简历写法

北京居之道科技有限公司　　大模型算法实习生　　2024.12-2025.05

智能客服Agent系统

项目背景：电商客服日均5k–10k咨询，高度依赖FAQ，命中率低、人工介入多；接入说明书/售后政策等知识库后仍存在幻觉严重、多跳推理差、结构化与非结构化数据割裂等问题。

解决方案：

- **意图识别**：Prompt-template + DeepSeek v3，硬编码5类标签(JSON 输出)；通过上线日志做**难负样本挖掘**，few-shot 三轮迭代将意图识别**F1由0.82提升至0.93。**
- **多Agent架构**：基于**LangGraph**设计Multi-Agent流程（意图识别 → 安全护栏 → Planner子任务分解 → 并行tools → 幻觉拦截），复用Map-Reduce机制并行调用图数据库与文本检索。
- **并行tools**：结构化数据走Neo4j Text2Cypher，非结构化数据走GraphRAG，并且预定义了一些 Cypher 查询语句，用于处理高频率的图数据库查询需求。并行执行后**单条复杂询问平均延迟由4.8s降至2.1s。**

项目成果：意图识别F1**0.93**，Cypher查询逻辑正确率**88%**，系统幻觉率降低至**0.8%。**沉淀100+ Cypher模板与完整防护链，为公司后续图数据库项目提供标准范式。

（一定要有数值化的评价指标）

项目描述

先介绍项目背景，再说一开始是怎么做的，但是遇到了什么困难、挑战，尝试了什么新的方法，结论是什么（数值型评估指标），一定要突出项目“迭代”的过程，实习/校招项目至少迭代一次，社招项目至少迭代两次。

这个项目的背景是公司（乙方）接到业务方（甲方）的需求，希望他们的电商客服机器人能更加智能，每天的咨询量大概在五千到一万条，此前的客服机器人主要依赖FAQ，命中率低，需要大量的人工介入，后来接入了产品说明书、售后政策、产品数据库等作为RAG的知识库，但依然存在幻觉严重、处理多跳问题效果不好、结构化数据与非结构化数据割裂等问题，因此我们用LangGraph 构建了 Multi-Agent 体系，让大模型像资深的客服人员一样，自动识别用户意图、调用多种工具、返回正确的答案，并且把幻觉率控制到业务可接受范围内。

意图识别用的是 Prompt-template + DeepSeek v3。模板里硬编码了五类标签——general、additional、graphrag、image、file——并加了结构化输出声明，让模型只返回 JSON。上线后我又**用上线日志做难负样本挖掘**，把误分样本反补进 few-shot 列表里，迭代了三轮后，Intent F1 从 0.82 提升到 0.93。

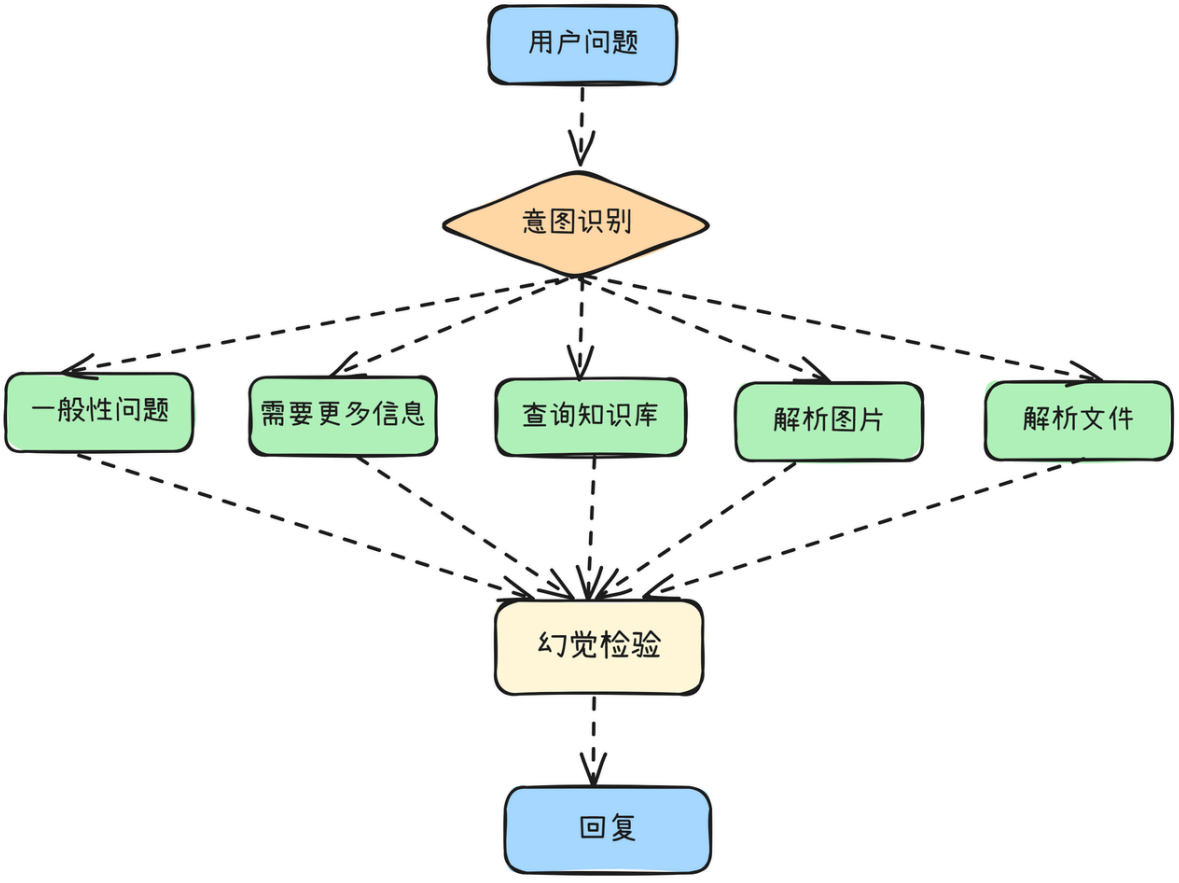
当问题落到 graphrag-query 时，先用 Planner 把复合询问拆成子任务——比如“查库存”与“查价格”各成一路。之后我借助 LangGraph 的 Map-Reduce 模式并行跑工具：结构化数据走 Neo4j Text2Cypher，非结构化数据走 Microsoft GraphRAG，并且预定义了一些 Cypher 查询语句，用于处理高频率的图数据库查询需求。这样单条复杂询问的平均延迟从串行的 4.8 s 降到 2.1 s。

Text2Cypher部分做了三层防护：

1. 人工梳理了80 条的高频查询 + 20 余条 LLM 自动补充，覆盖库存、价格、规格等核心场景；
2. 调用前实时抓取 Neo4j 元数据，自动填充占位符并规避字段/类型冲突；
3. 自动校验-纠错：先用 EXPLAIN 语法预检，若报错则由 DeepSeek 结合最新 Schema 生成修正版并自动重试。正式环境里Cypher 生成成功率 **94 %**，逻辑正确率 **88 %**。

在 additional-query 和 graphrag-query 开头都放了**安全护栏**节点。它先判断提问是否落在经营范围，再决定继续还是直接礼貌拒答；通过把业务品类写进 prompt 并动态拼 Neo4j Schema，我们把误放比例压到 4% 以内。最后输出前还走一个 Hallucination Detector，平均每 1000 句可拦截 7-8 条潜在幻觉回复。

以上是这个项目的详细介绍，看您这边有什么想问的吗？



项目相关的问题

1、意图识别这块是怎么优化的？（原代码未作优化）

langgraph中的Router 节点将用户问题分类的五个意图分别为：general-query（一般性问题）、additional-query（需要更多信息才能回答的问题）、graphrag-query（需要查询知识库的问题）、image-query（需要解析用户上传图片的问题）、file-query（需要解析用户上传文件的问题）。

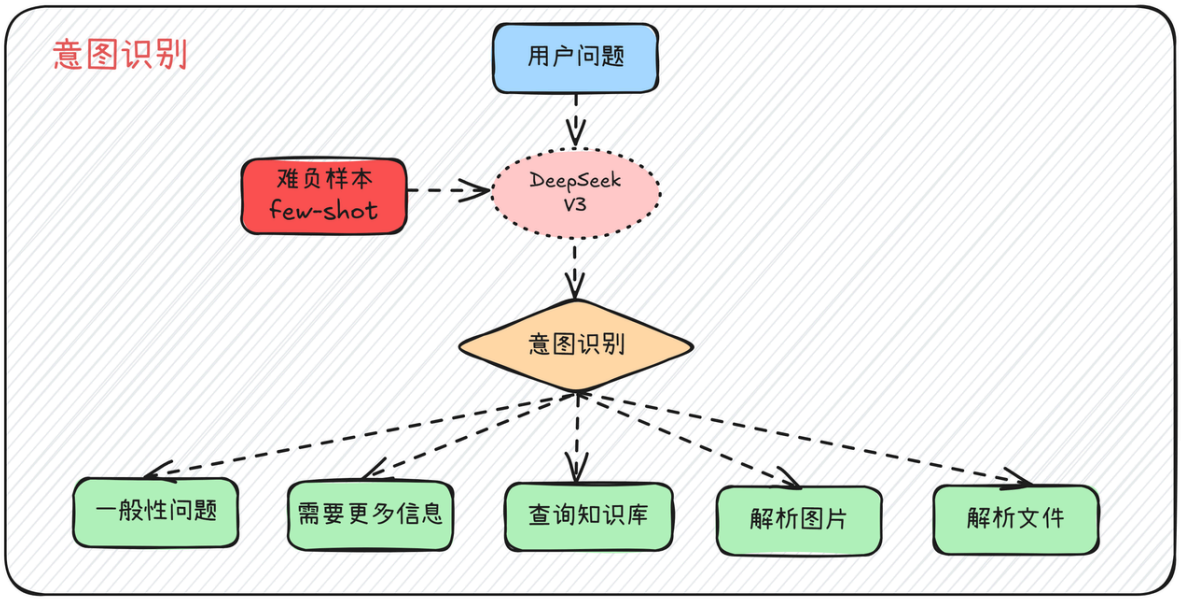
其中后面两个类别是用规则来识别的。

针对前三个类别，设计特定的提示词，明确不同类型问题的分类标准，对每一个子路由能处理的问题进行详细描述，并给出一些few shot引导大模型给出正确分类。

上线以后，分析bad case发现误分主要集中在长句和口语化表达上，于是我把线上误分做 难负样本抽样，定期把最具代表性的错例塞回 few-shot，保持提示长度不变；三轮下来，F1 从 0.82 → 0.93，延迟基本不动。

具体来说，我先把三天的线上日志跑离线评估，统计所有被模型**高置信度**判错的样本；

再用 bge-m3转向量，再用k-means做聚类，保证每种典型错法至少有 3-5 条，总共挑出大概 40 来条。把这些"模型不擅长、但业务高频"的句子反补进 prompt 的 few-shot 列表，同时淘汰掉老的冗余示例，保持示例总量不变。这样做了三轮迭代，意图识别的 F1 就从 0.82 拉到了 0.93，而且几乎不增加推理时延迟。



1.1、难负样本是怎么得出的?

把deepseek v3 输出的 **label token 的 logits** 计算 概率，得到 [0-1] 分数。从分数大于0.8的数据中人工挑选出分类错误的用户query作为难负样本，挖掘出的难负样本数量大约为总数的3%。

1.2、概率是怎么计算的?

DeepSeek v3 在推理时会给每个可输出 token 一个logprobs。把 **"最高分标签 token 的 logprobs 与第二高分的差值"** 取出来，再做一次 **sigmoid 归一化**，就能得到 0–1 之间的置信度 —— 差值越大，模型越肯定自己的选择。

> 通过 prompt 让模型**只能**输出 5 个意图里的某一个词，在调用 DeepSeek v3 API 时加 `logprobs=True`，它会把每个候选 token 的 logprobs 都返回。设 `l1 = 最高 logprobs`，`l2 = 第二高 logprobs`，令 `margin = l1 − l2`， **Sigmoid 差值**：`P = 1 / (1 + e^{−margin})`。

```
import numpy as np
```

```
# 假设从模型中得到的logits
logits = np.array([2.0, 1.0, 0.1])

# 计算Softmax概率
exp_logits = np.exp(logits)
probabilities = exp_logits / np.sum(exp_logits)

# 计算logprobs
logprobs = np.log(probabilities)

# 输出结果
print("Logits: ", logits)
print("Probabilities: ", probabilities)
print("Log-probabilities: ", logprobs)
```

1.3、意图识别的测试数据集是怎么来的?

主要来自客服系统的真实对话记录，共2000条样本，并做了人工标注。数据分布上，高频意图如订单处理（35%）和物流查询（30%）占比较大，低频如技术支持仅占5%。

1.4、f1的值是如何得到的?

这个F1指标是我们基于人工标注的离线评估集算出来的。具体做法是：

1. 从线上收集三天的用户问题及系统意图识别结果，再请业务人员或算法同学人工标注每条问题的真实意图标签。
2. 对于每一类意图（如general-query），把真实标签为该类别的样本视为正例，其他类别视为负例。
3. 用模型的预测结果和人工标签做对比，统计每类的TP/FP/FN，计算precision和recall，最后算F1。这里是多类别，使用micro-F1。
4. 这个评估过程和上线前/后的prompt优化迭代过程同步，保证评估数据和业务场景一致。我们最后公布的F1（0.93）是所有类别的加权平均结果。

正例是指"某问题真实标签和模型预测标签一致"，负例是指"模型预测错了意图类别"。尤其对于我们主要关注的graphrag-query、general-query和additional-query，都会分别统计对应的TP、FP、FN，最后整体加权平均。

1.5、为什么意图识别模块选择生成模型而非传统的判别模型（如BERT）？并且为什么选择deepseek这样参数量较大的模型而不是用参数量较小的模型，时延可以接受吗?

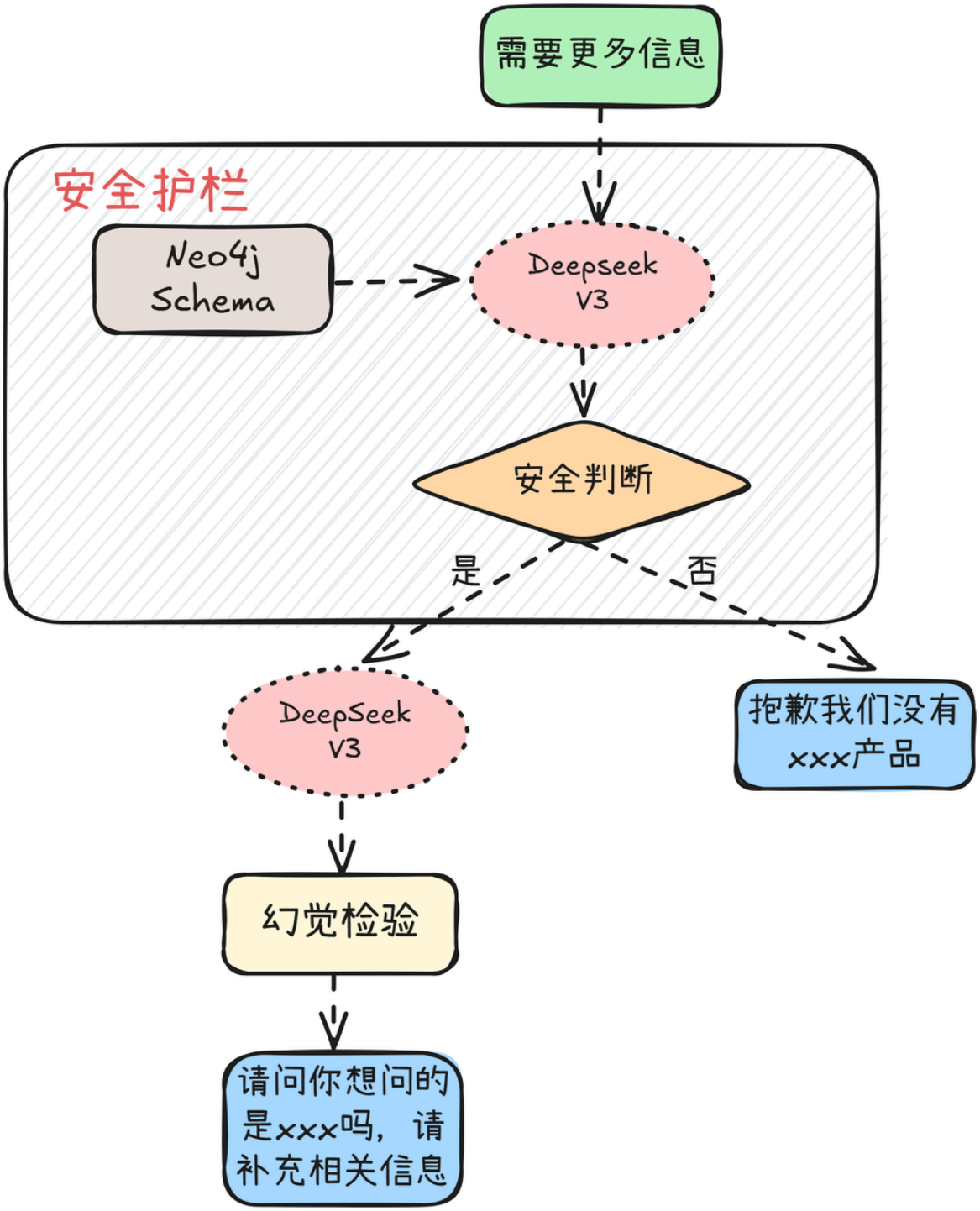
这里是通过few-shot+提示词的方式调用的deepseek，优点是在项目上线初期比较灵活，可以灵活调整意图种类、每种意图的示例等，而用bert/0.6B小模型每次增加新数据都要重新训练，并且大模型面对复杂的用户查询性能会更好。

时延方面肯定是bert更优，但考虑到智能客服允许一定的思考时间，时延是可以接受的。这种仅使用Prompt来做意图识别的方式也容易导致case by case的问题，后期收集到足够多的样本，可以训练bert/0.6B小模型替代deepseek。

2、“需要更多信息才能回答的问题” 这一类问题是怎么让用户补充信息的?

这一类的问题会先过一个安全护栏，这个护栏会把 **实时取回的一份 Neo4j Schema**（节点、属性、关系）塞进提示词里，让大模型判断“问题有没有落在我们的经营范围”。

经过安全护栏之后，additional-query 只是检查“缺了哪几个关键槽位（商品型号、订单号…）”，用规则库加 LLM 生成一句引导语请用户补充必要的信息。



2.1、每次都要实时获取的话，neo4j里的数据表很多怎么办？有做优化措施吗？

我们没有每次都全量实时读取Neo4j Schema，而是采用了多层优化措施。就是**缓存Schema并定期更新**，大幅减少了数据量。

后续优化可以考虑：如果用户问题比较模糊，还可以结合**问题关键字只加载相关局部Schema**，比如通过关键字匹配或向量召回，只返回与“用户”、“积分”等相关的节点和关系。对于极端复杂的Schema，可以采用**结构摘要**和多轮细化策略，或者Schema向量化做粗筛。

3、怎么将结构化的csv文件导入Neo4j?

第一步是准备数据，定义节点和关系的类型，根据现有的csv文件，我们需要确保CSV文件的格式符合Neo4j的要求，直接定义明确的规范构建知识图谱，导入到如 Neo4j 中进行统一管理。例如，对于产品节点，CSV文件中应该有产品ID、产品名称、单价等字段。对于关系，比如产品属于某个类别，CSV文件中则需要包含产品ID和类别ID等能够建立关联的字段。

第二步是利用Neo4j Admin工具来进行导入，Neo4j Admin工具是一个命令行工具，专门用于管理和维护Neo4j数据库实例，并行批量导入数据到Neo4j。

4、怎么将非结构化的PDF文件导入Neo4j?

首先，PDF文件需要被解析为结构化数据。我使用了MinerU，将PDF转换为Markdown格式的文本，保留文档的布局、元素位置、表格结构、图片路径等元数据。这一步的核心是确保文本、表格、图片等元素被准确分离并关联上下文信息。

第二步：语义增强与元数据整合，将解析后的Markdown数据进一步优化。例如：

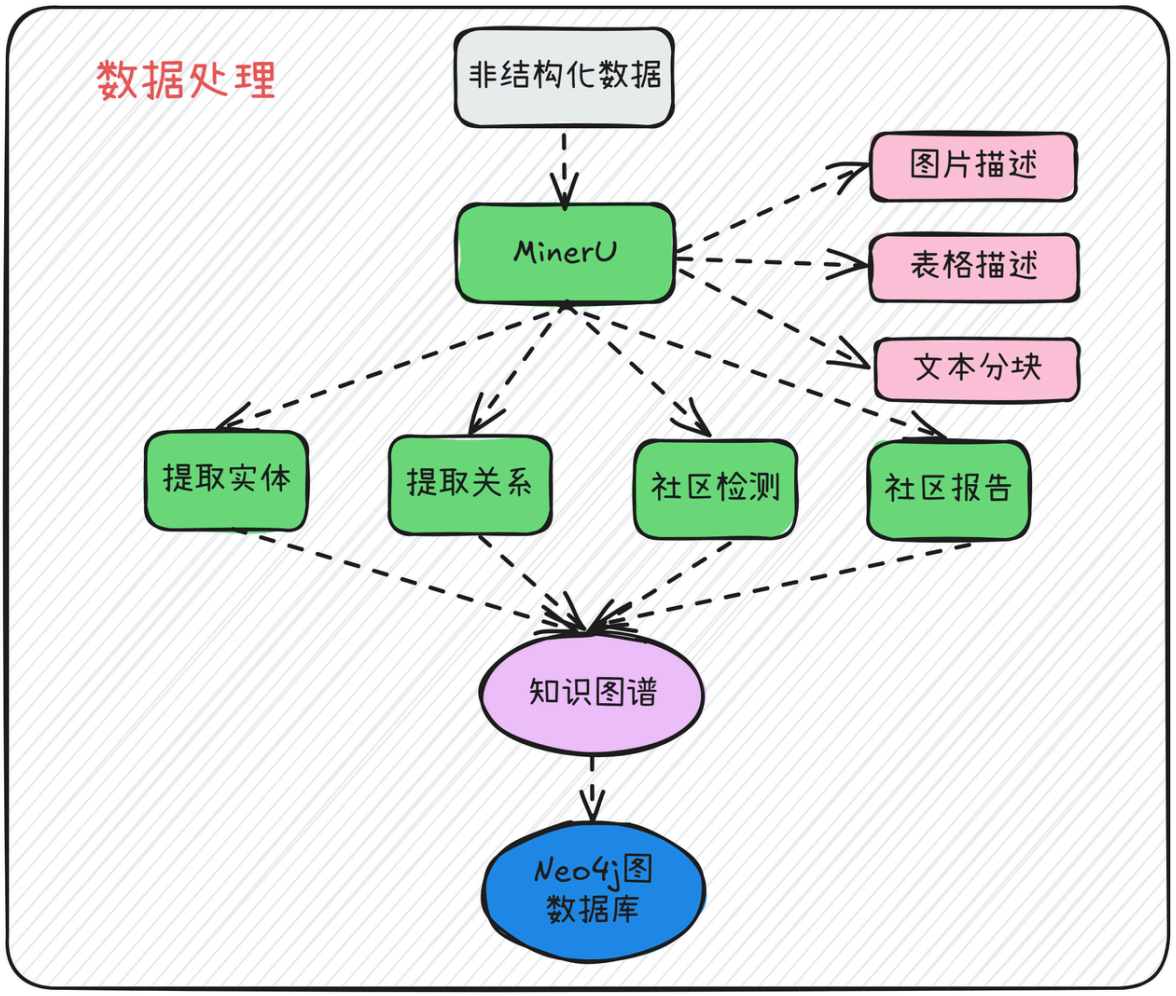
- **表格**：从Markdown中提取结构化表格数据，并通过大模型生成摘要描述，作为表格的补充信息。
- **图片**：提取图片的存储路径，并通过视觉模型生成内容描述，同时关联其前后文的文本说明。
- **文本分块**：采用“结构感知分块”策略，比如按标题层级（如一级标题、二级标题）划分主块，确保同一标题下的文本、表格、图片属于同一上下文单元。

这一步的目标是为每个内容块（无论是文本、表格还是图片）附加丰富的元数据，便于后续构建知识图谱中的节点属性。

第三步：实体与关系提取

- **实体**：每个文本块、表格、图片都可以视为一个节点。例如，文本块可能包含“产品功能介绍”，表格可能代表“销售额数据”，图片可能描述“系统架构图”。
- **关系**：基于文档结构建立关联。例如，一级标题节点下关联多个子标题节点，子标题节点进一步关联具体的文本块或表格。表格和图片也会通过上下文关系（如“隶属于”“引用自”）链接到对应的文本节点
- **社区**：通过**Leiden算法**构建社区

最后将实体、关系、社区、社区报告等并行批量导入Neo4j数据库。



4.1、上面提到的两个Neo4j是一个图数据库吗?

不是，是两个。

4.2、怎么进行知识图谱的新增与维护?

采用增量更新的方式，避免每次都要重新构建知识图谱。

结构化数据（CSV）：业务侧不断产出新的 CSV 文件（如新增商品、订单等），我们会先在本地进行格式校验，确保包含必要字段（比如商品必须包含ID、名称、存活量、价格等），然后借助 `neo4j-admin` 工具将新文件以增量模式导入到 Neo4j 中，提高效率。

非结构化文档（PDF/Markdown）：新增的技术文档、说明书或政策文件，会触发一条自动化流水线：

1. 使用 **MinerU** 将 PDF 转换为 Markdown，并保留布局、表格、图片等元数据；
2. 运行结构感知分块脚本，将文档切分为若干“内容块”（chunk），并附加页码、标题层级、包含“表格/图片”标记等元信息；
3. 对每个内容块做语义增强：表格提取并摘要、图片路径和文字描述抽取；
4. 基于之前介绍的实体与关系抽取规则+Leiden 算法，离线抽取出“实体节点”、“关系边”以及“社区子图”；
5. 最后将新生成的节点、边、社区报告等增量写入 Neo4j。

Schema同步：在缓存中存储 `get_schema()` 获取Neo4j的 数据库结构，定期（每分钟）进行增量同步，以捕获新增节点/属性或删除变更。

4.3、图数据库里面有多少条数据，有多少张表，特征字段有哪些?

（回答主要展示专业思路+细节覆盖面，他也不指望你报出精准数字）

我们项目实际用了**两个Neo4j图数据库**，一个用于存放**结构化数据**，比如产品、订单、用户、FAQ等业务核心数据；另一个Neo4j实例专门存放**非结构化数据**，比如解析自产品说明书、售后政策PDF、图片、表格等实体及其关系。这样设计有利于数据隔离和检索优化，后续如果业务量扩大还可以独立扩展。

- **结构化数据库**目前有**30~50万条节点**，**60~100万条关系**，主要是产品、订单、用户等信息和他们之间的业务关系。
- **非结构化数据库**主要是从产品说明书、售后政策、PDF等文件解析出来的内容块、表格、图片等节点和它们的上下文/引用关系，总体大概**20~30万条节点**，**50万条关系**左右。

这两个库的数据量会根据实际业务增量自动同步，每天都在增加。

结构化数据库（举例）：

- **Product**（产品）：ProductId, ProductNname, SupplierID, CategoryID, Unitprice, UnitInStock
- **Order**（订单）：OrderId, CustomerId，EmployeeID，OrderDate，ShippedDate，ShippedName，ShippedAddress，
- **Customers**（用户）：CustomerId, ContactName, Address，Phone

非结构化数据库（举例）：

- **DocumentBlock**（文本块）：block_id, content, doc_type, page_no, section
- **Table**（表格）：table_id, description, row_count, col_count
- **Image**（图片）：image_id, img_path, caption, doc_ref

关系类型（部分）：

- **BELONGS_TO**（归属某品类/章节）
- **CONTAINS**（章节包含内容块/图片/表格）
- **SHIPPED_VIA**（通过xx物流）
- **REFERS_TO**（引用了某实体/内容）

5、PDF是如何分块的?

PDF分块用的是结构感知分块，主要基于文档解析出来的标题层级来切分chunk。例如，每个一级标题作为一个主块，块内会聚合该章节下的文本、表格和图片。这样既保留了文档原有的逻辑结构，也方便后续Neo4j建模。如果遇到某块内容过长，还会在块内再按段落或语义边界进一步细分，保证chunk大小适中，提升检索和向量化效果。分块时还会附带元数据，包括所属标题、页码、是否含表格/图片等。整体流程是先用MinerU结构化解析，再用Python脚本自动化分块和元数据整合。

6、安全护栏是怎么做的?

我们系统在安全护栏的设计上采用了提示词工程加图数据库Schema辅助的方式。首先通过精心设计的提示词告诉大模型哪些问题是超出经营范围的，同时动态传入Neo4j数据库结构信息，帮助模型更准确地理解系统支持的数据类型。最终输出结构化的决策结果continue或end，用来控制对话是否继续。

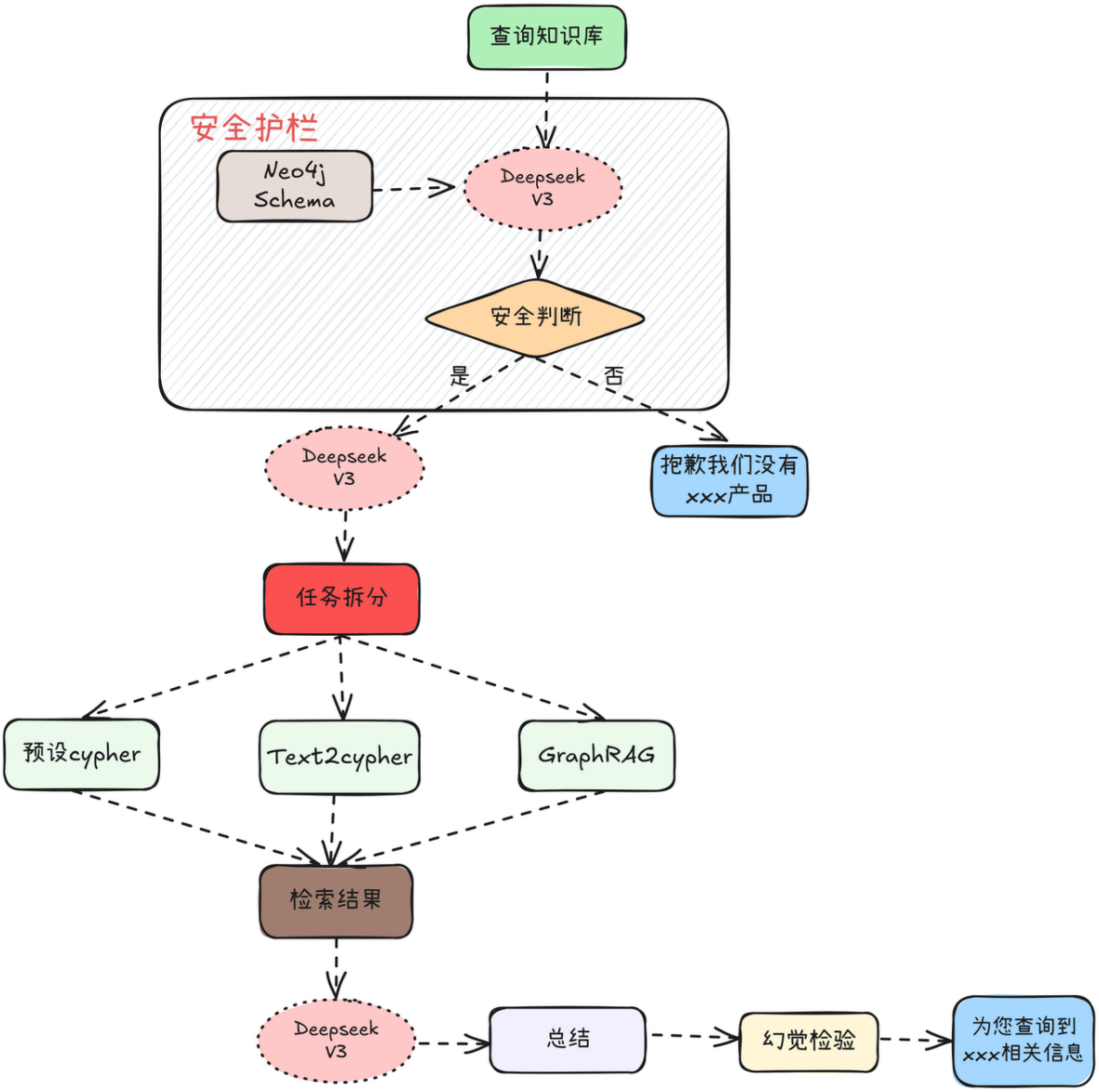
6.1、怎么将图数据库的 Schema 信息动态注入提示词中?

我们将 Neo4j 的 Schema 信息动态注入提示词中，主要是为了辅助大模型更准确地判断用户问题是否属于业务范围。具体做法是：首先通过 `get_schema()` 方法从 Neo4j 中获取数据库结构，然后对 Schema 进行清理，比如去除内部节点信息、避免与模板变量冲突的符号替换。接着，将整理后的 Schema 放入提示词模板中的 `{neo4j_schema}` 占位符里，最终作为 System Prompt 提供给大模型使用。这种方式可以让大模型实时了解系统支持的实体和关系，提高安全护栏判断的准确性。

例如，如果用户问'小米音箱价格'，我们需要知道数据库中是否有 Product 节点和 UnitPrice 属性。

7、知识库查询怎么做

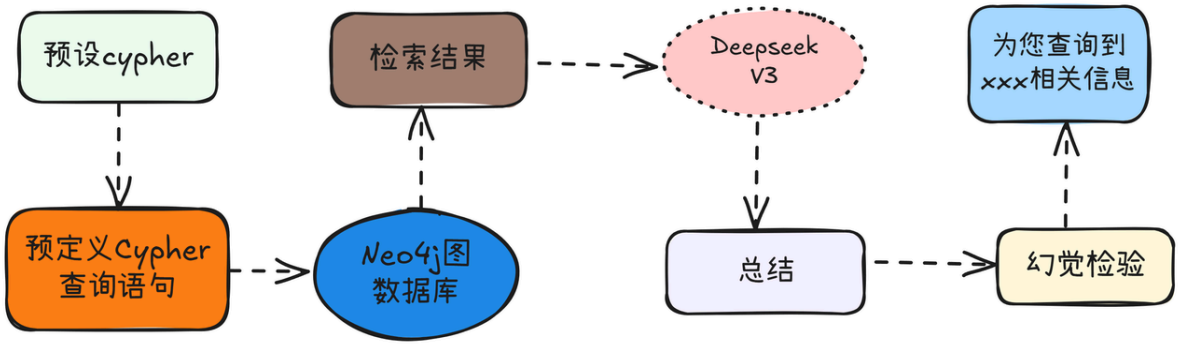
我们在处理复杂的知识库查询时，会先使用 Planner 将原始问题拆分成多个子任务。然后根据每个子任务的内容，动态选择最合适的检索方式：对于结构化数据，我们优先使用预设的 Cypher 查询（预定义 Cypher）或者让大模型根据 Few-shot 示例生成新的 Cypher（text2Cypher）；对于非结构化内容，比如产品说明书、售后政策，我们会使用 GraphRAG 进行语义级检索。所有子任务并行执行后，通过 Map-Reduce 机制聚合结果，最终形成统一的回答。这种方式既保证了回答的准确性，也提升了系统的响应效率。



7.1、预设cypher是怎么做的?

使用"人工构建+自动化补充"的方式生成预定义 Cypher 查询语句，用于处理高频率的图数据库查询需求。同时，我们为每个 Cypher 工具定义了Tool JSON Schema ，用来标准化工具调用的输入输出格式，

就是用一组预先定义好的标准 Cypher 查询语句，用于直接执行。



7.2、这些cypher语句是怎么来的？有多少条?

其中有80条是人工构建的，主要来自产品经理、运营团队提供的典型问题，还有从历史日志中提取高频问题。

此外，还用大模型 + Schema 自动生成补充了二三十条。

8、Graphrag query是怎么做的?

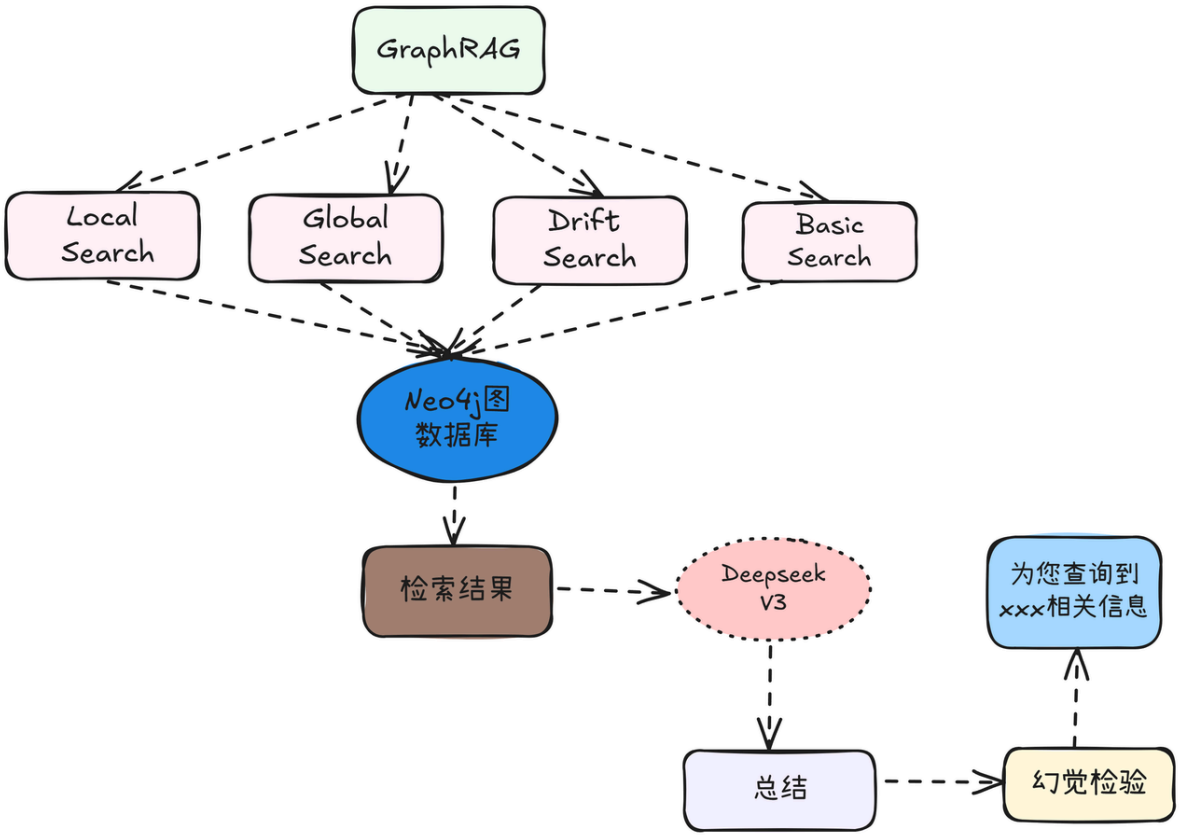
graphrag模块主要用于解决长尾知识问答类问题。预先离线构建的索引已经抽取了实体-关系、社区子图、上下文块及向量 embedding；查询时按需在 Basic、Local、Global、Drift 四种检索路径里动态挑选，Local 主要依赖实体图谱，Global 借助社区检测结果，而 Drift 则在多轮对话里根据用户上下文实时切换检索焦点 。

> 索引是离线建立的，索引内容包括：
>
> - 实体提取（如产品名、品牌、功能）
> - 社区划分（Community Detection）
> - 上下文图谱（Context Graph）
> - 向量表示（Embedding）
>
> GraphRAG 提供了四种搜索方式：`basic_search`、`local_search`、`global_search` 和 `drift_search`。
>
> - Basic Search是最基础的关键词匹配，直接对上下文内容进行相似度搜索，适用于简单的问题，如事实性查询、短句匹配。
> - Local Search利用实体图谱和上下文图谱进行局部语义增强检索，适用于实体相关问题，如“某产品的使用说明”
> - Global Search基于社区图谱进行全局语义检索，适合抽象归纳类问题，如“你们都有什么产品”
> - Drift Search动态调整检索方向，适用于多轮对话与连续推理，如“那如果我换了电池呢？”

比如，用户问：“我的 LG 智能门铃夜视效果不好，换了电池还是不亮，怎么办？” **（一条case从输入到输出的全流程，必准备）**

系统流程如下——

1. Router识别为 graphrag-query；护栏检查通过（商品售后问题）。
2. 任务分解器将问题拆成：
   a) “LG 智能门铃夜视不亮的原因” b) “换电池仍不亮的处理办法”。
3. 预定义 Cypher 中没有适配售后问答，Text-to-Cypher 亦因缺少结构化售后表而放弃；两条子任务同时落到 GraphRAG。
4. Local Search 以实体 “LG 智能门铃” + 上下文图谱，一跳命中产品说明书 PDF 片段“红外夜视灯由门铃内置锂电池供电，更换后需长按复位键 5 秒重新激活”；Basic Search 补充找到社区 FAQ “如仍不亮，请检查门前照度是否超过 50 lux”。
5. 聚合器把两条证据合成自然语言回答：“请先长按复位键 5 秒激活夜视，再确认门口环境光低于 50 lux；仍异常可联系售后更换红外灯板。”
6. Hallucinations 节点验证引用块确实存在于离线索引，确认无幻觉后发送给用户。



9、Text2Cypher是怎么做的?

Text2Cypher 模块采用"生成-验证-执行"的三步流程，先利用大模型将用户问题转为 Cypher 查询语句，再通过一系列规则和工具进行合法性验证，最后在 Neo4j 图数据库中执行查询，确保结构化知识库的准确访问。

生成 Cypher 的目标是：把用户的自然语言问题转化为一条合法的、能正确执行并返回结果的 Neo4j Cypher 查询语句 。这个过程主要分为两种方式：

- 先进行 Few-shot 示例检索 + 模板生成。就是通过构建一个高质量的预定义 Cypher 字典（Cypher Dictionary），从中检索出最相似的示例，作为大模型生成 Cypher 的参考。优先用于处理高频、结构清晰的问题。
- 利用 `neo4j_graphrag` 包 + LLM 自动生成。neo4j-graphrag 是 Neo4j 官方提供的 RAG 工具包，其中包含一个专门用于文本转 Cypher 的组件 —— `Text2CypherRetriever`。 用于处理低频、复杂组合、新出现的自然语言问题。

这两种方法的选择是通过一个工具选择模块（Tool Selection Module） 进行动态路由的。

验证 Cypher 包括五种验证方法，分别是**语法校验（Syntax Check）**、**关系方向校正（Relationship Direction Correction）**、**权限控制（No Writes Console）**、**Schema 校验（Structured Schema Check）**、**LLM 辅助校验（LLM-based Validation）**，如果检查出来有错误，会经过Error 处理与自动修正环节。

语法校验就是用Neo4j 提供的 `EXPLAIN` 命令对生成的 Cypher 进行解析，`EXPLAIN` 只分析执行计划，不真正执行查询，是一种轻量级校验方式，如果语句有拼写错误、关键字使用不当、括号未闭合等问题，会直接报错。

关系方向校正是用的 `langchain_neo4j` 提供的 `CypherQueryCorrector` 来自动检测并修正方向性错误，因为图数据库中的关系是有方向性的，比如 `(a)-[:REL]->(b)` 表示 a → b，如果方向写反了，可能查不到数据。

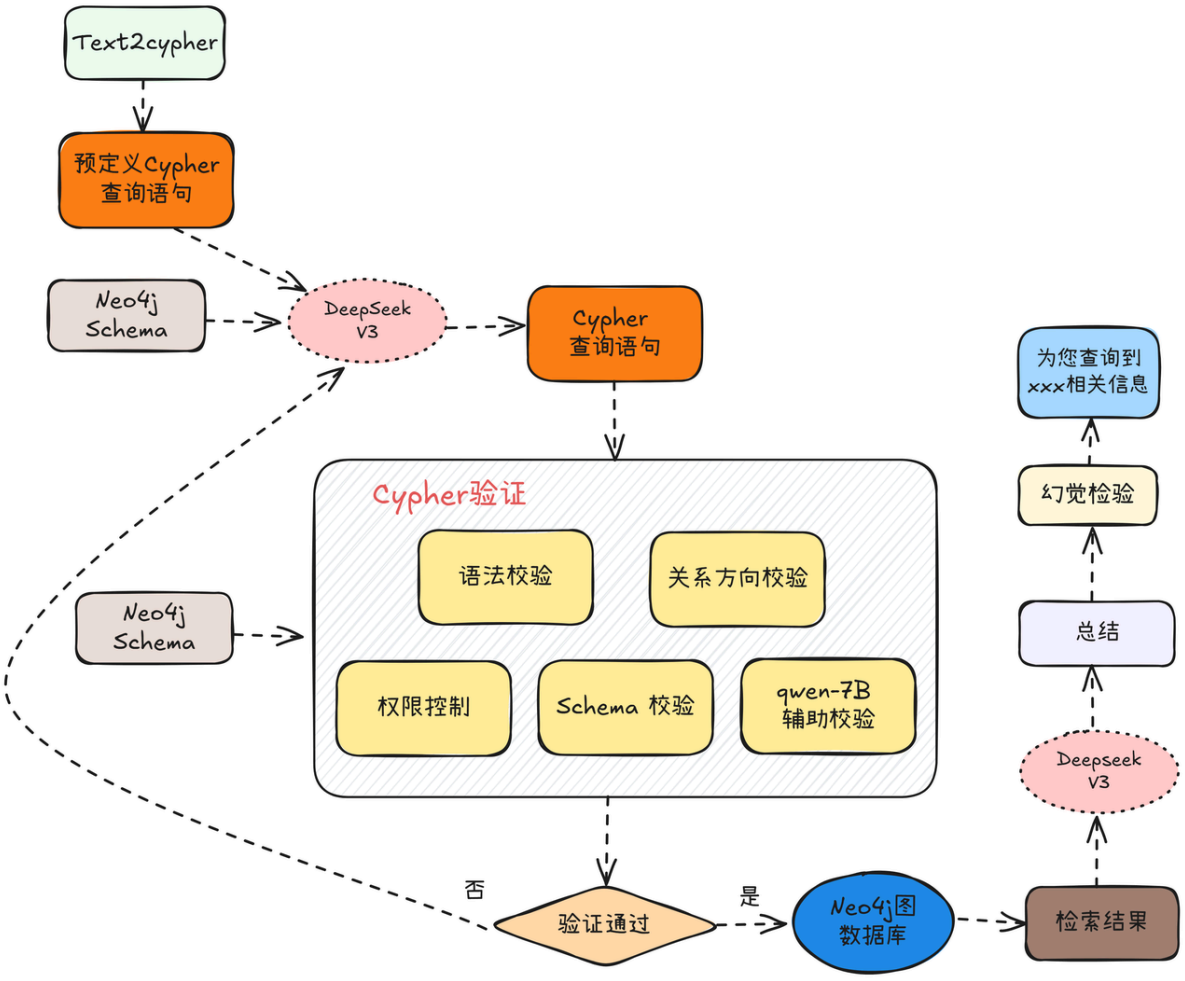
权限控制的意思是要控制大模型生成Cypher的操作类型，不允许执行破坏性操作，如：`DELETE`，`DROP`，`MERGE`，`REMOVE` 等，如果检测到这些关键词，直接拦截并反馈给用户“检测到非法删除操作。”，避免误操作导致数据丢失；

Schema 校验就是验证语句中使用的节点标签、属性名是否存在于当前数据库；例如，如果语句中写了 `RETURN p.Price`，但数据库中 Product 节点的属性是 `UnitPrice`，则应报错；这个步骤是结合 Pydantic 动态构建 Schema 模型进行自动化比对。

LLM 辅助校验是使用另一个大模型（deepseek-r1-distill-qwen-7B）来审查生成的 Cypher 是否符合语义；比如输入当前问题、Cypher 语句、Schema 信息，让模型判断是否有明显错误。

如果上面的五种验证中出现了验证失败，系统会进入自我修正流程：先记录错误类型，是error（语法错误或语义不一致），还是mapping（查询值在数据库中不存在，如 `ORDERID=AB123` 不存在）。

如果是方向错误就自动纠正，如果是字符串映射缺失，就尝试模糊匹配或提示用户提供更多信息。



9.1、为什么Cypher正确率停在88%，剩下错误的原因是什么

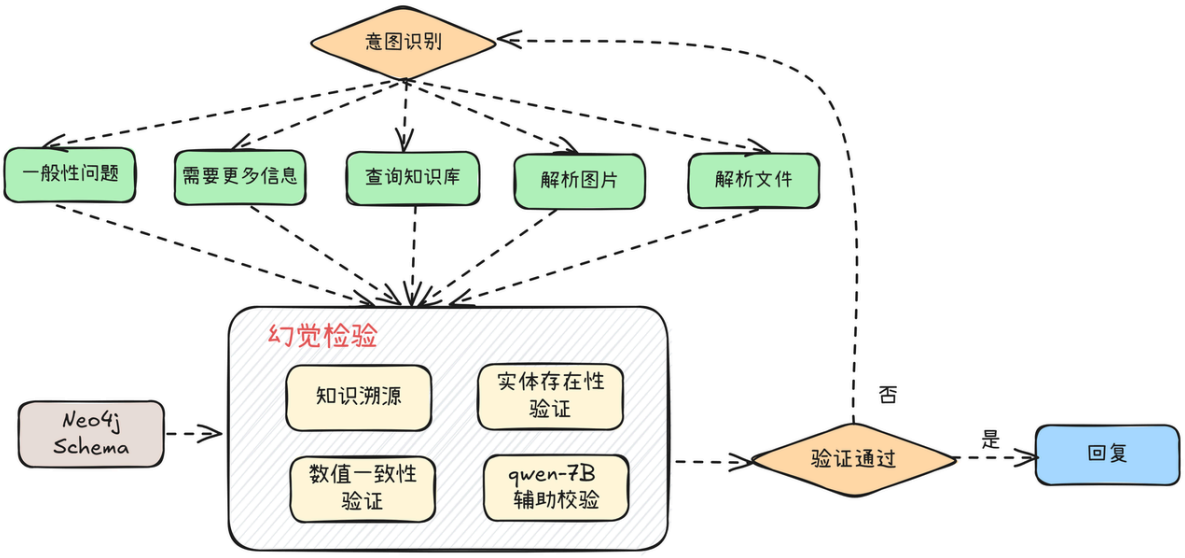
主要的bad case有以下几个类别：

- 有一部分用户问题本身表达得不清晰或带有歧义，比如“帮我查一下这个型号的剩余库存和适用配件”。这类多跳、复合或上下文依赖较强的问题，模型有时拆解不够彻底，导致 Cypher 结构不完整或者存在多余/遗漏。
- 数据库 Schema 会随着业务调整发生变化（比如新增字段、改名），但大模型缓存的 Schema 信息未及时同步，或用户用俗称、别名等指代字段，造成实体或属性映射错误。例如用户问“查一下库存单价”，实际数据库是 `UnitPrice` 字段，但模型有时无法准确对应。
- 高频、模板化场景基本都能覆盖，**长尾类的低频、组合型查询**（如带嵌套条件、时间区间、模糊匹配等）更容易出错。这一类问题不在Few-shot范畴，LLM 也容易生成不完全正确的 Cypher。
- 某些业务查询涉及多节点、多关系、深层级联查，例如：“查本周所有高于某价格且带赠品的订单”，此类查询对 Cypher 生成和优化能力要求很高，容易逻辑漏掉节点或条件写错。
- 用户输入的某些ID、订单号、SKU 等本身在数据库查无此项，语句虽然正确但查不到数据，容易被业务侧统计为“未能正确答复”。
- 个别情况下，多轮校验（如 Schema、语法、权限等）未能100%覆盖到所有异常路径，导致极个别错误语句流出。

88% 的逻辑正确率，已经覆盖了绝大多数高频场景，剩下的主要是因为复杂、模糊、多跳组合问题、实体字段匹配和业务长尾需求难以完全用模板/LLM兜底，这也是目前行业内 Text2Cypher/RAG 场景的共性难题。后续的优化空间在于不断完善 Few-shot 示例库、提升 LLM 检索和推理能力，以及加强动态 Schema 同步和多轮交互补全。

10、幻觉检测是怎么做的?

我们在客服Agent系统中加入了一个专门的 Hallucination Checker 模块，用于在最终输出前对大模型的回答进行幻觉检测。这个模块主要通过四种方式进行判断：一是知识溯源，确保回答的信息确实来自于知识库；二是数值一致性校验，比如价格、库存是否与数据库一致；三是实体存在性验证，比如提到的商品名是否真实存在；四是借助另一个小模型（deepseek-r1-distill-qwen-7B）作为裁判来判断回答是否可信。一旦发现幻觉，系统会触发自我修正机制，重新查询知识库并生成一个更准确的答案返回给用户。



10.1、为什么要用小模型作为裁判?

虽然大模型在生成能力上更强，但在幻觉检测这个任务中，小模型其实更适合做裁判。因为幻觉检测并不是创造性的任务，而是判断是否有矛盾、是否符合知识库、是否逻辑自洽。小模型在这方面不仅响应速度快、资源消耗低，而且不容易被'误导'，在事实判断上更加理性。本项目就是采用的'大模型生成 + 小模型验证'的双模型架构，在保证用户体验的同时，也提升了系统的安全性。

未来可以考虑引入统一的小模型服务层，统一处理幻觉检测、意图识别、关键词提取等判别类任务，而大模型只负责关键路径上的生成任务，进一步提升系统整体效率。

10.2、如果在前面的生成回答的过程中产生了幻觉，用模型的方法来检测不会加重幻觉吗?

您说的没错，如果幻觉检测模块本身也是基于同一类大模型，理论上会有"模型自证"的风险，也就是生成模型出的幻觉，检测模型可能因为参数或语料相似而"放行"错误内容。

但我们这个项目里，专门为了规避此类问题，做了特殊的设计：

1. **异构模型裁判，降低共振**：我们专门选用了体量更小、参数结构不同的小模型来做裁判，并且给检测模型的输入不仅有生成答案，还包括原始知识库溯源片段和事实型标签。小模型的判别能力集中在“一致性/矛盾性”，**不做开放式生成**，通常不容易和大模型共幻觉。
2. **判别任务比生成任务鲁棒**：幻觉检测本质是分类/判别型任务，尤其是做“对错”、“是否一致”、“是否出自知识库”，而不是创造内容。小模型在这类任务上的稳定性远高于生成任务，即使前面的生成模型出现了偏差，检测模型只需要关注事实匹配和逻辑一致性，容易发现问题。
3. **多通路验证，交叉比对**：我们不仅仅依赖模型裁判，工程上还结合了知识库溯源、实体和数值一致性校验等“硬规则”辅助判定，模型检测作为最后一道保险。如果知识库中能直接找到对应内容，系统就直接判为“没有幻觉”进行输出。如果前三个模块认为存在问题，也会将检索到信息交给辅助校验模块，避免出现实际语义一致，但表达不一致的情况，由llm判断是否存在幻觉。
4. **上线后持续回流难例，人工抽检优化**：实际运行中，我们会监控所有“误放”或“误杀”案例，通过难例回流和人工复核，不断优化幻觉检测的准确率，避免模型因训练局限性产生系统性漏判。
5. **未来可引入更异构的判别器**：比如基于规则、传统信息检索、知识图谱约束等多模态方法，进一步降低模型共振幻觉的风险，提升安全性。

10.3、知识溯源、数值一致性校验、实体存在性验证分别是怎么做的?

这三个模块都是基于规则实现的，并没有使用大模型。输入是模型回复和之前步骤中graphrag检索到的和text2cypher查询到的内容的列表。

知识溯源：将模型回复拆分为句子，计算每句模型回复和列表中内容之间的余弦相似度，判断每句是否与检索文档中某部分高度一致（例如相似度 > 0.8）。如果超过30%的句子没有来源，则不通过知识溯源检测。

数值一致性校验：基于正则表达式识别出回复中所有的数值，如果每个数值都能在列表中找到，则通过数值一致性校验。

实体存在性验证：使用bert-base-chinese-ner模型（110M参数），从回复中抽取neo4j schema中存在的属性内容，如人名、地名、商品名等实体，如果所有实体都能在列表中找到，则通过实体存在性检验。

11、这个项目中有没有做微调？（原代码未作微调）

Cypher 验证那里用LLM辅助验证时用到的qwen-7B 和 最后幻觉检测用到的qwen-7B 都经过了全量微调。

11.1、Cypher 验证为什么要微调?

主要是为了解决通用大模型在垂直领域任务中的适配性问题。

以Cypher验证为例，电商场景的Neo4j数据库结构和查询逻辑具有高度特异性，例如商品、订单、供应商等实体关系与通用知识图谱存在显著差异。通用模型虽然能处理基础语法，但对业务专属的节点类型、属性约束（如库存单位命名规则）或高频查询模式（如“根据会员等级动态计算优惠”）缺乏深层理解，可能导致漏判语义错误或误判合法查询。通过微调，可以让模型从大量标注的领域查询样本中学习到业务专属的Cypher模式，例如识别“BELONGS_TO关系的反向使用是否合法”“价格区间过滤是否与数据库字段类型匹配”等细粒度规则，从而提升验证准确率。

微调后**语义错误检出率**从微调前的78%提升至93%，尤其在业务专属规则（如关系方向合法性、属性字段匹配）的判定上，准确率提升超过20%。例如，针对“BELONGS_TO关系反向使用”这类领域特异性错误，漏判率从35%降至5%。

误判率（False Positive）从15%降至4%，避免了对合法查询（如动态优惠计算逻辑）的过度拦截。

11.2、Cypher 验证的微调数据集怎么构造的?

输入是当前问题、Cypher 语句、Schema 信息，输出是true/false。

在Cypher验证任务中，我们从历史查询日志中提取正样本（正确执行的查询），同时通过规则注入生成负样本——例如，随机打乱节点关系方向（将“Product→SUPPLIED_BY→Supplier”改为“Supplier→SUPPLIED_BY→Product”）、伪造不存在的属性字段（如将“UnitsInStock”改为“StockQuantity”），并邀请领域专家标注错误类型（语法/语义/映射）。

在Cypher验证任务中，我们从历史查询日志中提取了约3万条正样本（正确执行的查询），这些数据来源于实际业务中用户和系统生成的合法Cypher查询，例如订单状态查询、商品检索等。负样本通过规则注入生成，总计约3万条，包括随机打乱节点关系方向（如将“Product→SUPPLIED_BY→Supplier”改为“Supplier→SUPPLIED_BY→Product”）、伪造不存在的属性字段（如将“UnitsInStock”改为“StockQuantity”），以及引入语法错误（如缺失关键字或括号不匹配）。负样本中，语义错误占50%（约1.5万条），映射错误占30%（约1万条），语法错误占20%（约7千条）。所有负样本均经过领域专家审核并标注错误类型，确保数据质量。训练集、验证集、测试集按8:1:1划分，即训练数据4.8万条，验证和测试数据各6千条。

测试采用自动化评估与人工抽检结合。使用测试集的6000条数据计算语义错误检出率、误判率等指标，并通过A/B测试对比微调前后模型性能。人工抽检覆盖一部分的测试数据（1000条），由3名标注专家独立审核，标注一致性需达到95%以上。争议样本通过多数投票或领域专家仲裁解决。测试结果显示，Cypher验证的**语义错误检出率**从78%提升至93%。

11.3、这些指标都是怎么评估的?

语义错误检出率

即模型成功识别出语义错误样本的比例。

在测试集的负样本（人工标注为语义错误的）上，让模型进行判别。如果模型输出为“错误”，且与人工一致，记为TP，如果模型未能识别（输出“正确”），则为FN，计算：**语义错误检出率 = TP / (TP + FN)。**

误判率

即模型错误地将“合法样本”判为“有错误”的比例。

在测试集的正样本（人工标注为合法查询）上，让模型判别。如果模型错误判为“错误”，为FP，如果模型正确判为“正确”，为TN，计算：**误判率 = FP / (FP + TN)。**

11.4、幻觉检测为什么要微调?

对于幻觉校验模块，电商场景的客服回答需严格遵循商品详情、促销政策等结构化数据，但通用模型缺乏对业务知识库边界和动态数据（如实时库存）的感知。例如，当用户询问“智能音箱是否支持蓝牙5.3”时，若知识库中仅记录“蓝牙5.2”，通用模型可能因语义近似而错误放行回答。通过微调，模型能更精准捕捉实体属性一致性（如型号后缀差异）、数值时效性（如“限时折扣已过期”）等关键风险点，而非依赖通用语义相似度。

微调后**数值一致性校验**的精确率从82%提升至96%，例如能准确识别“商品价格±5%浮动”是否与数据库记录一致（微调前易被模糊语义干扰）。

实体存在性验证的召回率从70%跃升至89%，尤其是对长尾商品型号（如“小米智能音箱Pro X12”）的虚构检测能力显著增强。

整体幻觉漏检率从25%优化至8%，在促销政策时效性（如“活动已结束但回答仍提及优惠”）等动态场景中表现尤为突出。

11.5、幻觉检测的微调数据集怎么构造的?

对于幻觉校验，则基于真实用户对话构建三元组数据，包含“用户问题-原始回答-修正回答”，通过人工标注和自动化对抗生成（如替换回答中的商品参数、虚构促销活动）来构建存在数值偏差、实体虚构、政策过时等幻觉类型的样本。此外，针对高频业务场景（如退换货流程），还会通过知识库快照对比生成动态负例，例如将回答中的“7天无理由退货”改为“15天”，与知识库当前政策不一致的样本即标记为幻觉。这种数据构建方式确保了模型能覆盖业务核心风险场景，同时通过持续集成最新工单数据实现迭代优化。

对于幻觉校验任务，数据来源于真实用户对话工单和客服历史记录，总计约2.4万条。通过人工标注构建“用户问题-原始回答-修正回答”三元组，其中数值偏差样本占35%（如价格、库存与数据库不一致）、实体虚构占30%（如虚构商品型号）、政策过时占25%（如引用失效促销活动）、逻辑矛盾占10%（如回答自相矛盾）。此外，通过自动化对抗生成技术，将知识库中的真实回答随机篡改关键参数（如将“7天无理由退货”改为“15天”），生成约3万条动态负例。训练集、验证集、测试集按8:1:1划分。

测试采用自动化评估与人工抽检结合。通过自动化脚本验证回答与数据库的一致性，例如实时查询商品价格、库存等字段。人工抽检覆盖一部分的测试数据（500条），由3名标注专家独立审核，标注一致性需达到95%以上。争议样本通过多数投票或领域专家仲裁解决。测试结果显示，幻觉校验的整体漏检率从25%降至8%。

11.6、这些指标都是怎么评估的?

数值一致性校验准确率

即模型判断“数值信息”与数据库记录一致的准确率。

对涉及价格、库存、促销等数值的回答，自动与数据库对比。若模型判断与数据库一致，且事实也一致，为TP，若模型判断为“不一致”，且实际也不一致，为TN，错判则为FP或FN。计算：**准确率 = (TP + TN) / 总数。**

实体存在性验证召回率

即模型能识别出“回答涉及不存在实体”样本的比例。

在测试集中，人工标注为“实体虚构”（如虚构商品型号）的负例样本。模型输出为“实体不存在”即为TP，输出为“实体存在”即为FN。计算：**召回率 = TP / (TP + FN)。**

整体幻觉漏检率

即模型未能发现幻觉（如虚构、过时、数值不一致等）的比例。

对测试集所有“有幻觉”负例，让模型判断。若模型未能发现幻觉（输出“正常”），则为FN，若正确发现为“幻觉”，为TP。计算：**漏检率 = FN / (TP + FN)。**

11.7、为什么使用这三种指标不用其他指标?

这三项指标分别直接覆盖了我们在电商客服知识场景中最易出错的风险点——**数值失实（如价格库存）、实体虚构（如型号、品牌）和整体幻觉遗漏。**这样可以最大程度与实际业务事故和客户体验损失直接挂钩，评估结果更具落地价值。

这三类指标计算方法清晰、评判标准一致，非常适合自动化批量评测和线上监控，便于模型快速迭代优化，人工复核时也容易达成共识。

我们的幻觉检测属于“判别型”任务，这些指标天然与**二分类/异常检测问题**高度适配，比如准确率反映业务容错，召回率保障低漏检，漏检率则能定量监控系统风险。

像BLEU、ROUGE等文本生成类指标，或AUC、F1等通用指标虽然有参考价值，但在我们这种强约束、业务驱动的判别场景下，不能精细反映各类高风险幻觉的实际影响，因此我们优先选择这三项。

12、有哪些bad case？/后续怎么优化？（必准备）

多跳查询的拆分与执行偏差

用户提问“订单456的收货地址是否在小米智能家居的覆盖区域？”，系统需先查询订单地址，再关联供应商服务范围。初期因Planner子任务拆分逻辑不完善，可能跳过“供应商服务区域”检索，直接返回订单地址信息，导致答非所问。此类问题源于任务拆解时对隐式依赖关系的忽略，后续通过增强Planner的上下文感知能力（如动态注入关系图谱元数据）优化。

跨模态数据融合的局限性

用户上传图片提问“帮我检查订单789的商品是否与这张说明书兼容”，需同时解析图片中的型号信息并关联订单数据。早期因OCR识别误差（如将“Model X12”误识别为“Model X2”）导致检索结果错误，或非结构化文档与结构化订单的关联逻辑不完善，返回“兼容性未知”。通过引入多模态对齐校验（如图文联合Embedding）和容错重试机制减少此类问题。

复杂逻辑的过度简化

用户提问“如果退货后重新下单，能否保留原优惠券？”，涉及订单状态、促销策略、用户权益等多条件判断。早期系统可能仅依赖单一规则（如“优惠券一次性有效”）直接拒绝，而忽略“特殊活动允许复用”的例外情况。通过增强策略引擎的动态规则加载能力（如实时读取促销政策文档）和引入人工审核兜底机制，降低误判率。

13、接入Agent过后，用户问题的解决率有多少？

接入多Agent体系之后，我们的用户问题**一次性自动解决率**有了明显提升。上线初期，我们用人工标注日志+抽样人工回访的方式评估解决率，
在早期纯FAQ方案下，一次性自动解决率只有约 45%-50%，许多复杂/复合型/结构化问题需要人工介入。

升级为RAG知识库后，解决率提升到60%左右，但还存在幻觉和复杂多跳场景命中率低的问题。

而在引入LangGraph的Multi-Agent体系，完善意图识别、图数据库结构化检索和安全防护后：

- 整体自动解决率（即用户无需转人工，直接满意离开）稳定在 78%~82%。
- 复杂/多跳/结构化问题的自动解决率比RAG方案提升了 20% 左右。
- 剩余未能自动解决的部分，大多为超出经营范围或法律合规限制的需求（如灰产、过度索赔、产品外部政策等），这些都能通过安全节点自动识别并妥善拒答。

此外，幻觉率大幅下降，误答率压到业务可接受的4%以内。我们每月有定期复盘机制，发现新型未覆盖场景就反补进Agent和检索流程，不断迭代。

14、multi-agent之间是怎么交互协作的？为什么使用multi-agent，单一agent不可以解决吗？multi-agent有没有出现什么问题？

（其实面试官问出这几个问题，说明他并没有认真地听我们前面的项目介绍，或者说听了他也没有很理解，所以我们把前面说过的内容按照原来的思路再复述一下解释清楚就可以。最后一个问题，他问出来，说明他最近也看了很火的那篇论文Why Do Multi-Agent LLM Systems Fail，《LLM知识体系搭建》里面4.6节有相应的论文解读）

multi-agent之间是怎么交互协作的？

在我们的LangGraph Multi-Agent体系中，首先由**意图识别Agent**对用户问题进行类别判断，决定主流程。对于其中的graphrag-query类型，会交给**Planner Agent**将复杂问题拆分为子任务。每个子任务交由合适的**工具Agent**（如Text2Cypher或GraphRAG Agent）并行处理。各Agent之间通过结构化JSON消息进行数据传递，保证信息闭环和可追溯。处理完成后，结果再由**汇总Agent**统一整理，并经过**安全与幻觉检测Agent**复核，最终返回给用户。LangGraph的graph workflow可以支持我们灵活地将不同类型的任务串行或并行调度。

单一agent不可以解决吗？

我们尝试过单一大模型Agent方案，比如早期的FAQ、后面尝试的RAG、或者单独使用graphrag，都可以看做是单一Agent的方案，但发现其在多步推理、复杂流程调度时难以保障准确性和可控性，比如结构化和非结构化检索无法并行，多跳问题的处理经常混乱。

采用multi-agent体系后，可以将每个关键能力“组件化”，如意图识别、任务拆解、结构化查库、内容审核等，都由专属Agent负责，各Agent间通过LangGraph组织成高效的分布式工作流。这样不但提升了准确率、并发处理能力，还便于后续能力升级和插拔新功能。

multi-agent有没有出现什么问题?

multi-agent体系存在一些问题，比如最早期我们遇到过Agent之间上下文传递不规范，有时Planner输出的子任务描述不够清晰，导致工具Agent执行时出错。

为此我们严格限定了Agent之间的输入输出格式，全部用结构化JSON传递，并且定义了schema校验流程。

此外，初期串行处理导致延迟较高，我们后来通过LangGraph的map-reduce并行机制大幅提升了速度。

还有就是各Agent边界不明时，出了问题很难定位，所以我们在每个环节加了详细日志和链路追踪，方便溯源。

15、如果检索到的内容与用户提问不符怎么办

我们的安全检测模块、幻觉检测模块就是用来解决这个问题的。假如用户问的是篮球，而数据库里没有篮球，正常情况下在安全判断那一步就应该判断否，回复用户“抱歉我们没有篮球”。如果安全判断没问题，也就是说我们的数据库里有篮球，但却回复了用户足球相关的东西，这可能是embedding模型语义检索效果差，可以对embedding模型进行微调或者蒸馏；还可以引入rerank；可以调整知识图谱检索跳数，检索策略（比如什么问题应该用local search）；检查图数据库是否存在问题，比如是否把足球和篮球分为两个实体。

16、业务相关的指标成果有哪些?

客服机器人自动应答率由原来的 **52% 提升到 81%**，大幅减少人工介入，日均人工接入量由 **2300 条降到 800 条以内。**（Agent优先原则，即每一条咨询，Agent会先尝试应答，只有遇到超出业务范围、意图不明确、强制人工类场景（如投诉、敏感问题等），才由Agent主动交给人工。这一策略原本就有，但过去由于智能客服能力有限，很多复杂咨询都要转人工）

客服自动化带动下，用户**下单转化率提升了 3.7%**，**复购率提升了 2.4%**

客户满意度得分从3.5提升到4.2（智能客服交流完毕后会让用户评分，在5个小心心里面点亮）。

17、了解哪些压力测试的指标

1. **性能指标:**
    - **响应时间/延迟 :** 从发送请求到收到完整响应所需的时间。这是用户体验的关键指标。会关注平均延迟、P90/P95/P99 延迟（即 90%/95%/99% 的请求响应时间低于某个值）。
    - **吞吐量 :** 系统在单位时间内能够成功处理的请求数量，通常用 QPS (Queries Per Second) 或 TPS (Transactions Per Second) 来衡量。这是衡量系统处理能力的核心指标。
    - **并发用户数 :** 系统能够同时处理多少个用户的请求。
2. **系统稳定性与可靠性 :**
    - **错误率:** 在高压下，请求处理失败（如超时、返回错误码）的比例。

    - **系统崩溃/宕机情况:** 在极限压力下，服务是否会崩溃，能否自动恢复。
    - **长时间运行稳定性:** 在持续高压下运行一段时间（例如几小时甚至几天），系统是否依然稳定。
3. **资源利用率:**
    - **CPU/GPU 使用率:** 模型推理主要依赖计算资源，监控其使用率可以发现计算瓶颈。
    - **内存/显存使用率:** 大模型通常需要大量内存/显存，监控其使用情况防止 OOM (Out Of Memory) 错误。
    - **网络带宽:** 请求和响应数据的传输会消耗网络带宽。
    - **磁盘 I/O:** 如果模型加载或数据处理涉及磁盘读写，也需要关注。
4. **可扩展性:**
    - 测试系统在增加资源（如增加服务实例、GPU卡数）后，性能指标（如吞吐量）是否能相应提升。

17.1、怎么做的压力测试?

我们在Agent系统上线后做了系统性的压力测试，主要目标是评估大模型多Agent流程在高并发场景下的稳定性和性能瓶颈。具体做法如下：

- **模拟多用户并发请求**：使用requests请求，模拟数百到上千用户同时访问，发起批量文本问答、检索和复杂查询。
- **测试真实业务流**：压力测试用例覆盖了最常见的业务主流程（如意图识别、知识库查询、多跳推理、幻觉检测），还包含极端场景，比如连续大批量复杂查询。
- **逐步提升压力**：从小并发（5 QPS）逐步提升到高并发（20 QPS及以上）.

17.2、有哪些具体测试结果?

- **系统响应时间**：平均响应时间2.1秒，P95 2.7秒，P99 3.6秒。（主要时间消耗在调用deepseek API上，部署的deepseek-r1-distill-qwen-7B推理很快）
- **吞吐量**：最大QPS能支持每秒20请求，平均每天请求大概5000~10000条，吞吐量能满足日常需求。
- **系统稳定性**：高并发场景下错误率低于1%，主要是因为网络请求超时。
- **资源利用率**：生产环境部署了4张A100显卡, GPU利用率平均65%，未出现OOM。

> 18的两个问题中的很多方法都是没有实现的，很多都是开发和工程方面的优化，主要针对这位同学的问题 居丽叶的简历项目1：智能客服Agent，面试官深入追问router/planner节点的细节时再说。

18.1、Router 节点有哪些未来的优化点更深入的细节/未来优化点?

1. **多层次意图识别体系**

- **主分类器采用DeepSeek v3**：选择这个模型是因为其在指令遵循和结构化输出方面表现优异，特别是在JSON格式输出的稳定性上。模型通过API调用，设置temperature=0.1保证输出的一致性，同时启用logprobs参数获取概率分布用于置信度计算。

- **五类意图的设计逻辑**：系统将用户查询分为general-query（占比约5%）、additional-query（15%）、graphrag-query（65%）、image-query（10%）、file-query（5%）。这种分布是基于三个月的历史数据统计得出，其中graphrag-query占比最高反映了电商场景下产品咨询的复杂性。
- **混合识别策略**：前三类使用大模型判断，后两类（图片和文件）采用规则引擎。
- **置信度评估的多维度机制**
- **Logprobs差值分析**：通过计算最高概率标签与次高概率标签的logprobs差值，经过sigmoid归一化得到0-1之间的置信度分数。差值越大说明模型对分类结果越确定，这比单纯使用softmax概率更能反映模型的判断确定性。
- **关键词规则增强**：维护了一个包含500+个业务关键词的词典，如"订单号"、"退货"等强信号词。当查询包含这些关键词时，会对相应意图的置信度进行加权调整（权重系数1.2-1.5）。
- **Few-shot示例的智能管理**
- **动态示例池维护**：系统维护一个容量为200条的示例池，实际使用时根据查询特征动态选择最相关的5条作为few-shot。使用FAISS构建向量索引，实现毫秒级的相似示例检索。
- **示例质量评分机制**：每个示例都有质量分数，包括：使用频率（被检索次数）、效果反馈（分类准确率）、时效性（添加时间）三个维度。低分示例会被自动淘汰。
- **难负样本挖掘的自动化流程**
- **多源数据收集**：从Kafka消息队列实时收集用户查询日志，从MySQL存储人工标注结果，从Redis缓存高频误分案例。每天处理约5000-10000条查询记录。
- **误分类模式识别**：使用Spark Streaming进行实时日志分析，通过滑动窗口（1小时）检测误分类率突增。当某类误分类在窗口内超过20次，自动触发样本收集。
- **主动学习策略**：对于置信度在0.6-0.8之间的边界样本，系统会主动请求人工标注。使用uncertainty sampling选择最有价值的样本，每天推送50条给标注团队。
- **容错与降级策略**
- **多模型冗余备份**：除主模型DeepSeek v3外，还部署了Qwen-7B作为备用模型。当主模型响应超时（>2s）时，自动切换到备用模型。
- **规则引擎兜底**：维护了覆盖80%高频场景的规则库（约300条规则），当模型服务完全不可用时，降级到纯规则分类，保证基础可用性。

18.2、Planner 节点有哪些未来的优化点更深入的细节/未来优化点?

1. **任务分解的核心策略**

- **语义依赖图构建**：Planner首先将复杂查询解析成语义依赖图（DAG），识别子任务间的依赖关系。比如"查询某产品的库存并对比同类产品价格"会被解析为两个可并行的子任务。系统使用spacy的依存句法分析提取查询中的核心动词和宾语，构建任务之间的前后依赖关系。
- **任务粒度控制机制**：通过预定义的任务模板库（包含150+种常见任务模式），Planner会评估分解粒度。过细的分解会增加调度开销，过粗则无法充分并行。系统设定每个子任务的理想执行时间为0.5-2秒，据此动态调整分解策略。
- **意图链路分析**：对于多跳查询如"查看上周订单中已发货但未签收的商品"，Planner会识别出时间过滤→状态筛选→商品提取的链式依赖，确保子任务按正确顺序执行。使用图算法（拓扑排序）确定最优执行序列。

- **智能任务编排机制**
- **并行度评估算法**：Planner会分析每个子任务的计算复杂度和数据依赖，计算最大并行度。对于IO密集型任务（如数据库查询）设置较高并行度（最多5个），对于计算密集型任务限制并行度避免资源争抢（这一点本项目不涉及，都是要查询数据库）。
- **动态资源分配**：集成k8s的HPA（Horizontal Pod Autoscaler），根据任务队列长度和资源使用率动态调整Worker数量。高峰期自动扩容到20个Worker Pod，低峰期缩容到5个，实现资源的弹性利用。
- **Map-Reduce执行框架**
- **Map阶段的并行调度**：基于LangGraph的并行执行能力，Planner将独立子任务分发到不同的执行节点。使用消息队列（RabbitMQ）进行任务分发，每个Worker维护本地任务队列。
- **中间结果缓存策略**：Map阶段产生的中间结果存储在Redis中，设置TTL为300秒。对于相同的子查询，直接从缓存返回结果。缓存key使用查询内容的MD5哈希，命中率约25%，显著减少重复计算。
- **子任务预先填充**
- **与Text2Cypher的协同**：当子任务涉及结构化查询时，Planner会生成查询模板，预填充部分参数，减少Text2Cypher的生成难度。同时传递查询上下文，帮助生成更准确的Cypher语句。
- **GraphRAG任务优化**：对于需要GraphRAG的子任务，Planner会预判需要的检索模式（local/global/drift），并提前触发相关索引的加载。通过任务特征（如是否包含"对比"、"总结"等关键词）选择最适合的检索策略。
- **结果格式标准化**：Planner定义了统一的中间结果schema，包含data、file、confidence等字段。所有Tool返回的结果都要符合这个schema，便于Reduce阶段的处理。使用JSON Schema进行运行时校验。

5. **容错与恢复机制**

- **子任务重试策略**：采用指数退避算法，首次失败等待1秒重试，之后按2、4、8秒递增，最多重试3次。对于不同类型的错误有不同策略：网络超时立即重试，语义错误不重试直接降级。
- **部分失败处理**：当部分子任务失败时，Planner会评估是否影响最终结果。如果是非关键信息（如补充说明），会继续返回部分结果并标注信息不完整。如果是核心信息缺失，会触发降级策略或转人工。
- **断点续传能力**：对于执行时间较长的复杂查询，Planner支持断点续传。将执行状态持久化到Redis，即使服务重启也能从中断点继续执行。每个checkpoint包含已完成任务、待执行任务、中间结果等完整信息。

八股类问题

embedding模型用的哪个?

bge-m3

介绍一下bge-m3

在模型架构方面，BGE - M3 和常规的embedding模型一样，是双塔结构，采用了一种基于 XLM-RoBERTa 并结合 RetroMAE 方法的架构，以更好地捕捉文本中的语义信息，能够处理长达 8192 的输入文本，这对于处理长文档和复杂语义关系非常关键，展现了其强大的长序列建模能力。

在训练策略上，BGE - M3 运用了多种训练技术来提升模型性能。它采用了自激励蒸馏方法，这种方法有助于模型在训练过程中更好地学习到语义表示，提高模型的泛化能力。同时，模型经过了多阶段的训练，包括 RetroMAE 预训练、无监督对比学习以及多检索方式统一优化等。通过在大规模的多种数据上进行训练，模型能够学习到丰富的语言知识和语义模式，从而在不同的任务和领域中都能表现出色。

在语言处理能力方面，BGE - M3 支持超过 100 种语言，这得益于其精心设计的多语言编码机制。它能够将不同语言的文本映射到统一的语义向量空间中，实现跨语言的语义理解和检索。这对于处理全球化的文本数据和构建多语言应用非常有帮助，体现了其强大的跨语言语义对齐能力。

在检索功能实现上，BGE - M3 集成了稠密检索、稀疏检索和多向量检索三大能力。通过巧妙地融合这三种检索方式，模型能够在不同的语义检索场景中发挥优势。例如，在语义搜索中，稠密检索可以快速找到与查询语义相似的文本；在关键字搜索中，稀疏检索能够利用关键词的稀疏表示提高检索效率；而多向量检索则可以从多个角度对文本进行表示和检索，进一步提高检索的准确性。并且，BGE - M3 能够在一次推理中同时输出多种检索模式的结果，无需额外的计算开销，这体现了其高效的推理机制和良好的工程实现。

为什么选择bge-m3作为embedding模型?

因为这个项目的第一个版本——就是依赖于FAQ那个版本中，使用到的embedding模型是**Sentence-BERT**，时过境迁，需要更换为更新的embedding模型，主要依赖于huggingface上一个公开的embedding模型排行榜——MTEB，参考这个排行榜同时考虑以下几个因素我选择了bge模型：

1. **语言支持和性能**：bge-m3 支持 100 多种语言，适合多语言需求的场景。
2. **模型在特定领域的表现**：在**业务数据集**上测试了排行榜前二十的embedding模型，并用抽样的方式计算了Recall@N，综合来看bge-m3在业务数据集上表现最优。
3. **存储和内存等资源需求**：bge模型大小是568M，支持 8192 tokens 的输入长度，max token是8192，在embedding模型中属于中等大小的参数量，上下文长度对于当前场景来说也足够。
4. **模型响应时间**：也评估了query从输入到输出为embedding过程中的延迟，测试了1000条query取平均延迟约为200ms。

综合以上四个因素选择了bge-m3。

追问：业务数据集如何构造

总共构造了5000条样本，内容主要来自实际电商客服日志、FAQ文档和产品说明书。其中80%是人工标注的高质量语义相似对（之前训练Sentence-Bert用的），剩下20%是利用自动数据增强（如同义改写、GPT重写和文本扰动等）扩充得到。正负样本的比例大约是3:1。

我们在所有业务数据上，分别测试了排行榜前二十的主流embedding模型，包括bge、gte、m3e等。

在评测过程中，我们从总数据集中**随机抽取500条样本**作为评估子集。对于每一条测试样本，都用其问题部分（query）到整个数据集中进行embedding向量召回，检索出相似度最高的前K（K分别取1、3、5）条候选结果，并与人工标注的正确答案进行对比。以此分别计算recall@1、recall@3、recall@5，最后对500条样本的召回结果取平均，得到整体的Recall@N指标。

除了Recall外，我们还同步计算了**NDCG@N（Normalized Discounted Cumulative Gain）**，衡量模型在召回列表中将相关结果排到前面的能力。具体做法是对每条query的候选列表打分，相关性越高的排位分值越大，然后用NDCG公式归一化统计每个模型的排序表现，再对500条样本平均。这样能更全面反映模型在“相关性排序”上的综合效果。

最终，通过这一套业务数据集和评测流程，综合recall和NDCG等多项指标，bge-m3模型在我们业务场景下表现最优，无论是多语言、长文本还是语义复杂度，bge-m3的Recall@3和NDCG@5都领先于其他模型，整体效果提升显著。因此最终选择了bge-m3作为当前主力embedding模型，全面替换原有的Sentence-BERT版本。

介绍下XLM-RoBERTa

XLM-RoBERTa是Facebook AI研发的多语言预训练模型，它以RoBERTa为基础，运用自监督学习，在海量多语言文本数据上训练，规模超此前的多语言模型。这种训练方式，让它能学习不同语言的语法、语义和结构模式，构建统一的语言表示空间。

RoBERTa 摒弃了 BERT 使用的固定长度掩码模式，采用动态掩码机制。训练时，每次输入数据都会重新生成掩码，模型能在不同训练步骤学习不同掩码下的文本表示，加深对语言结构和语义的理解。同时，增加训练数据量，使用了 BookCorpus、CC-NEWS 等大规模语料库，让模型学习到更丰富的语言知识和语言模式，提升泛化能力。训练过程中，RoBERTa 调整了训练超参数，如学习率、训练步数等，使模型训练更高效，收敛更快，能更好地捕捉文本中的复杂语义和语法信息。

介绍下retroMAE

RetroMAE也是embedding模型，依然是双塔的设计。

对输入句子使用两种不同掩码分别污染编码器和解码器的输入。基于编码器的掩码输入生成句子嵌入，再结合解码器的掩码输入，通过掩码语言建模（MLM）恢复原始句子，增强了对编码质量的要求 。

编码器采用完整规模的类似BERT的Transformer，能够为输入句子生成具有判别力的嵌入；解码器则是极为简化的单层Transformer，专门用于重建输入句子 。

编码器输入的掩码比例为15% - 30%，略高于传统MLM中的比例；解码器输入的掩码比例高达50% - 70%，提高了重建任务的难度，使模型学习到更鲁棒的特征 。

全量微调7B的模型需要多大显存?

120G，用了2张A100的显卡做多卡并行训练1个epoch，训练用deepspeed zero3，训练12个小时。

方法	精度	7B	13B	30B	70B	8x7B	8x22B
Full	AMP	120GB	240GB	600GB	1200GB	900GB	2400GB
Full	16	60GB	120GB	300GB	600GB	400GB	1200GB
Freeze	16	20GB	40GB	80GB	200GB	160GB	400GB
LoRA/GaLore/BAdam	16	16GB	32GB	64GB	160GB	120GB	320GB
QLoRA	8	10GB	20GB	40GB	80GB	60GB	160GB
QLoRA	4	6GB	12GB	24GB	48GB	30GB	96GB
QLoRA	2	4GB	8GB	16GB	24GB	18GB	48GB

deepspeed

利用deepspeed的 ZeRO进行内存优化并实现多卡并行训练

- **Stage 1：分割优化器状态** ZeRO 的第一个阶段将优化器状态（如 Adam 优化器中的一阶和二阶动量）分布到多个 GPU 上，减少每个 GPU 上的冗余内存占用。
- **Stage 2：分割梯度** 在此阶段，不仅优化器状态被分割，模型的梯度也会在多个设备之间分割和存储，进一步减少显存开销。
- **Stage 3：分割参数** 在最后的阶段，模型的所有参数也会被分布到各个 GPU 上，这样每个 GPU 只保存一部分参数，从而显著节省显存。这一阶段使得单个 GPU 能够训练数千亿参数的模型。

场景题

何时使用Agent，何时要接入人工客服?

可以从以下几个角度考虑：

可以通过实时情感分析模型（如基于语音/文本的情绪识别）监测用户情绪波动。若检测到用户处于愤怒、焦虑等高敏感状态（如连续重复提问、负面关键词频发），即使问题理论上可由Agent解决，也需优先转人工以避免体验恶化。同时，对涉及人身安全、资金欺诈等高危场景（例如“我被骗转账了”），需通过关键词触发规则强制转人工，确保风险可控。

在流量高峰时段，Agent可作为缓冲层处理80%的简单请求，降低人工队列压力。此时需结合排队论模型动态调整路由策略：当人工客服平均等待时间超过阈值（如3分钟），将部分边界案例（如置信度低于预设的意图识别结果）暂由Agent接管，并通过强化学习实时优化分流比例。反之，在低峰期可适当放宽转人工阈值以提升问题解决率（FCR）。

若用户历史行为表明其偏好人工服务（如曾多次主动要求转人工），或当前会话涉及复杂上下文继承（例如“修改我刚提到的订单的收货地址，但保留之前的支付方式”），需通过用户画像模型和对话状态跟踪（DST）判断Agent的上下文理解是否足够鲁棒。对于长尾个性化需求（如定制化产品咨询），Agent可尝试提供初步建议，并在生成结果置信度低于阈值时提示人工介入。

对于法律、医疗等强监管领域（如“工伤赔偿计算”“药物副作用咨询”），即便Agent能生成技术性回答，也需强制转人工以避免合规风险。此时需设计双层验证机制：先通过实体识别（NER）检测敏感领域，再通过规则引擎限制Agent输出，并返回标准化转接话术（如“该问题需由专业顾问为您服务”）。

最终决策需依赖在线学习机制：通过记录用户对Agent服务的满意度（如五星评分）、人工介入后的解决时长等指标，持续优化意图分类、情感分析等模型。例如，对频繁触发人工接管的问题聚类分析，反向扩充Agent知识库或调整路由阈值，形成“数据收集-模型更新-AB测试-全量部署”的闭环优化链路。

##