600 transition-colors"
        >
          <X className="w-4 h-4" />
        </button>
      </div>
    })

    {!currentPdf ? (
      /* =====
      无 PDF 时显示上传区域（保持不变）
      ===== */
      <div className="flex items-center justify-center min-h-[500px]">
        <div className="w-full max-w-2xl">
          {/* 拖拽上传区域 */}
          <div
            ref={dropZoneRef}
            onDragOver={handleDragOver}
            onDragLeave={handleDragLeave}
            onDrop={handleDrop}
            className={`border-2 border-dashed rounded-2xl p-12 text-center
transition-all ${
              isDragging
                ? 'border-blue-500 bg-blue-50'
                : 'border-gray-300 bg-white hover:border-gray-400'
            }}`
          >
            <div className="w-20 h-20 bg-gradient-to-br from-red-500 to-pink-500
rounded-2xl flex items-center justify-center mx-auto mb-6 shadow-lg">
              <FileText className="w-10 h-10 text-white" />
            </div>

            <h3 className="text-xl font-semibold text-gray-900 mb-2">
              上传 PDF 文件
            </h3>
          </div>
        </div>
      </div>
    )
  }

```

```
<p className="text-gray-500 mb-6">
```

```
  点击或拖拽到此处上传
```

```
</p>
```

```
{/* 上传按钮 */}
```

```
<button
```

```
  onClick={() => fileInputRef.current?.click()}
```

```
  disabled={isUploading}
```

```
  className="inline-flex items-center px-6 py-3 bg-gradient-to-r from-blue-600 to-purple-600 text-white font-medium rounded-lg hover:from-blue-700 hover:to-purple-700 transition-all shadow-lg hover:shadow-xl disabled:opacity-50 disabled:cursor-not-allowed mb-4"
```

```
>
```

```
  {isUploading ? (
```

```
    <>
```

```
      <Loader2 className="w-5 h-5 mr-2 animate-spin" />
```

```
      上传中...
```

```
    </>
```

```
  ) : (
```

```
    <>
```

```
      <Upload className="w-5 h-5 mr-2" />
```

```
      选择文件上传
```

```
    </>
```

```
  )}
```

```
</button>
```

```
{/* 隐藏的文件输入 */}
```

```
<input
```

```
  ref={fileInputRef}
```

```
  type="file"
```

```
  accept=".pdf,application/pdf"
```

```
  onChange={handleFileSelect}
```

```
  className="hidden"
```

```
/>
```

```
{/* URL 上传按钮 */}
```

```
<div className="mt-4">
```

```
  {!showUrlInput ? (
```

```
    <button
```

```
      onClick={() => setShowUrlInput(true)}
```

```
      className="inline-flex items-center text-blue-600 hover:text-blue-
```

```
700 font-medium"
```

```
>
```

```
    <LinkIcon className="w-4 h-4 mr-2" />
```

```
    使用链接上传
```

```
  </button>
```

```
) : (
```

```
  <div className="flex items-center space-x-2 max-w-md mx-auto">
```

```
    <input
```

```
      type="url"
```

```
      value={urlInput}
```

```
      onChange={(e) => setUrlInput(e.target.value)}
```

```
      placeholder="输入 PDF 链接"
```

```

        className="flex-1 px-4 py-2 border border-gray-300 rounded-lg
focus:outline-none focus:ring-2 focus:ring-blue-500"
      />
      <button
        onClick={handleUrlUpload}
        disabled={isUploading}
        className="px-4 py-2 bg-blue-600 text-white rounded-lg hover:bg-
blue-700 transition-colors disabled:opacity-50"
      >
        上传
      </button>
      <button
        onClick={() => {
          setShowUrlInput(false);
          setUrlInput('');
        }}
        className="px-4 py-2 bg-gray-200 text-gray-700 rounded-lg
hover:bg-gray-300 transition-colors"
      >
        取消
      </button>
    </div>
  )}
</div>

  { /* 提示信息 */ }
  <p className="text-sm text-gray-400 mt-6">
    支持的文件类型: <span className="font-medium">PDF</span> |
    最大文件大小: <span className="font-medium">20MB</span>
  </p>
</div>
</div>
</div>
) : (
  /* =====
   修改位置 5: 有 PDF 时显示聊天界面 - 添加 Markdown 和复制功能
  ===== */
  <div className="flex flex-col h-full">
    { /* 当前 PDF 信息 */ }
    <div className="bg-white rounded-lg shadow-sm border border-gray-200 p-4 mb-
4">
      <div className="flex items-center">
        <FileText className="w-6 h-6 text-red-500 mr-3" />
        <div className="flex-1">
          <h3 className="font-medium text-gray-900">{currentPdf.name}</h3>
          <p className="text-sm text-gray-500">
            {formatFileSize(currentPdf.size)} •
            {formatDate(currentPdf.uploadedAt)}
          </p>
        </div>
      </div>
      <button
        onClick={() => {
          setCurrentPdf(null);

```

```

        setMessages([]);
        setUploadError('');
        setUploadSuccess('');
    }}
    className="text-gray-400 hover:text-gray-600 transition-colors"
  >
    <X className="w-5 h-5" />
  </button>
</div>
</div>

{/* 聊天消息区域 */}
<div className="flex-1 bg-white rounded-lg shadow-sm border border-gray-200
mb-4 overflow-hidden flex flex-col">
  {messages.length === 0 ? (
    /* 空状态 */
    <div className="flex-1 flex items-center justify-center p-8">
      <div className="text-center">
        <div className="w-16 h-16 bg-gradient-to-br from-blue-500 to-purple-
500 rounded-2xl flex items-center justify-center mx-auto mb-4 shadow-lg">
          <FileText className="w-8 h-8 text-white" />
        </div>
        <h3 className="text-lg font-semibold text-gray-900 mb-2">
          开始与 PDF 对话
        </h3>
        <p className="text-gray-500 mb-4">
          问我关于这个 PDF 的任何问题
        </p>
        <div className="flex flex-wrap gap-2 justify-center">
          <button
            onClick={() => setInputMessage('这个文档的主要内容是什么? ')}
            className="px-4 py-2 bg-gray-100 hover:bg-gray-200 rounded-lg
text-sm text-gray-700 transition-colors"
          >
            📄 文档主要内容
          </button>
          <button
            onClick={() => setInputMessage('请总结这个文档的关键要点')}
            className="px-4 py-2 bg-gray-100 hover:bg-gray-200 rounded-lg
text-sm text-gray-700 transition-colors"
          >
            ✨ 总结关键要点
          </button>
          <button
            onClick={() => setInputMessage('这个文档中有哪些重要的数据或统计信
息? ')}
            className="px-4 py-2 bg-gray-100 hover:bg-gray-200 rounded-lg
text-sm text-gray-700 transition-colors"
          >
            📊 数据统计
          </button>
        </div>
      </div>
    </div>
  )}

```

```

    </div>
  ) : (
    /* ✅ 修改位置 6: 消息列表 - 添加 Markdown 渲染和复制功能 */
    <div className="flex-1 overflow-y-auto custom-scrollbar p-6">
      {messages.map((message) => (
        <div
          key={message.id}
          className={`mb-6 flex ${
            message.role === 'user' ? 'justify-end' : 'justify-start'
          }`}
        >
          <div
            className={`max-w-3xl rounded-2xl px-6 py-4 relative group ${
              message.role === 'user'
                ? 'bg-blue-600 text-white'
                : 'bg-gray-100 text-gray-900'
            }`}
          >
            {/* ✅ 新增: 复制按钮 (仅 AI 消息显示) */}
            {message.role === 'assistant' && (
              <button
                onClick={() => copyToClipboard(message.content, message.id)}
                className="absolute top-2 right-2 opacity-0 group-
hover:opacity-100 transition-opacity p-1.5 hover:bg-gray-200 rounded-lg"
                title="复制消息"
              >
                {copiedMessageId === message.id ? (
                  <Check className="w-4 h-4 text-green-500" />
                ) : (
                  <Copy className="w-4 h-4 text-gray-400" />
                )}
              </button>
            )}
          </div>
          <div className="pr-8">
            {message.role === 'user' ? (
              /* 用户消息: 普通文本 */
              <div className="whitespace-pre-wrap break-words">
                {message.content}
              </div>
            ) : (
              /* ✅ AI 消息: Markdown 渲染 */
              <div className="prose prose-sm max-w-none prose-slate
prose-headings:text-gray-900 prose-headings:font-semibold
prose-headings:mb-3
prose-p:text-gray-700 prose-p:leading-relaxed prose-p:mb-4
prose-strong:text-gray-900 prose-strong:font-semibold
prose-code:text-blue-600 prose-code:bg-blue-50 prose-
code:px-2 prose-code:py-1 prose-code:rounded prose-code:text-sm
prose-pre:bg-gray-900 prose-pre:text-gray-100 prose-pre:p-
4 prose-pre:rounded-lg prose-pre:overflow-x-auto"
            >

```

```

        prose-blockquote:border-l-4 prose-blockquote:border-blue-
500 prose-blockquote:bg-blue-50 prose-blockquote:py-3 prose-blockquote:px-4 prose-
blockquote:rounded-r-lg prose-blockquote:my-4
        prose-ul:text-gray-700 prose-ul:mb-4 prose-ol:text-gray-
700 prose-ol:mb-4
        prose-li:text-gray-700 prose-li:my-1 prose-li:leading-
relaxed
        prose-a:text-blue-600 prose-a:no-underline hover:prose-
a:underline
        prose-table:border-collapse prose-table:border prose-
table:border-gray-300
        prose-th:border prose-th:border-gray-300 prose-th:bg-gray-
50 prose-th:px-4 prose-th:py-2 prose-th:text-left prose-th:font-semibold
        prose-td:border prose-td:border-gray-300 prose-td:px-4
prose-td:py-2">

        <ReactMarkdown remarkPlugins={[remarkGfm]}>
            {message.content}
        </ReactMarkdown>
    </div>
    )}
</div>

    {/*  新增：时间戳 */}
    <div className={`text-xs mt-2 ${
        message.role === 'user' ? 'text-blue-100' : 'text-gray-400'
    }`}>
        {new Date(message.timestamp).toLocaleTimeString()}
    </div>
</div>
    )})

    {/* 加载动画 */}
    {isGenerating && (
        <div className="mb-6 flex justify-start">
            <div className="max-w-3xl rounded-2xl px-6 py-4 bg-gray-100">
                <div className="flex items-center space-x-2">
                    <div className="w-2 h-2 bg-gray-400 rounded-full animate-
bounce" style={{ animationDelay: '0ms' }}></div>
                    <div className="w-2 h-2 bg-gray-400 rounded-full animate-
bounce" style={{ animationDelay: '150ms' }}></div>
                    <div className="w-2 h-2 bg-gray-400 rounded-full animate-
bounce" style={{ animationDelay: '300ms' }}></div>
                </div>
            </div>
        </div>
    )}

    <div ref={messagesEndRef} />
</div>
    )}
</div>

```

```

    { /* 输入区域（保持不变） */ }
    <div className="bg-white rounded-lg shadow-sm border border-gray-200 p-4">
      <div className="flex items-end space-x-3">
        <textarea
          value={inputMessage}
          onChange={(e) => setInputMessage(e.target.value)}
          onKeyDown={handleKeyPress}
          placeholder="输入你的问题..."
          rows={1}
          className="flex-1 resize-none px-4 py-3 border border-gray-300
rounded-lg focus:outline-none focus:ring-2 focus:ring-blue-500 focus:border-transparent"
          style={{ minHeight: '52px', maxHeight: '150px' }}
        />
        <button
          onClick={handleSendMessage}
          disabled={!inputMessage.trim() || isGenerating}
          className="px-6 py-3 bg-blue-600 text-white rounded-lg hover:bg-blue-
700 transition-colors disabled:opacity-50 disabled:cursor-not-allowed flex items-center
justify-center"

          style={{ minHeight: '52px' }}
        >
          {isGenerating ? (
            <Loader2 className="w-5 h-5 animate-spin" />
          ) : (
            <Send className="w-5 h-5" />
          )}
        </button>
      </div>
    </div>
  </div>
) }
</div>
</div>
</div>
);
}

```

## 步骤 5.2: 添加处理进度提示















✅ 修改位置 11: 在 ChatPDF 页面添加进度提示组件

```
//  在渲染部分添加处理中提示
{currentPdf && pdfStatus[currentPdf.id] === 'processing' && (
  <div className="mb-4 p-4 bg-blue-50 border border-blue-200 rounded-lg flex items-center">
    <Loader2 className="w-5 h-5 text-blue-500 mr-3 animate-spin" />
    <div className="flex-1">
      <p className="text-sm text-blue-800 font-medium">正在处理 PDF 文件</p>
      <p className="text-sm text-blue-600 mt-1">
        正在进行文本提取、分块和向量化，预计需要 1-2 分钟...
      </p>
    </div>
  </div>
)}

```

## 阶段 6: 测试与优化

### 步骤 6.1: 测试清单

- # 1. 测试 PDF 上传
  -  上传小文件 (< 1MB)
  -  上传大文件 (10-20MB)
  -  上传非 PDF 文件 (应拒绝)
  -  上传超大文件 (> 20MB, 应拒绝)
- # 2. 测试 RAG 处理
  -  检查数据库中的 document\_chunks 表
  -  验证向量是否正确保存
  -  查询 PDF 状态 API
- # 3. 测试对话功能
  -  提问简单问题
  -  提问复杂问题
  -  提问超出文档范围的问题
  -  检查引用来源是否正确
- # 4. 性能测试
  -  多个 PDF 同时上传
  -  大量对话请求
  -  检查响应时间

### 步骤 6.2: 性能优化建议

```
//  优化位置 1: 添加缓存机制 (可选)
// 在 lib/rag/retrieval.js 中添加
import NodeCache from 'node-cache';
const cache = new NodeCache({ stdTTL: 600 }); // 10 分钟缓存

export async function searchSimilarChunks(query, options = {}) {
  const cacheKey = `search:${query}:${JSON.stringify(options)}`;

```

```

const cached = cache.get(cacheKey);

if (cached) {
  console.log('✅ 使用缓存结果');
  return cached;
}

const results = await performSearch(query, options);
cache.set(cacheKey, results);

return results;
}

// ✅ 优化位置 2: 批量处理优化
// 在 app/api/pdf/upload/route.js 中
// 将向量化批次大小调整为 50（避免超时）
const vectors = await embedBatch(texts, { batchSize: 50 });

// ✅ 优化位置 3: 添加索引
// 在数据库中添加额外索引
CREATE INDEX idx_pdf_status ON pdfs(status);
CREATE INDEX idx_chunk_pdf_id ON document_chunks(pdfId);

```

## 完整文件清单

```

ai-chat-app/
├── prisma/
│   ├── schema.prisma          ✅ 修改
│   └── migrations/
│       └── create_vector_index.sql  ✅ 新增
├── lib/
│   ├── rag/                   ✅ 新增目录
│   │   ├── embeddings.js      ✅ 新增
│   │   ├── chunking.js        ✅ 新增
│   │   └── retrieval.js        ✅ 新增
│   └── prisma.js               (保持不变)
├── app/
│   ├── api/
│   │   ├── pdf/
│   │   │   ├── upload/route.js  ✅ 修改
│   │   │   ├── status/route.js  ✅ 新增
│   │   │   └── chat/route.js      (保持不变)
│   │   └── chat-pdf/route.js     ✅ 修改
│   └── chatpdf/
│       └── page.js               ✅ 修改
├── .env.local                  ✅ 修改
└── package.json                ✅ 修改

```

## 快速启动指南

```
# 1. 安装依赖
npm install langchain @langchain/openai tiktoken

# 2. 配置环境变量 (.env.local)
# 添加 RAG 相关配置 (见步骤 2.2)

# 3. 数据库迁移
npx prisma migrate dev --name add_rag_support
npx prisma generate

# 4. 创建向量索引
psql -U your_user -d ai_chat -f prisma/migrations/create_vector_index.sql

# 5. 启动开发服务器
npm run dev

# 6. 测试上传
# 访问 http://localhost:3000/chatpdf
# 上传一个 PDF 文件
# 等待处理完成 (1-2 分钟)
# 开始对话测试
```

## ? 常见问题

### Q1: 向量索引创建失败?

```
-- 检查 pgvector 是否安装
SELECT * FROM pg_extension WHERE extname = 'vector';

-- 如果未安装, 执行
CREATE EXTENSION vector;
```

### Q2: 向量化速度慢?

```
// 调整批次大小
const vectors = await embedBatch(texts, { batchSize: 50 });
```

### Q3: 检索结果不准确?

```
// 调整相似度阈值
const results = await searchSimilarChunks(query, {
  threshold: 0.5, // 降低阈值
  topK: 10,       // 增加返回数量
});
```

## Q4: 数据库查询超时?

```
-- 确保索引已创建
\d document_chunks

-- 如果没有索引，手动创建
CREATE INDEX document_chunks_embedding_idx
ON document_chunks
USING ivfflat (embedding vector_cosine_ops)
WITH (lists = 100);
```

---

## 总结

本方案



