

RAG 技术详解与实战应用

第15讲：大视角问答：

RAG如何支持跨文档、跨维度总结



目录



1. 上节回顾



2. RAG解决宏观问题基本思路



3. 核心模块原理及实现思路

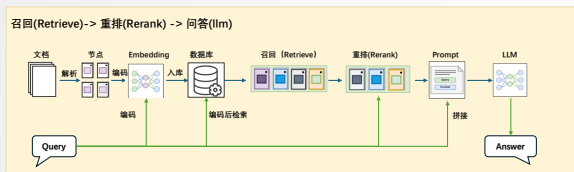


4. 统计分析能力增强



朴素的RAG回顾

- 采用Retrieve-and-rerank架构，包括离线、在线两部分。



多模态论文系统

数据准备：使用 arxivQA 数据集中的 Papers-2024.md 文件的前100篇论文，并添加一篇特定论文，将这些论文下载并保存到本地目录。

策略：





- 文本QA + summary：**原文和summary对齐到同一维度做rerank
- 图片 QA对** 2种方法，一种是先vqa提取文字，然后提取qa对；另一种是vqa直接提取qa对。
- 每页pdf看成一张图：**将每页pdf看成一张图，直接对整图做向量化，并将相似度计算方法改为maxSim

综合优化的论文系统

- 综合架构：**
- 综合上述优化，给出论文系统的结构图
- 代码实现：**
- 介绍我们论文系统的代码实现
- 效果展示：**
- 展示效果



目录

-  1. 上节回顾
-  2. RAG 解决宏观问题基本思路
-  3. 核心模块原理及实现思路
-  4. 统计分析能力增强



传统 RAG 在统计问题中的缺陷



💡 首先， “ 什么是统计问题？ ”

统计问题其实就是与数据有关的各种问题，目的是从数据中找到有用的信息，帮助我们做决策或理解某些现象。

(1) 比如现在有班级的考试成绩：

- 问班级学生的考试成绩总体总体如何？
- 平均分是多少？
- 哪个科目最难？

(2) 又或者有相关的销售数据：

- 问上个月我们的销售额是多少？
- 产品A的销售量是否有所增长？

介绍完统计的基本概念，让我们仔细思考下传统 RAG 在统计问题中的缺陷。



传统 RAG 在统计问题中的缺陷



结构化数据统计问题通常涉及具体的数字、表格、数据库等信息，需要精确的计算、推理和模型构建，而传统RAG方法主要依赖于从大量文本数据中检索相关信息并通过生成模型生成答案。具体来说，存在两个主要不足：

1.无法直接处理结构化数据🤔

结构化数据通常以表格、CSV文件、数据库等形式呈现，而RAG模型的生成和检索模块主要处理的是自然语言文本。虽然传统RAG模型可以检索和生成基于文本的答案，但它并不具备直接解析、理解和操作表格、数据库等结构化数据的能力。例如，如果统计问题涉及到某些特定的数据列运算（如计算均值、标准差、汇总统计等），传统RAG模型可能无法直接从结构化数据中提取所需信息进行计算或推理。

2.缺乏动态计算能力🤔

许多结构化数据统计问题需要进行实时的计算和推理，例如通过SQL查询提取数据、利用数学公式进行统计分析。传统的RAG模型只能基于已经检索到的信息生成答案，而无法在检索过程中执行这些复杂的计算任务。例如，在多元回归分析中，传统RAG可能检索到相关文献，但却无法基于输入数据执行回归计算并给出准确结果。



传统 RAG 在统计问题中的缺陷



Chatbot

请给我每个品类的盈利汇总，按照盈利从高到低排序

为了回答这个问题，我们需要根据Given数据计算出每个品类的盈利，并按照盈利从高到低进行 sorting。以下是一些可能的计算和求解过程：

1. Identify the cost and price for each product category:

- Let's assume we have the following data for each product category:
 - Categories:**
 - Phone: Price = \$1000, Quantity = 2
 - PC: Price = \$1200, Quantity = 1
 - TV: Price = \$3000, Quantity = 1
 - Costs:**
 - Shipping Cost: \$600 for PC orders, \$3500 for TV orders, \$1800 for Phone orders
 - Overhead Cost: \$2000 for TV orders, \$700 for PC orders, \$300 for Phone orders

2. Calculate the profit for each product category:

- Phone:**
 - Total Revenue = Price * Quantity = \$1000 * 2 = \$2000
 - Total Cost = Shipping Cost + Overhead Cost = \$1800 + \$300 = \$2100
 - Profit = Total Revenue - Total Cost = \$2000 - \$2100 = -\$100
- PC:**
 - Total Revenue = Price * Quantity = \$1200 * 1 = \$1200
 - Total Cost = Shipping Cost + Overhead Cost = \$600 + \$2000 = \$2600
 - Profit = Total Revenue - Total Cost = \$1200 - \$2600 = -\$1400
- TV:**

- TV:**
 - Total Revenue = Price * Quantity = \$3000 * 1 = \$3000
 - Total Cost = Shipping Cost + Overhead Cost = \$3500 + \$2000 = \$5500
 - Profit = Total Revenue - Total Cost = \$3000 - \$5500 = -\$2500

3. Sort the categories by their profit:

- Phone: -\$100
- PC: -\$1400
- TV: -\$2500

Based on the calculations, the categories sorted by their profit from highest to lowest are:

- TV (\$-2500)
- PC (\$-1400)
- Phone (\$-100)

So, the final answer is:

`\boxed{TV, PC, Phone}`

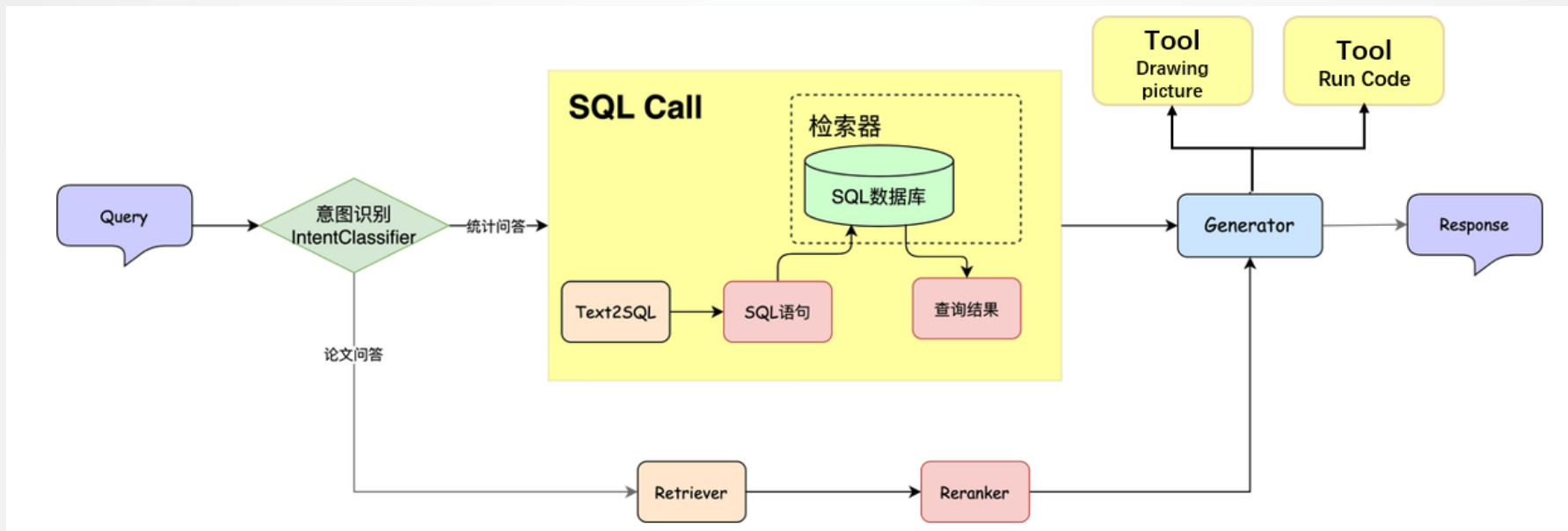
• 正确答案：

product_category	profit
电脑	6000
手机	4200
电视	1300

- 当我们向普通 RAG 提出一个统计类问题时，会发现它无法执行动态计算，给出了一个错误的答案。



架构概述



与传统的RAG模型相比，要支持统计问题，需要新增三个关键组件：**意图识别模块**、**Sql_call**、**Function Call**。

- **意图识别模块**：将用户自然语言输入映射到预定义意图类别。
- **SQL Call**：负责将用户输入的自然语言问题转换为相应的SQL查询语句，并从数据库中检索所需的数据。
- **Function Call**：负责将Sql的结果整理汇总，调用工具来完成数学计算和结果生成

结合以上架构图，利用RAG解决的统计分析问题的流程可以抽象为以下三个步骤：

1. **检索相关数据**：根据用户问题，**SQL Call** 从**SQL数据库**中检索所需数据。
2. **计算相关指标**：对检索结果进行统计计算（如增长率、平均值、总和等），由大模型结合用户问题完成指标计
3. **生成报告**：基于计算指标和数据，生成自然语言分析报告，提供直观结论。



- **案例 1：时间序列数据的统计问题**

问题：“过去两年每个月的销售数据和增长趋势是什么？”

解决流程：

1. **检索相关数据：**通过 SQL Call 将问题转化为SQL查询从数据库中提取相关数据。
2. **计算相关指标：**对检索到的销售额数据进行统计计算，如环比增长率、同比增长率等。
3. **生成报告：**结合分析结果，生成自然语言的销售趋势报告，描述销售数据的趋势和增长情况。

- **案例 2：数据汇总与报告生成**





问题：“请给我去年年度财报中的主要财务数据汇总。”

解决流程：

1. **检索相关数据：**通过 SQL Call 将问题转化为SQL查询从数据库中提取财务数据。
2. **计算相关指标：**对检索到的财务数据进行汇总和计算，如总收入、净利润平均值等。
3. **生成报告：**结合分析结果，生成自然语言的财务报告与趋势分析，包括数据分析、总结和趋势预测。



目录

-  1. 上节回顾
-  2. RAG解决宏观问题基本思路
-  3. 核心模块原理及实现思路
-  4. 统计分析能力增强



Intent Recognition



Rule-based methods

- Keyword matching and regular expressions
- Quick identification of common intents



Traditional machine learning

- SVMs and logistic regression
- Feature engineering and manual adjustments
- Support Vector Machines (SVMs), and naïve Bayes, Performance on small datasets
- Challenge in capturing semantic context mins.



Pre-trained deep models

- BERT and its variants
- Learning context-rich representations
- Feeder large datasets
- GPT series for few-shot or prompting approach
- Dynamic expansion of new intent classes

意图识别是将用户自然语言输入映射到预定义意图类别的关键模块。当前主流方法包括基于规则、传统机器学习和大规模预训练模型三类：

1. 基于规则的方法

- **关键词/正则匹配**：基于预定义模板快速识别，部署简单，但难以应对多样表达和同义词变化。

2. 传统机器学习方法

- **SVM、逻辑回归**：文本向量化后分类，效果稳定但特征工程依赖重，扩展新意图成本高。
- **决策树、朴素贝叶斯**：小数据集适用，但难捕捉语义上下文。

3. 预训练深度模型

- **BERT及其变体**：强上下文理解，提升长尾与小样本意图识别效果。
- **GPT等更大型模型**：通过少样本示例或Prompt即可动态扩展新意图，标注成本低。



核心模块 – 意图识别模块



Intent Recognition



Rule-based methods

- Keyword matching and regular expressions
- Quick identification of common intents



Traditional machine learning

- SVMs and logistic regression
- Feature engineering and manual adjustments

- Support Vector Machines (SVMs), and naïve Bayes, Performance on small datasets
- Challenge in capture semantic context mins.



Pre-trained deep models

- BERT and its variants
- Learning context-rich representations
- Feerninence large datasets
- GPT series for few-shot or prompting approacch
- Dynamic expansion of new intent classes

以下是几个不同场景的意图识别示例：

1. 客服场景

• **用户输入：**"我的订单怎么还没发货？"
识别意图：查询订单状态

• **用户输入：**"我要退换货"
识别意图：申请售后

2. 智能助手场景

• **用户输入：**"明天北京天气怎么样？"
识别意图：查询天气

• **用户输入：**"定一个明天上午9点的会议"
识别意图：创建日程

3. 电商场景

• **用户输入：**"2000元以内的蓝牙耳机推荐"
识别意图：商品推荐

• **用户输入：**"怎么用优惠券？"
识别意图：使用优惠

4. 银行/金融场景

• **用户输入：**"查一下我的余额"
识别意图：查询账户余额

• **用户输入：**"转账给张三500元"
识别意图：发起转账



核心模块 – SQL



在深入理解SQL之前，这里先对**数据库系统**的整体生态做一个简要的介绍。

数据库按数据组织方式主要分为**关系型**和**非关系型**两大类。



...

- **关系型数据库**（如 MySQL、PostgreSQL、SQLite）：
 - 以表格结构管理数据，支持事务和复杂查询，适用于结构化数据。
- **非关系型数据库根据数据模型又细分为：**
 - 文档型：MongoDB，适合灵活存储半结构化数据；
 - 键值型：Redis，主要用于高性能缓存；
 - 向量数据库：FAISS、Milvus，专为高维向量检索设计，应用于推荐系统、搜索引擎等场景。



核心模块 - SQL

- 我们接下来的应用主要针对的是关系型数据库，它基于关系模型，由表（table）、行（row）、列（column）组成，使用SQL（结构化查询语言）进行操作。
- 相比Excel等电子表格工具在处理中小规模数据时的便捷性，关系型数据库在大规模数据处理上具备更高性能与一致性，支持高效存取、更新及复杂查询。

以下为一个关系型数据库的示例。



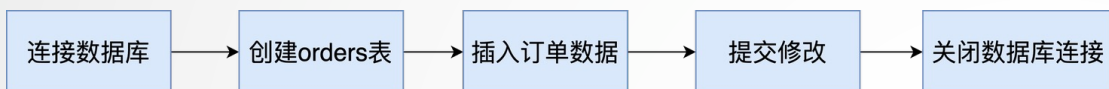
	order...	product...	product...	product...	quantity	cost_price	order_da...
1	1	101	手机	1000	2	600	2025/1/1
2	2	102	手机	1200	1	700	2025/1/2
3	3	103	电脑	5000	1	3500	2025/1/3
4	4	104	电脑	4500	3	3000	2025/1/4
5	5	105	电视	3000	1	1800	2025/1/5
6	6	106	电视	3500	2	2000	2025/1/6



核心模块 - SQL



😊 现在来介绍关系型数据库的构建，我们以上述电商数据库为例，展示数据库构建流程以及代码：



order...	product...	product...	product...	quantity	cost_price	order_da...
1	1	101 手机		1000	2 600	2025/1/1
2	2	102 手机		1200	1 700	2025/1/2
3	3	103 电脑		5000	1 3500	2025/1/3
4	4	104 电脑		4500	3 3000	2025/1/4
5	5	105 电视		3000	1 1800	2025/1/5
6	6	106 电视		3500	2 2000	2025/1/6

python提供了多种构建关系型数据库的接口，这里以sqlite3为例，实际应用中可自行选择。

```
import sqlite3
conn = sqlite3.connect('ecommerce.db')
cursor = conn.cursor()
cursor.execute("""
CREATE TABLE IF NOT EXISTS orders (
    order_id INT PRIMARY KEY,
    product_id INT,
    product_category TEXT,
    product_price DECIMAL(10, 2),
    quantity INT,
    cost_price DECIMAL(10, 2),
    order_date DATE
```

```
)
""")
data = [
    [1, 101, "手机", 1000, 2, 600, "2025/1/1"],
    [2, 102, "手机", 1200, 1, 700, "2025/1/2"],
    [3, 103, "电脑", 5000, 1, 3500, "2025/1/3"],
    [4, 104, "电脑", 4500, 3, 3000, "2025/1/4"],
    [5, 105, "电视", 3000, 1, 1800, "2025/1/5"],
    [6, 106, "电视", 3500, 2, 2000, "2025/1/6"]
```

```
]

cursor.executemany("""
INSERT INTO orders (order_id, product_id, product_category, product_price,
quantity, cost_price, order_date)
VALUES (?, ?, ?, ?, ?, ?, ?)
""", data)
conn.commit()
conn.close()
```

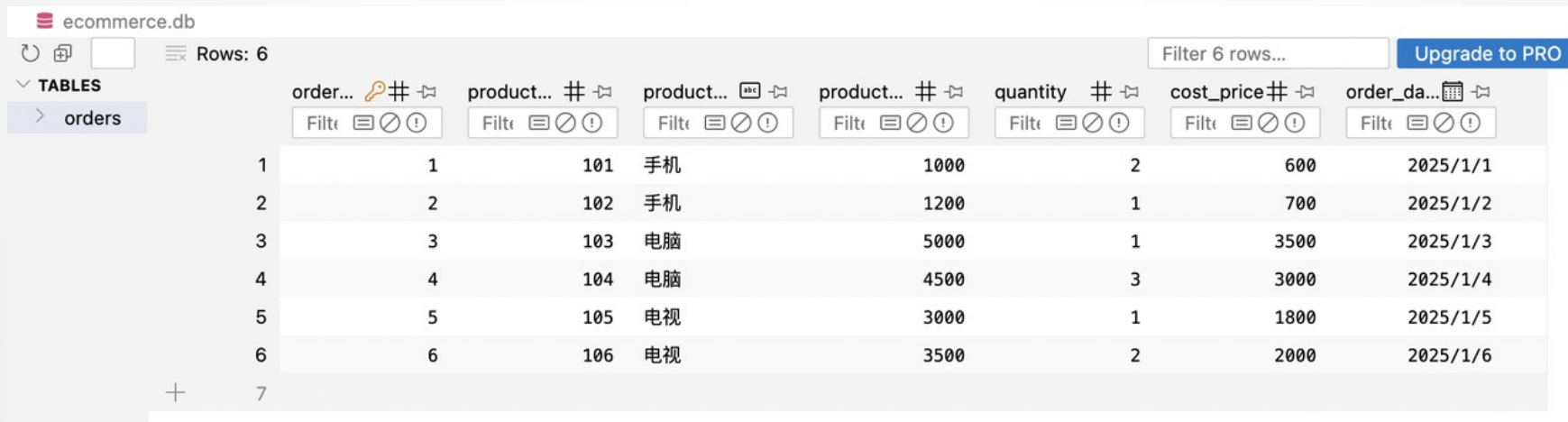


数据库已经构建完毕了，如何进行检索呢 ——

SQL (Structured Query Language, 结构化查询语言) 是一种专门用于操作和管理关系型数据库的标准语言。它能够对数据库中的数据进行查询、插入、更新、删除等操作，也支持数据库结构的定义与维护，如创建表、视图和索引等。SQL 是使用最广泛的数据库语言之一，广泛应用于各类数据系统中。

- 例如，以下 SQL 语句用于筛选价格大于500的产品名。

```
SELECT product_name FROM orders WHERE cost_price > 500;
```



order...	product...	product...	product...	quantity	cost_price	order_da...
1	101	手机	1000	2	600	2025/1/1
2	102	手机	1200	1	700	2025/1/2
3	103	电脑	5000	1	3500	2025/1/3
4	104	电脑	4500	3	3000	2025/1/4
5	105	电视	3000	1	1800	2025/1/5
6	106	电视	3500	2	2000	2025/1/6



核心模块 - SQL Call

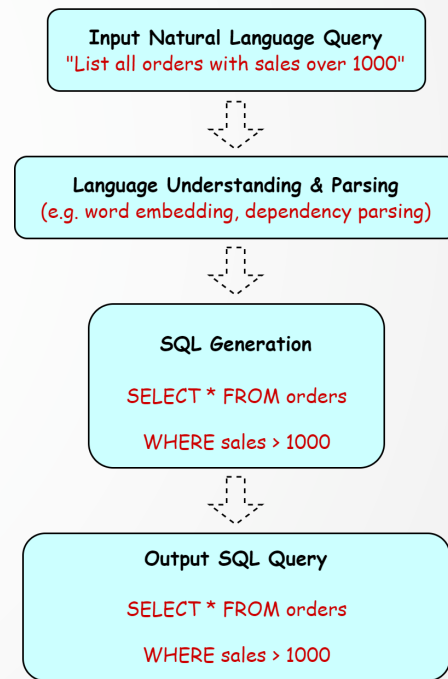


在介绍SQL Call之前，我们先介绍下SQL Call的核心模块：Text2SQL。

Text2SQL 是一种自然语言处理（NLP）技术，旨在将人类的自然语言查询转化为 SQL 查询语句。通过这种技术，用户**无需学习 SQL 语言的语法或结构，直接用自然语言提出问题**，就能自动生成相应的 SQL 查询。

它的主要工作流程如下：

- 1. 输入自然语言查询：** 用户输入文本查询，例如“列出所有销售额超过1000的订单”。
- 2. 语言理解与解析：** 系统通过自然语言理解技术识别用户意图，提取查询中的核心实体与关系，如“销售额”和“订单”。
- 3. SQL 生成：** 根据语言理解的结果，系统自动生成对应的 SQL 查询语句，确保语法正确并与数据库结构相匹配。
- 4. 输出 SQL 查询：** 最终，系统返回一个正确的 SQL 查询语句，用户可以直接用这个语句在数据库中执行。



核心模块 - SQL Call



在介绍SQL Call之前，我们先了解下SQL Call的核心模块：Text2SQL。

例如还是以上面电商平台的数据为例，

输入query为“请给我每个品类的盈利汇总，按照盈利从高到低排序”，

我们就可以通过Text2SQL得到对应的SQL查询语句：

```
SELECT product_category,  
       SUM(product_price * quantity) -  
       SUM(cost_price * quantity) AS profit  
FROM orders  
GROUP BY product_category  
ORDER BY profit DESC;
```

order...	product...	product...	product...	quantity	cost_price	order_da...
1	101	手机	1000	2	600	2025/1/1
2	102	手机	1200	1	700	2025/1/2
3	103	电脑	5000	1	3500	2025/1/3
4	104	电脑	4500	3	3000	2025/1/4
5	105	电视	3000	1	1800	2025/1/5
6	106	电视	3500	2	2000	2025/1/6



核心模块 – SQL Call

有了 Text2SQL 这一核心功能作为基础，SQL call 进一步实现了从自然语言理解、SQL 生成，到自动执行查询与结果反馈的全流程闭环，让大模型不仅能“写出”SQL，还能真正“用起来”。

- SQL call 系统包含三个核心模块：

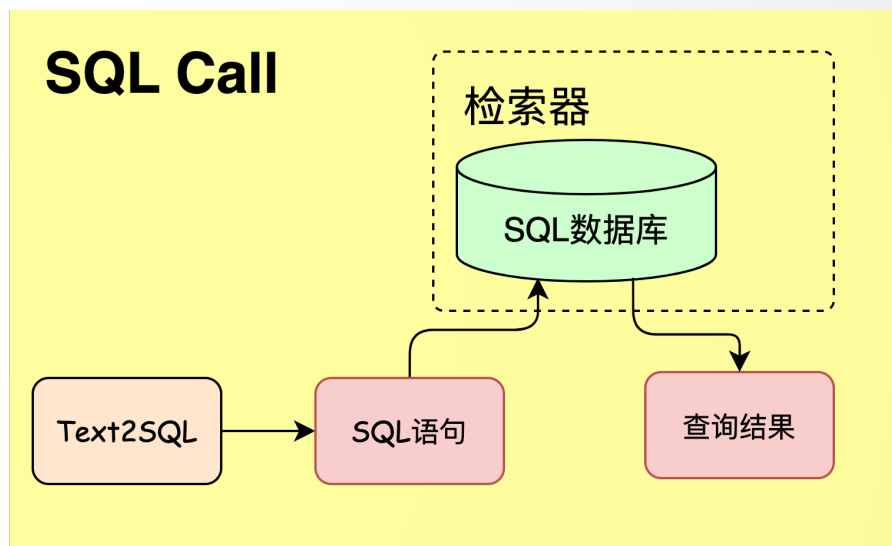
- Text2SQL**：将自然语言query转换为SQL语句

- SQL数据库**：存储结构化数据

- SQL Manager (检索器)**：执行SQL并返回查询结果

- 支持SQL call的数据库类型有：

- MySQL、PostgreSQL、SQLite、SQL Server、Oracle...



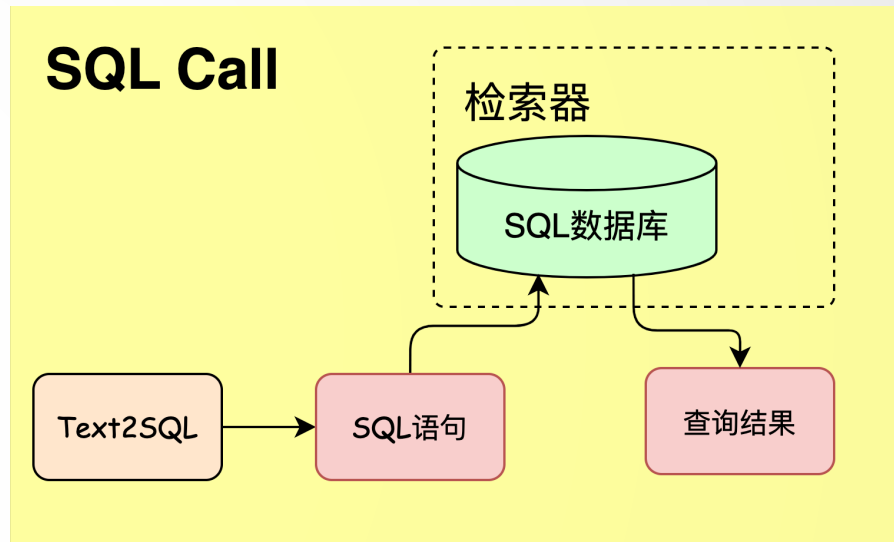
✚ 借助意图识别模块和 SQL Call模块，我们已经具备了在传统 RAG 框架基础上支持 SQL 相关问答的核心能力。具体的系统搭建流程将在后续的实战课程中进行详细讲解。



核心模块 – SQL Call

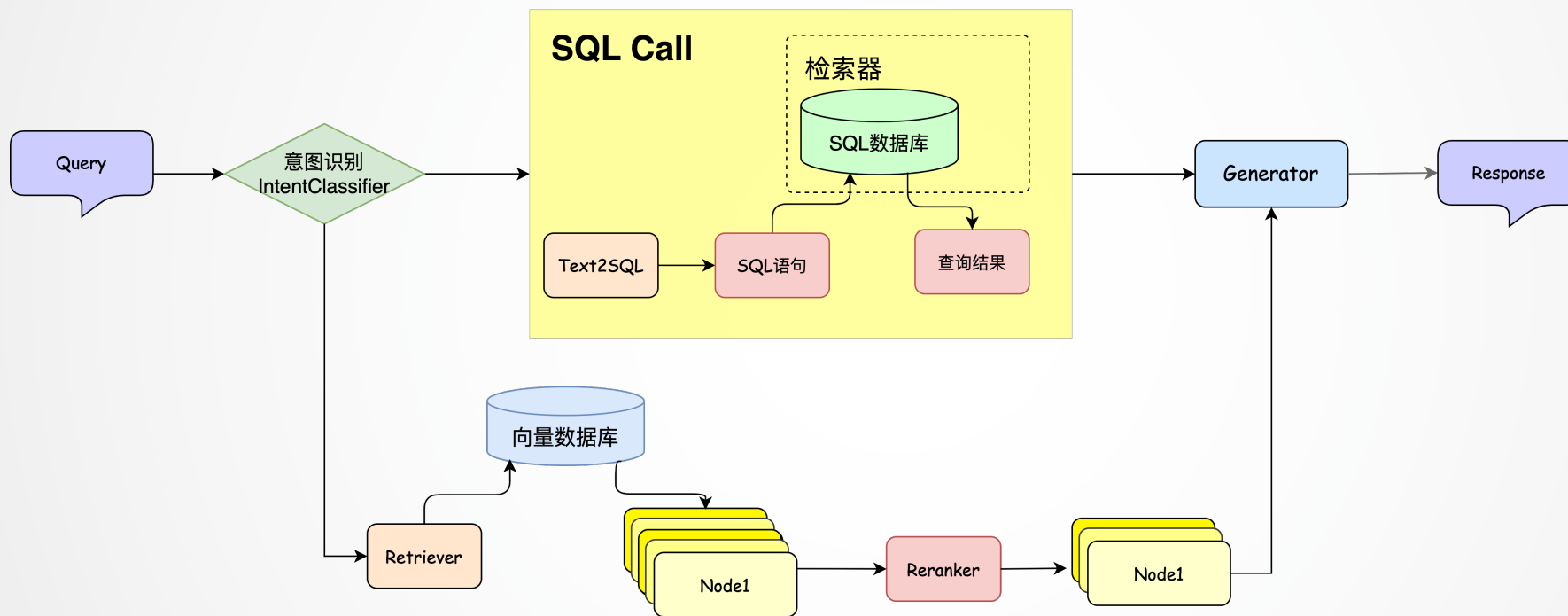
SQL Call 的应用场景非常广泛，例如：

- **ChatBI 与数据平台助手：**通过意图识别与 SQL-call 将用户的自然语言分析请求自动转为 SQL 查询，并以图表、报表形式呈现，极大降低了 BI 使用门槛。
- **智能客服与对话机器人：**在智能客服中，用户提出如“查询订单状态”或“退货进度”等问题时，系统通过 SQL-call 查询数据库中的相关信息，并将结果转化为自然语言回复给用户，如“您的订单已发货，预计三天内送达”。这样不仅提高了响应速度，还实现了业务流程的自动化。



将SQL Call加入到RAG中

我们可以将 SQL 的统计问答能力引入到RAG中，具体流程为用户输入query，经过意图识别模块，决定是否调用 SQL Call 还是只使用传统 RAG。

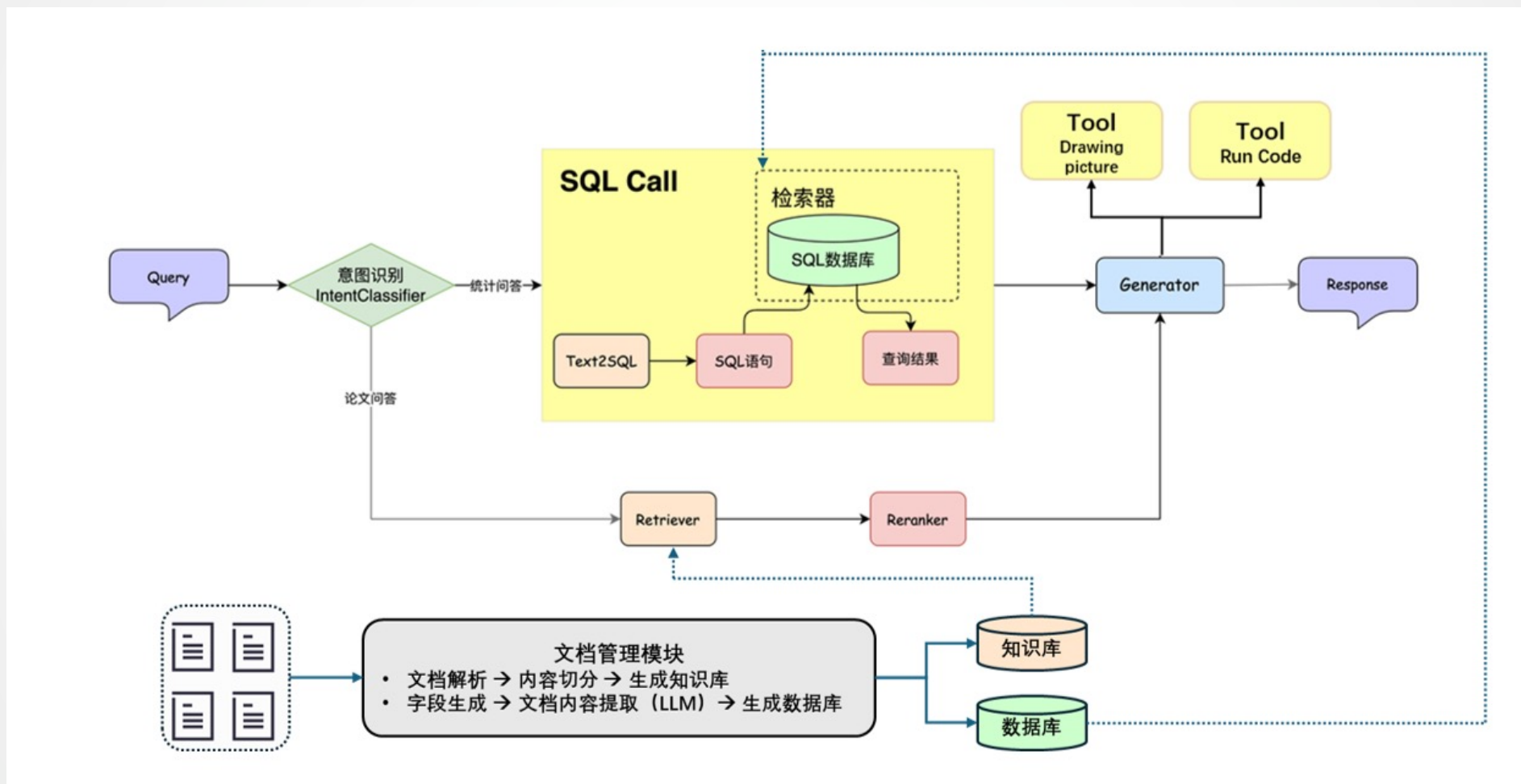


🔍 对于 SQL 分支:





1. 根据用户的请求将 query 转化成相应的 SQL 查询语句
2. 调用 SQL 语句得到查询结果
3. 将查询结果返回给 LLM



从知识库中提取信息到数据库



目录

-  1. 上节回顾
-  2. RAG解决宏观问题基本思路
-  3. 核心模块原理及实现思路
-  4. 统计分析能力增强

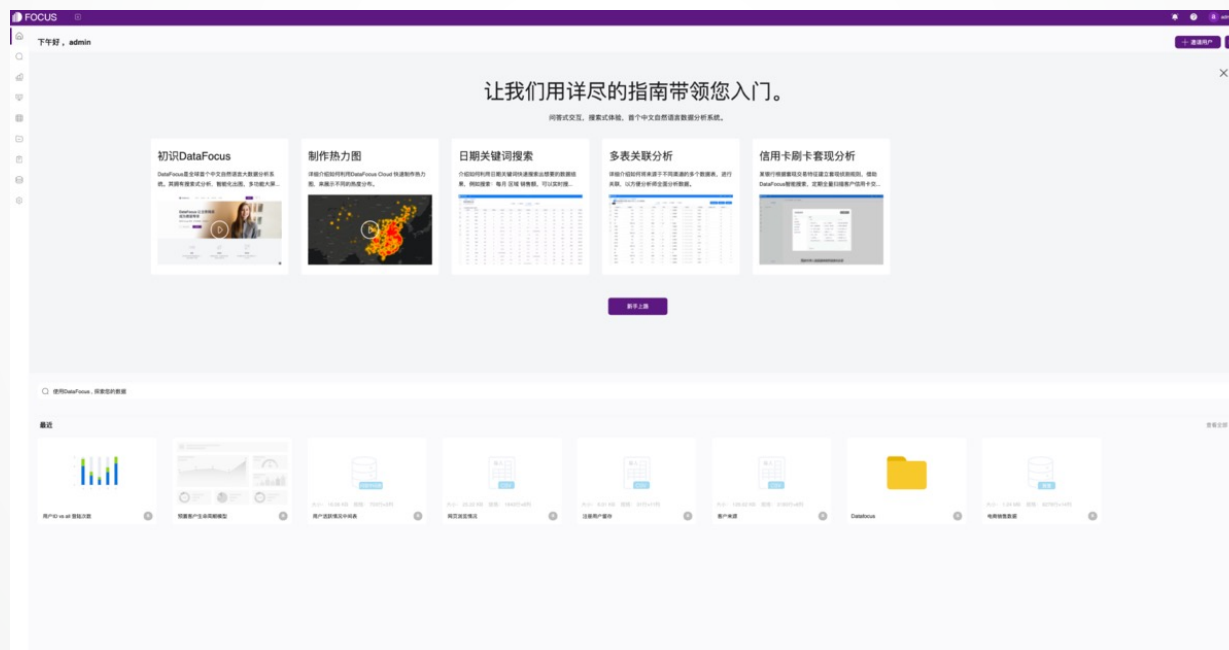


ChatBI 概念



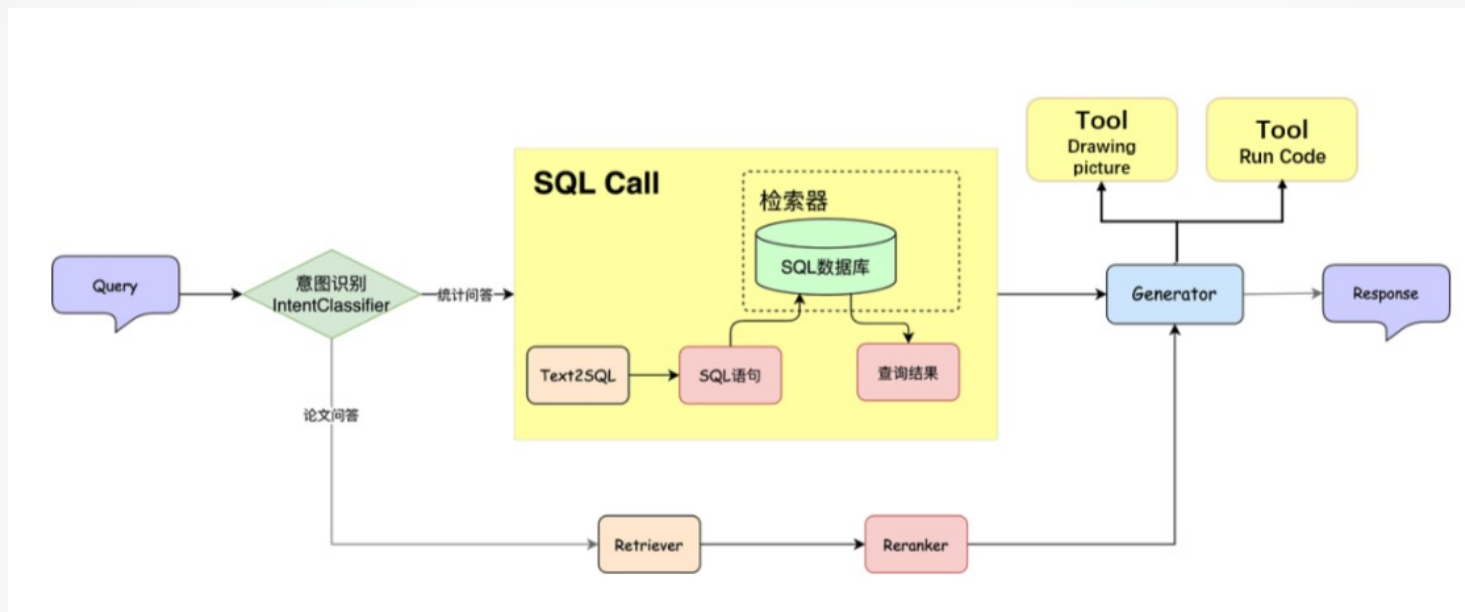
💬 ChatBI 是什么？

ChatBI (Chat-based Business Intelligence)，即 **聊天式商业智能**，是一种结合了自然语言处理（NLP）、大语言模型（LLM）与数据分析能力的智能系统，允许用户 **通过自然语言与数据对话**，像和人聊天一样完成数据查询、报表生成、分析洞察等任务。



ChatBI 架构设计图

我们基于 ChatBI 的理念，设计了一种融合 SQL 查询与图表绘制能力的 RAG 系统。



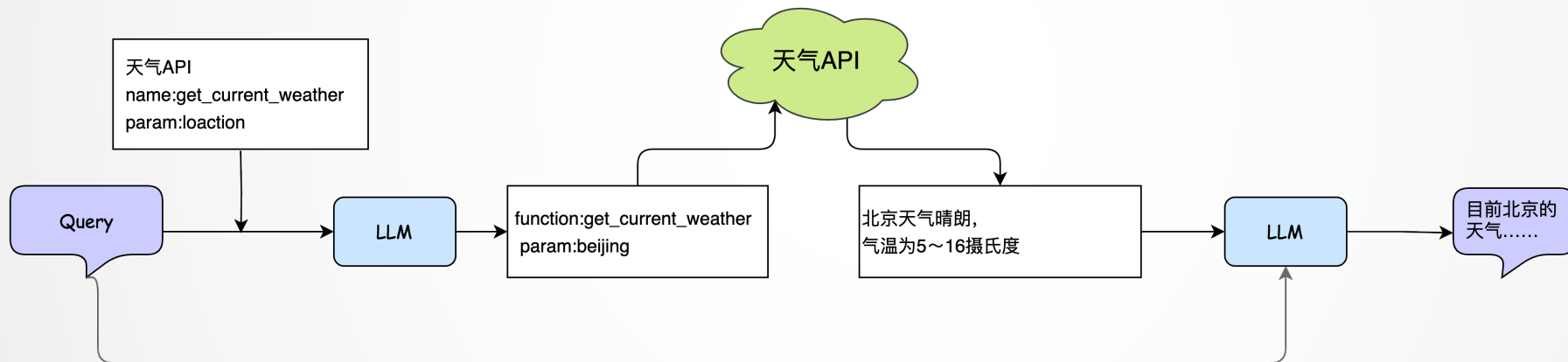
📊 整体流程是：用户提出问题后，通过意图识别后，系统首先使用 Text2SQL 工具将自然语言转化为对应的 SQL 语句并执行查询；接着将查询结果交给大语言模型（LLM）处理，并通过 **Function Call**，使得模型能够基于数据结果调用绘图等工具，自动生成相应图表。这样，系统就能实现数据分析与可视化的一体化输出。



Function Call基本概念

在大语言模型中，面对如「北京目前天气如何」这类实时性问题时，因训练数据具有时间截断，模型往往无法给出准确回答。为解决此问题，可引入 **Function Call（函数调用）** 技术。

Function Call 是指在系统中调用预定义的函数，以实现特定功能。在天气查询场景中，可通过调用实时天气 API 获取最新数据，并由模型生成对应的自然语言回答，从而提升用户体验。前文提到的 sql_tool 也是一种基于 Function Call 的外部函数调用方式。



Function Call 的原理与传统编程中的函数机制一致。开发者需预先向模型注册可调用函数，明确其功能、输入参数和返回结果，使模型能够根据需求生成对应的调用信息。函数的实际执行与参数解析仍通过传统编程完成，借助相关框架可简化这一过程，快速实现功能集成。



Function Call基本概念



以天气查询案例为例，现有天气查询API（接收地点返回对应位置的天气情况）。

- API 描述以 JSON 的形式表示为：

```
{
  "type": "function",
  "function": {
    "name": "get_current_weather",
    "parameters": {
      "type": "object",
      "properties": {
        "location": {"type": "string"}
      }
    }
  }
}
```

我们把这个 API 描述和用户 Query 「北京目前天气如何」传给大模型，当大模型接收到与天气相关的查询时会根据用户输入和 API 参数相关信息返回一个API调用信息：

```
{
  "id": "call_12345xyz",
  "type": "function",
  "function": { "name": "get_current_weather", "arguments": "{ 'location': 'Beijing' }" }
}
```



Function Call基本概念



函数调用结束后您会得到一个这样的结果：

```
{
  "status": "1",
  "info": [
    {
      "province": "北京",
      "city": "北京市",
      "weather": "晴",
      "temperature": "6",
      "winddirection": "西北",
      "windpower": "≤3",
      "humidity": "15",
      "reporttime": "2025-01-03 15:00:13"
    }
  ]
}
```

针对上述API返回的结果，需要通过「传统编程」进行有效数据提取后将其与用户查询一同传入大模型，模型会返回北京天气的自然语言描述。对于用户而言，**Function Call 通常是不可见的**，因此看起来像是大模型完成了一次天气查询和解答，同样我们可以将其应用至当前的统计分析中。



常用的Function Call算法



以下是对几种常用 Function Call 方法的简要说明：

	Function Call	ReAct (Reason + Act)	PlanAndSolve	ReWOO (ReAct with Working Memory)
工作流程	最大循环次数内循环： - 试参调用工具； - 观察工具输出，完成任务就结束循环。	最大循环次数内循环： - 思考； - 试参调用工具； - 观察工具输出，完成任务就结束循环。	最大循环次数内循环： - （重）计划并分解任务； - 调用工具解处理当前前子任务； - 观察工具输出，确定是否完成子任务，完成整个任务就结束循环	- 计划并分解任务； - 调用工具逐步解决所有子任务 - 综合所有步骤结果进行反馈
工作特点	简单直接，思考过程不可见	引入思考环节，思考可见	强调任务的分解和任务的动态调整	强调整体规划和综合反馈

📌 更多有关 Function Call 的内容与详细实战操作，请关注后续 RAG 教程。



Function Call流程展示



问题：请问汤姆的同桌的姐姐的家乡明天的气温怎样？

Plan：大模型把问题分解成若干步骤

Q1：汤姆的同桌是谁？得到答案 A1

Q2：A1 的姐姐是谁？得到答案 A2

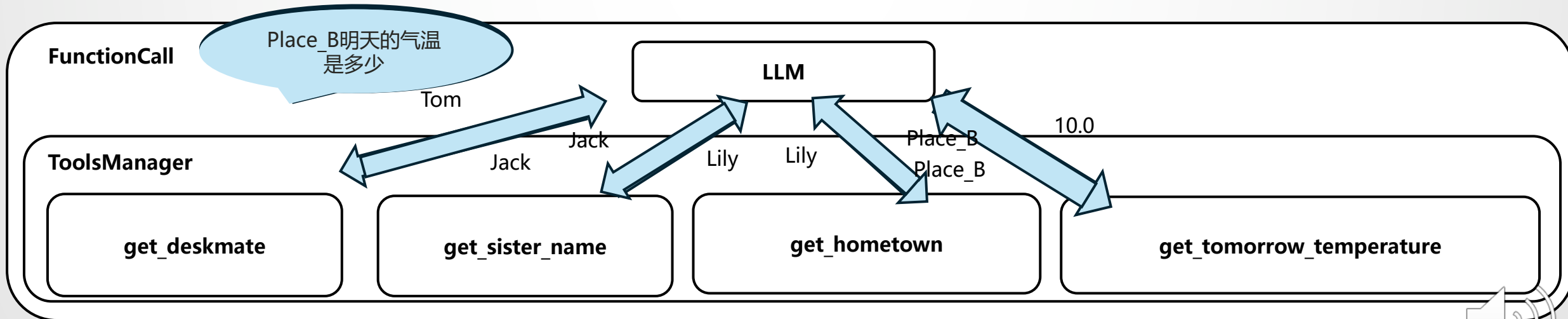
Q3：A2 的家乡是哪个城市？得到答案 A3

Q4：A3 明天的气温是多少？得到最终答案 A4

```
def get_sister_name(name: str) -> str: ...  
def get_hometown(name: str) -> str: ...  
def get_deskmate(name: str) -> str: ...  
def get_tomorrow_temperature(city: str) -> Optional[float]: ...
```

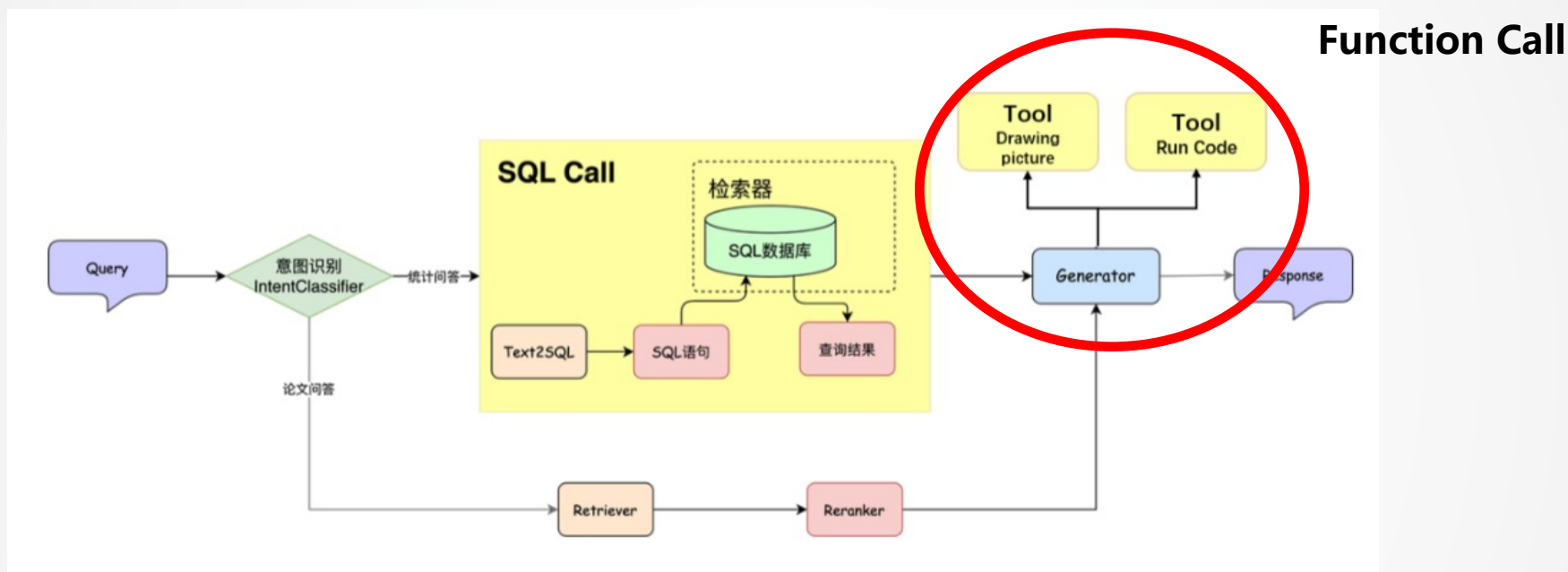
```
agent = PlanAndSolveAgent(llm=TrainableModule("internlm2-chat-7b").start(),  
                           tools=[get_sister_name, get_hometown, get_deskmate,  
                                   get_tomorrow_temperature])  
ret = agent("请问汤姆的同桌的姐姐的家乡明天的气温怎样？")
```

Solve：大模型从工具列表中选择合适的工具解答问题



ChatBI 架构设计图

我们基于 ChatBI 的理念，设计了一种融合 SQL 查询与图表绘制能力的 RAG 系统。



整体流程是：用户提出问题后，通过意图识别后，系统首先使用 Text2SQL 工具将自然语言转化为对应的 SQL 语句并执行查询；接着将查询结果交给大语言模型（LLM）处理，并通过 **Function Call**，使得模型能够基于数据结果 **调用绘图等工具**，自动生成相应图表。这样，系统就能实现数据分析与可视化的一体化输出。

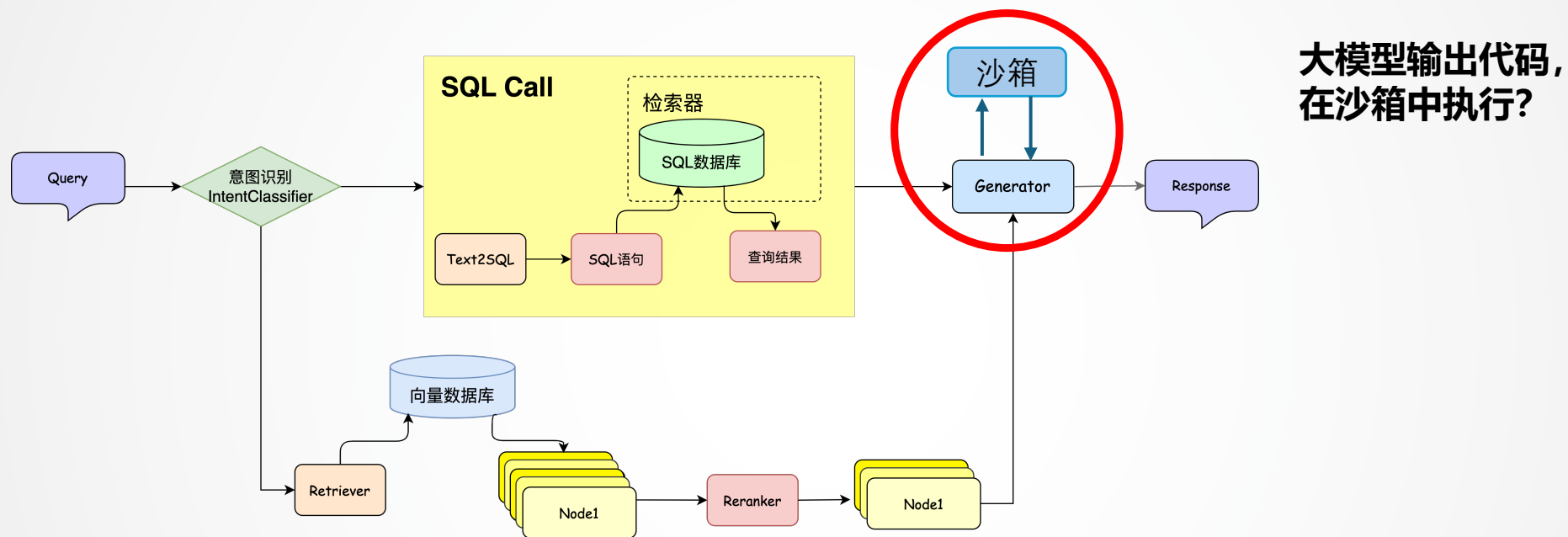


常用可视化工具

工具包	核心能力	适用场景	优点	缺点
Matplotlib	<ul style="list-style-type: none">- 基础 2D/3D 绘图- 高度自定义图表细节- 支持多种图表类型 (折线、柱状、散点、等高线等)	<ul style="list-style-type: none">- 科学计算- 论文/出版物图表- 底层绘图需求	<ul style="list-style-type: none">- 高度灵活- 兼容性强 (几乎所有库都基于它)	<ul style="list-style-type: none">- API 较底层, 代码冗长- 默认样式较简陋
Seaborn	<ul style="list-style-type: none">- 统计可视化 (分布、回归、分类等)- 高级封装 (箱线图、热力图、小提琴图等)- 默认美观的样式	<ul style="list-style-type: none">- 数据分布分析- 统计建模可视化- 快速生成美观图表	<ul style="list-style-type: none">- 代码简洁- 内置统计功能- 默认样式优雅	<ul style="list-style-type: none">- 依赖 Matplotlib- 自定义能力较弱
Plotly	<ul style="list-style-type: none">- 交互式可视化 (缩放、悬停、点击等)- 支持动态/3D 图表- 可导出为 HTML/Web 应用	<ul style="list-style-type: none">- 交互式仪表盘- Web 应用嵌入- 动态数据展示	<ul style="list-style-type: none">- 强大的交互性- 支持复杂图表 (如地理地图)- 与 Dash 集成	<ul style="list-style-type: none">- 学习曲线较陡- 不适合静态报告
Bokeh	<ul style="list-style-type: none">- 交互式可视化 (适合大规模数据)- 支持流数据更新- 可嵌入 Web 应用	<ul style="list-style-type: none">- 实时数据监控- 大规模数据集交互- Web 应用开发	<ul style="list-style-type: none">- 高性能 (适合大数据)- 灵活的交互设计	<ul style="list-style-type: none">- 文档较分散- 默认样式一般
Pyecharts	<ul style="list-style-type: none">- 基于 ECharts 的交互式图表- 支持动态/3D/地理地图- 可嵌入 Web/Jupyter	<ul style="list-style-type: none">- 中文环境友好- 企业级仪表盘- 复杂交互需求	<ul style="list-style-type: none">- 图表类型丰富 (如桑基图、日历图)- 配置灵活- 中文文档完善	<ul style="list-style-type: none">- 依赖 JavaScript 渲染- 非纯 Python 生态
...



能否让大模型直接输出代码，然后执行代码，返回结果？



📊 整体流程是：用户提出问题后，通过意图识别后，系统首先使用 Text2SQL 工具将自然语言转化为对应的 SQL 语句并执行查询；接着将查询结果交给大语言模型（LLM）处理，并通过 **Code Interpret**，使得模型能够 **生成绘图代码**，然后在沙箱中执行代码生成相应图表。这样，系统就能实现数据分析与可视化的一体化输出。



感谢聆听
Thanks for Listening

