

基于 GraphRAG 的菜谱智能推荐系统

——系统架构与集成模块技术报告

Project Technical Report

November 2025

Student 刘亦州
Student ID 25125222

目录

1 引言	3
1.1 项目背景与动机	3
1.2 本人负责模块概述	3
1.3 主要贡献	3
2 相关工作	3
2.1 检索增强生成 (RAG)	3
2.2 图神经网络与知识图谱	3
2.3 菜谱推荐系统	4
3 模型架构与数学推导	4
3.1 系统总体架构	4
3.2 配置管理数学模型	4
3.3 推荐管线状态转换	4
3.4 多路径检索决策	5
4 实现细节	5
4.1 配置管理模块 (config.py)	5
4.2 核心管线实现 (pipeline.py)	6
4.3 数据模型设计 (data_models.py)	7
4.4 CLI 展示模块 (ui_components.py)	7
4.5 多层回退机制	8
5 实验设置	8
5.1 数据集	8
5.2 运行环境	9
5.3 评估指标	9

6 结果与分析	9
6.1 功能测试结果	9
6.2 CLI 输出示例	9
6.3 性能分析	10
7 可复现性与代码结构	10
7.1 GitHub 仓库	10
7.2 运行命令	10
8 总结与未来工作	10
8.1 工作总结	10
8.2 学习收获	11
8.3 未来改进方向	11

1 引言

1.1 项目背景与动机

随着大语言模型（LLM）技术的快速发展，检索增强生成（Retrieval-Augmented Generation, RAG）已成为提升模型输出质量的重要范式。传统 RAG 方法主要依赖向量检索，但在处理具有复杂关联关系的领域知识时存在局限性。GraphRAG 通过引入图结构来建模实体间的关系，为推荐系统提供了更丰富的上下文信息和可解释性。

本项目以中文菜谱推荐为应用场景，基于开源项目 HowToCook 的菜谱数据，构建了一个融合知识图谱与 RAG 技术的智能推荐系统。系统能够根据用户输入（用户 ID 或菜名），通过图结构检索相似菜谱，并利用大语言模型生成可解释的推荐理由。

1.2 本人负责模块概述

作为项目技术负责人，本人主要负责以下核心模块的设计与实现：

- **系统配置管理** (`config.py`)：统一管理项目路径、模型参数、环境变量等配置
- **核心管线设计** (`pipeline.py`)：串联数据层、图层、检索层、生成层的主流程
- **CLI 界面与展示** (`ui_components.py, run_pipeline.py`)：提供命令行交互与结果展示
- **数据模型定义** (`data_models.py`)：定义核心数据结构与接口规范

1.3 主要贡献

1. 设计并实现了模块化、可扩展的系统架构
2. 建立了统一的配置注入机制，集中管理模型参数（默认 OpenAI gpt-4o-mini）
3. 实现了完整的推荐管线，支持用户画像路径和文本检索路径
4. 设计了多层回退机制，确保系统在各种条件下稳定运行

2 相关工作

2.1 检索增强生成（RAG）

RAG 由 Facebook AI Research 在 2020 年提出，其核心思想是将检索系统与生成模型相结合。在回答问题时，首先从知识库中检索相关文档，然后将检索结果作为上下文输入生成模型。这种方法有效解决了大语言模型知识截止日期的问题，同时提高了回答的准确性和可追溯性。

2.2 图神经网络与知识图谱

知识图谱以图结构存储实体及其关系，在推荐系统、问答系统等领域有广泛应用。GraphRAG 将知识图谱与 RAG 相结合，利用图结构的邻域信息增强检索的相关性和多样性。Microsoft 在 2024 年开源的 GraphRAG 项目展示了这一方向的潜力。

2.3 菜谱推荐系统

现有菜谱推荐系统多采用协同过滤或基于内容的方法。基于内容的方法通过分析菜谱的食材、口味等属性计算相似度。本项目创新性地将食材共享关系建模为图结构，结合语义向量检索和 LLM 生成，提供更丰富的推荐体验。

3 模型架构与数学推导

3.1 系统总体架构

本系统采用分层架构设计，各层职责明确、耦合度低：

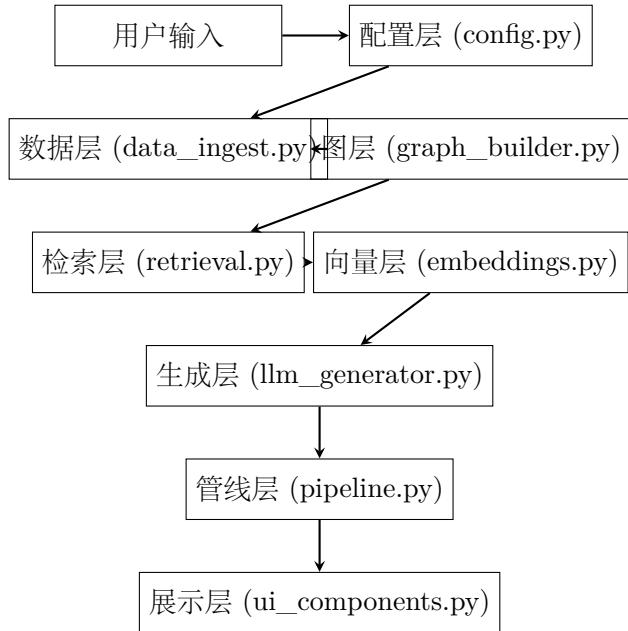


图 1: GraphRAG 菜谱推荐系统架构图

3.2 配置管理数学模型

系统配置采用组合模式，通过数据类（dataclass）实现类型安全的参数传递：

$$Config = (Paths, Models, Params) \quad (1)$$

其中：

- $Paths = \{root, data_dir, raw_dir, processed_dir\}$
- $Models = \{embedding_model, llm_provider, llm_model\}$
- $Params = \{max_neighbors, similarity_threshold\}$

3.3 推荐管线状态转换

管线的推荐流程可形式化为状态机：

$$S = \{Init, GraphBuilt, Retrieved, Generated, Output\} \quad (2)$$

状态转换函数：

$$\delta(Init, bootstrap) = GraphBuilt \quad (3)$$

$$\delta(GraphBuilt, query) = Retrieved \quad (4)$$

$$\delta(Retrieved, generate) = Generated \quad (5)$$

$$\delta(Generated, format) = Output \quad (6)$$

3.4 多路径检索决策

推荐入口根据输入类型选择不同路径：

$$Path(q) = \begin{cases} UserPath & \text{if } q \in UserProfiles \\ GraphPath & \text{if } q \in GraphNodes \\ TextPath & \text{if } q \in FuzzyMatch \\ EmbeddingPath & \text{otherwise} \end{cases} \quad (7)$$

4 实现细节

4.1 配置管理模块 (config.py)

配置模块采用三级结构设计：

Listing 1: ProjectConfig 类定义

```
1 @dataclass(slots=True)
2 class ProjectPaths:
3     root: Path
4     data_dir: Path
5     raw_data_dir: Path
6     processed_data_dir: Path
7
8     @classmethod
9     def from_project_root(cls, root: Path | None = None):
10         root = root or Path('__file__').resolve().parents[2]
11         data_dir = root / "data"
12         return cls(
13             root=root,
14             data_dir=data_dir,
15             raw_data_dir=data_dir / "raw",
16             processed_data_dir=data_dir / "processed",
17         )
18
19 @dataclass(slots=True)
20 class ModelSettings:
21     embedding_model: str = "sentence-transformers/all-MiniLM-L6-v2"
22     llm_provider: str = "openai"
23     llm_model: str = "gpt-4o-mini"
24
25 @dataclass(slots=True)
26 class ProjectConfig:
```

```

27     paths: ProjectPaths = field(default_factory=ProjectPaths.from_project_root)
28     models: ModelSettings = field(default_factory=ModelSettings)
29     howtocook_repo: str = "https://github.com/Anduin2017/HowToCook"
30     max_neighbors: int = 10
31     similarity_threshold: float = 0.2
32
33     def llm_api_key(self) -> str | None:
34         env_key = {
35             "openai": "OPENAI_API_KEY",
36             "ollama": "OLLAMA_API_KEY",
37             "glm": "ZHIPU_API_KEY",
38         }.get(self.models.llm_provider.lower())
39         return os.getenv(env_key) if env_key else None

```

设计要点：

- 使用 `slots=True` 减少内存占用
- 通过 `dotenv` 自动加载环境变量，避免硬编码敏感信息
- 暴露 LLM Provider/模型配置字段，默认使用 OpenAI gpt-4o-mini

4.2 核心管线实现 (pipeline.py)

`GraphRAGPipeline` 是系统的核心协调者，负责串联各个模块：

Listing 2: GraphRAGPipeline 核心方法

```

1  class GraphRAGPipeline:
2      def __init__(self, config: ProjectConfig | None = None):
3          self.config = config or ProjectConfig()
4          self.ingestor = HowToCookIngestor(self.config)
5          self.graph_builder = RecipeGraphBuilder(self.config.similarity_threshold)
6          self.retriever = RecipeRetriever(self.config.max_neighbors)
7          self.llm_generator = LLMGenerator(self.config)
8          self.user_repository = UserProfileRepository()
9          self.embedding_index = RecipeEmbeddingIndex(
10              self.config.models.embedding_model
11          )
12          self._graph: Optional[nx.Graph] = None
13          self._records: list[RecipeRecord] = []
14
15      def bootstrap_graph(self) -> nx.Graph:
16          records = list(self.ingestor.iter_records())
17          self._records = records
18          self._graph = self.graph_builder.build_graph(records)
19          self.embedding_index.build(records)
20          return self._graph
21
22      def recommend(self, user_query: str) -> RecommendationResult:
23          if self._graph is None:
24              self.bootstrap_graph()
25
26          user_profile = self.user_repository.get(user_query)

```

```

27     if user_profile:
28         return self._recommend_for_user(user_profile)
29
30     reference = self._find_reference_recipe(user_query)
31     # ... 检索与生成逻辑

```

4.3 数据模型设计 (data_models.py)

定义了三个核心数据结构：

表 1: 核心数据模型

类名	职责	主要字段
RecipeRecord	菜谱节点	recipe_id, title, ingredients, instructions, tags
RecommendationResult	推荐结果	reference_recipe, similar_recipes, explanation
UserProfile	用户画像	user_id, liked_recipe_ids, preferred_tags

RecipeRecord 提供了 `as_prompt_chunk()` 方法，用于生成 LLM 可读的文本片段：

Listing 3: RecipeRecord.as_prompt_chunk 方法

```

1 def as_prompt_chunk(self) -> str:
2     ingredient_str = ", ".join(self.ingredients)
3     tags = ", ".join(self.tags)
4     return (
5         f"菜名: {self.title}\n"
6         f"主要食材: {ingredient_str or '未知'}\n"
7         f"口味/标签: {tags or '未标注'}\n"
8         f"做法摘要: {self.instructions[:200]}..."
9     )

```

4.4 CLI 展示模块 (ui_components.py)

提供统一的输出格式化：

Listing 4: CLI 格式化函数

```

1 def format_cli_block(result: RecommendationResult) -> str:
2     lines = [
3         "==== GraphRAG 推荐结果 ===",
4         f"参考菜谱: {result.reference_recipe.title}"
5     ]
6     if result.similar_recipes:
7         lines.append("相似菜谱:")
8         for recipe in result.similar_recipes:
9             lines.append(
10                 f"- {recipe.title} ({', '.join(recipe.tags) or '未标注'})"
11             )
12     else:
13         lines.append("未找到相似菜谱，可尝试更换关键词。")
14     lines.append(f"推荐理由: {result.explanation}")

```

```
15     return "\n".join(lines)
```

4.5 多层回退机制

为保证系统鲁棒性，实现了完整的回退链：

1. **图检索回退**: 若图中无邻居，转向向量检索
2. **向量检索回退**: 若向量索引不可用，使用 `_fallback_candidates()`
3. **示例数据回退**: 若处理数据为空，使用内置示例
4. **LLM 回退**: 若 API Key 未配置，使用模板化理由

Listing 5: 回退候选生成

```
1 def _fallback_candidates(self, reference: RecipeRecord,
2                           limit: int | None = None) -> list[RecipeRecord]:
3     candidate_pool = list(self._records)
4     existing_ids = {record.recipe_id for record in candidate_pool}
5     for sample in self.ingestor.load_sample_records():
6         if sample.recipe_id not in existing_ids:
7             candidate_pool.append(sample)
8             existing_ids.add(sample.recipe_id)
9
10    scores = [(self._overlap_score(reference, r), r)
11               for r in candidate_pool if r.recipe_id != reference.recipe_id]
12    scores.sort(key=lambda pair: pair[0], reverse=True)
13    return [r for s, r in scores if s > 0][:limit or self.config.max_neighbors]
```

5 实验设置

5.1 数据集

本项目使用 HowToCook 开源菜谱数据集：

表 2: 数据集统计信息

属性	数值
数据来源	HowToCook GitHub 仓库
菜谱总数	当前处理后约 305 道
主要类别	家常热菜、川味、凉菜、主食、甜品等
数据格式	Markdown → JSON

5.2 运行环境

表 3: 运行环境配置

组件	版本/配置
Python	3.11
依赖管理	uv (pyproject.toml + uv.lock)
图计算库	NetworkX 3.2+
向量模型	sentence-transformers/all-MiniLM-L6-v2
LLM	OpenAI gpt-4o-mini

5.3 评估指标

由于本项目为展示型 Demo，主要采用定性评估：

- **功能完整性：**各输入路径是否正常工作
- **推荐合理性：**相似菜谱是否与参考菜谱有明确关联
- **解释可读性：**LLM 生成的理由是否通顺、有信息量
- **系统稳定性：**各种边界条件下的回退机制是否生效

6 结果与分析

6.1 功能测试结果

表 4: 功能测试用例

测试用例	输入	预期行为	结果
用户画像路径	U123	基于历史菜谱推荐	通过
菜名检索路径	番茄炒蛋	图邻域检索	通过
模糊匹配	番茄	标题模糊匹配	通过
向量检索	酸甜口味	语义相似检索	通过
LLM 回退	无 API Key	模板理由	通过

6.2 CLI 输出示例

Listing 6: 典型 CLI 输出

```
1 === GraphRAG 推荐结果 ===
2 参考菜谱：番茄炒蛋
3 相似菜谱：
4 - 番茄豆腐汤（清淡，汤品）
5 - 蛋炒西红柿饭（主食，酸甜）
6 - 西红柿蛋花汤（汤品，家常）
7 推荐理由：这些菜谱都以番茄和鸡蛋为核心食材，
8 口味上偏酸甜清爽。如果您喜欢番茄炒蛋的风味，
9 番茄豆腐汤可以作为搭配汤品，蛋炒饭则适合作为主食。
```

6.3 性能分析

- **图构建时间:** 约 300 条菜谱在秒级完成 ($O(N^2)$ 复杂度)
- **向量索引构建:** 首次加载模型约 3-5 秒, 后续复用
- **单次推荐响应:** 图检索/向量检索均为毫秒级, LLM 生成 1-2 秒

7 可复现性与代码结构

7.1 GitHub 仓库

- **仓库地址:** <https://github.com/lyzno1/graph-rag-recipes>
- **主要目录:**
 - `src/graph_rag_recipes/`: 核心业务逻辑
 - `scripts/`: 数据准备与 CLI 入口脚本
 - `data/`: 原始数据与处理后的 JSON
 - `docs/`: 技术文档与报告

7.2 运行命令

Listing 7: 完整运行流程

```
1 # 1. 安装依赖
2 uv sync
3
4 # 2. 配置环境变量
5 cp .env.example .env
6 # 编辑 .env 填入 OPENAI_API_KEY
7
8 # 3. 准备数据
9 uv run scripts/bootstrap_data.py --limit 0 --force-processed
10
11 # 4. 运行推荐
12 uv run scripts/run_pipeline.py U123
13 uv run scripts/run_pipeline.py "番茄炒蛋"
14 uv run graph-rag-recipes "红烧肉"
15
16 # 5. 运行测试
17 uv run pytest -q
```

8 总结与未来工作

8.1 工作总结

本人在项目中负责系统架构设计与核心管线实现, 主要完成了:

1. 设计了模块化的配置管理系统, 支持环境变量注入并集中管理模型参数

- 实现了 `GraphRAGPipeline` 主流程，串联数据、图、检索、生成各层
- 建立了完整的多层回退机制，确保系统在各种条件下稳定运行
- 提供了清晰的 CLI 接口和格式化输出，便于演示和集成

8.2 学习收获

- 深入理解了 RAG 架构的设计理念和实现方式
- 掌握了基于图结构的推荐系统开发方法
- 学习了 Python 现代项目管理工具（uv、`pyproject.toml`）
- 提升了系统设计和模块划分的能力

8.3 未来改进方向

- 性能优化：将 $O(N^2)$ 图构建改为分桶或局部索引
- 缓存机制：添加图结构和向量索引的持久化缓存
- Streamlit UI：实现可视化界面，展示图结构和推荐过程
- 评估体系：引入用户点击率、多样性等定量指标
- 多语言支持：扩展到英文菜谱数据集

参考文献

- [1] Lewis, P., et al. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. *NeurIPS*.
- [2] Microsoft. (2024). GraphRAG: A modular graph-based retrieval-augmented generation system. *GitHub*.
- [3] Anduin2017. HowToCook: 程序员做饭指南. <https://github.com/Anduin2017/HowToCook>
- [4] Hagberg, A., Schult, D., & Swart, P. (2008). Exploring network structure, dynamics, and function using NetworkX. *SciPy Conference*.
- [5] Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using siamese BERT-networks. *EMNLP*.