

Langchain 大模型开发入门课

Kai

X: @real_kai42

<https://kaiyi.cool>

Hi there 🙌



从前端到 AI: LangChain.js 入门和实战

手把手带你开发 AI 应用，提升大模型实战能力

👤 Kai42 💡 LV.2 🏆 @某知名外企, Github 15k+ stars

已购买 已完结 24 小节 · 1,471人已购买

- 🧑‍💻 Kai
- 某外企 - AI 部门
- Github 获得 16 k star ⭐
- 大模型入门课 销量 1.5k
- <https://kaiyi.cool>



课程目标

- Langchain.js 的基础入门
- 基础 RAG 实现
- reAct Agent 实现 —— 理解 Agent 原理
- OpenAI Agent 实现 —— Agent 的最佳实现

For Your Information

- 没有 Live Coding
课程最后会附完整代码开源仓库
- 不是 Langchain.js 的基础教程
聚焦在如何用 Langchain.js 实现大模型落地,
入门知识官方文档更合适
- 核心在大模型应用的实现逻辑
聚焦在 RAG 和 Agent 背后的大模型应用思想, 而不仅仅是代码

Langchain 基础知识

应用

定位是一个大模型的应用框架，处于应用和模型之间的生态位

Langchain

提供了一系列的内置工具和调试工具，方便迅速落地大模型应用

大模型

提供针对不同模型、向量数据库、搜索工具等 第三方工具的集成

适合迅速测试和落地 idea，多种服务的胶水层

LCEL First

LangChain Expression Language (LCEL) 是 Langchain 提出的用于构建链式调用的语法。是经历重构后 langchain 唯一推荐的使用方式

LCEL 描述的就是数据在 Chain 中，从一个块（Runnable）流向另一个块的过程

一个 Chain 由多个块组成，Chain 又可以作为另一个 Chain 的块来复用

基础 Model 调用



```
1 async function basicModel() {  
2   // 直接调用模型  
3   const chatModel = new ChatOpenAI();  
4   // chat model 的参数要求是一组 message  
5   const res = await chatModel.invoke([  
6     new SystemMessage("你是一个非常有趣的 AI, 会用非常严肃的口吻讲  
笑话"),  
7     new HumanMessage("讲个笑话"),  
8   ]);  
9  
10  console.log(res);  
11 }
```

- 创建 `chatModel` 实例
- 构建一组对话 (`message`)
- 调用并返回结果



```
1
2 AIMessage {
3     "id": "chatcmpl-A3g9XsSHWEOKz4ehmdms9i0Qi1mRZ",
4     "content": "一位虔诚的教徒，一天站在教堂外，抬头看天，喃喃自语道：  
“主啊，请给我一个中奖的彩票号码，我实在是太需要这个奇迹了。”\n\n他每日  
祈祷，坚持不懈。数周过去了，仍无所获。一次祈祷时，他忽然听到天上传来深  
沉的声音：“孩子，我愿意帮助你，但请你也务必帮帮我—至少，去买张彩  
票。”",
5     "additional_kwargs": {},
6     "response_metadata": {
7         "tokenUsage": {
8             "completionTokens": 113,
9             "promptTokens": 34,
10            "totalTokens": 147
11        },
12        "finish_reason": "stop",
13        "system_fingerprint": "fp_80a1bad4c7"
14    },
15    "tool_calls": [],
16    "invalid_tool_calls": [],
17    "usage_metadata": {
18        "input_tokens": 34,
19        "output_tokens": 113,
20        "total_tokens": 147
21    }
22 }
```

- 原始的 LLM 输出，包含大量的附加信息和 meta data

构建基础的 Chain



```
1 async function basicChain() {  
2   const chatModel = new ChatOpenAI();  
3   // 定义 outputParse, 其作用是解析 AI 的输出, 将其中的  
   // 文本部分提取出来  
4   const outputPrase = new StringOutputParser();  
5  
6   // 通过 pipe 连接成 chain  
7   const simpleChain = chatModel.pipe(outputPrase);  
8  
9   const res = await simpleChain.invoke([new  
  HumanMessage("Tell me a joke")]);  
10  console.log(res);  
11 }
```

- 通过 Pipe 语法将两个块连接起来
- 输入先进入 Chat Model, 获得原始的 LLM 输出
- 流入 Chain 中的下一个块 StringOutputParser, 提取出其中的文本信息
- 输出给用户

构建基础的 Chat Chain



```
1 // 构建 chatPrompt
2 const prompt = ChatPromptTemplate.fromMessages([
3     // fromMessages 中可以使用的简化的语法。 tone 是变量
4     ["system", "你是一个非常有趣的 AI, 会用非常{tone}的口吻讲笑话"],
5     ["human", "讲个关于{topic}的笑话"],
6 ]);
```

- 定义一个 Prompt 模板，其中变量用 花括号表示
- 变量可以从上一个块的输出中获得，并填充到对应的位置
- 等价于字符串模板



```
1 const chatModel = new ChatOpenAI({  
2   // 设置为 true 时, 会输出更多的信息, 方便调试  
3   verbose: true,  
4 });  
5 // 定义 outputParse, 其作用是解析 AI 的输出, 将其中的文本部分提取出来  
6 const outputPrase = new StringOutputParser();
```

- 创建 Chat Model 实例
- 创建 String Output Parser 实例



```
2 const chatChain = RunnableSequence.from([prompt, chatModel,  
outputPrase]);  
3  
4 const res = await chatChain.invoke({ topic: "猫", tone: "二次元" });  
5 console.log(res);
```

- 通过 RunnableSequence.from 将所有 块 链接成 Chain, 等价于 pipe
- 调用 Chain, 这里输入 Chain 需要的变量 topic 和 tone



你已经学会了国棋的基本规则，接下来请
你迎战九段围棋高手柯洁

基础 RAG 实现

RAG

RAG (检索增强生成, Retrieval-Augmented Generation)

根据用户的提问 (Query) , 从向量数据库中检索相关的信息 (Retrieval) , 将检索到的信息和原始提问提供给 LLM (Augmented) , 最后生成更高质量的回复 (Generation)

简单理解就是，给 LLM 外挂了一个数据库，可以是公司内部文档、互联网搜索结果、实时的数据和信息等

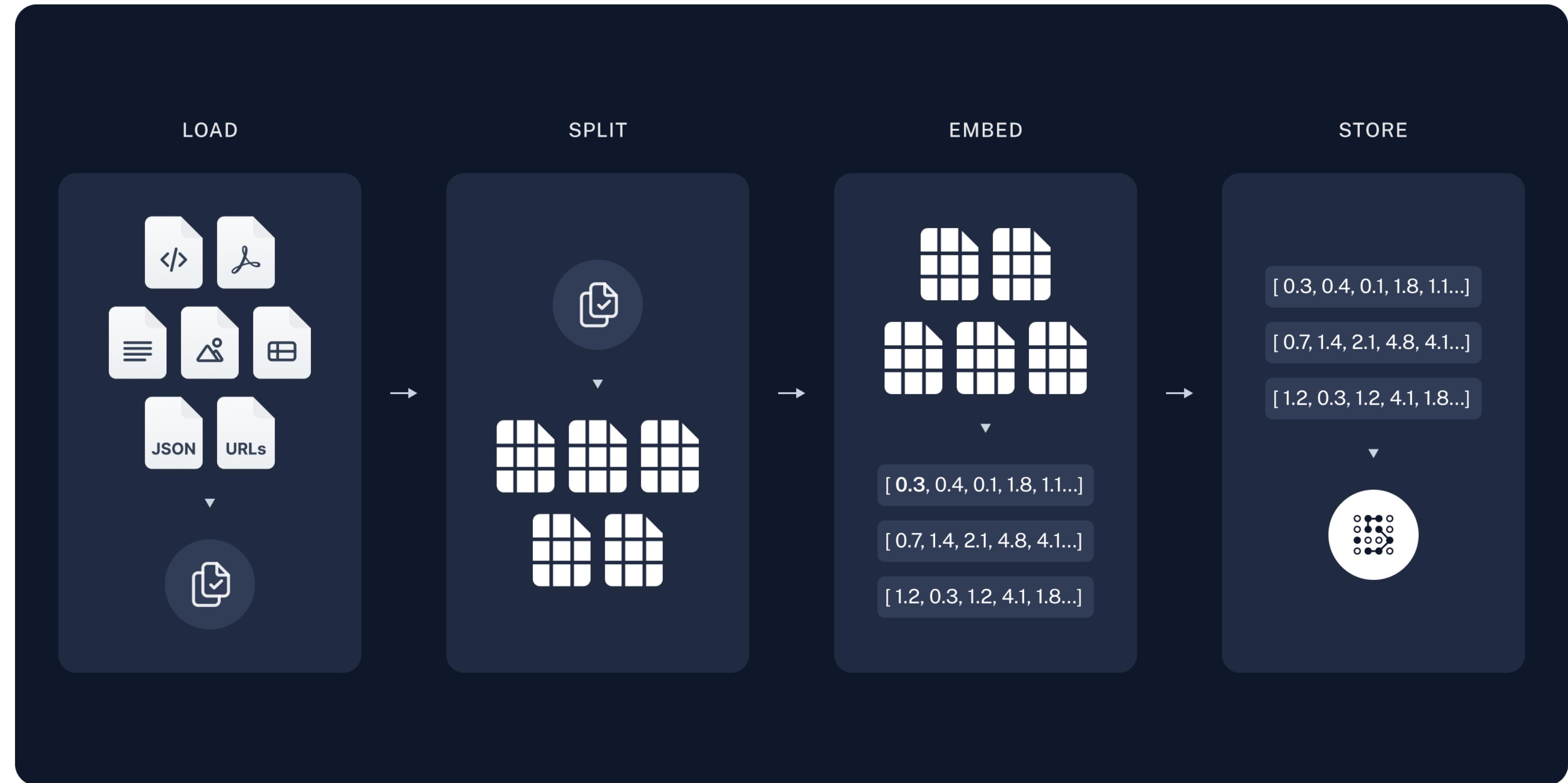
依靠高质量的实时数据，减少大模型的幻想问题

RAG - 切割、向量化、存储

切割：大模型上下文有限，我们需要将原始数据切割成合理的小块，方便检索和嵌入到大模型的上下文中

向量化：也叫嵌入（embedding），将切割后的文档向量化成向量，方便检索。不同 embedding model 的向量化方式不一样，要保证向量化文档的模型和向量用户查询的模型是一个模型，才有检索的意义

存储：将文档和文档对应的向量存储到向量数据库中，方便后续进行检索



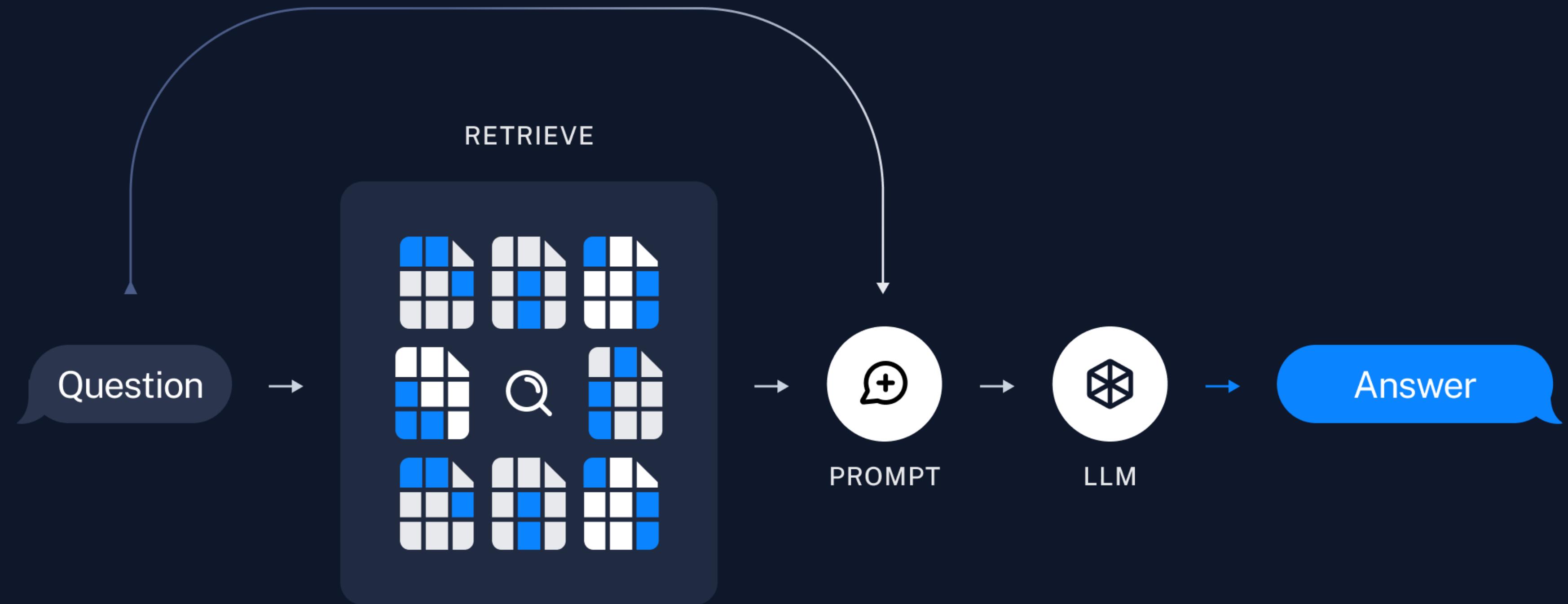
RAG - 查询、检索、增强、生成

查询：使用与向量化文档一样的嵌入模型（embedding model），将查询向量化成向量

检索：使用查询向量，对向量数据库进行检索，找到相关度高的文档

增强：将查询到的文档插入到 LLM 的 prompt 中

生成：LLM 生成回答



RAG - 代码实现

- 嵌入模型使用的是 OpenAI 的 text-embedding-3-large，更换嵌入模型的成本非常高，选择后不要更换
- 向量数据库使用的是 lanceDB，有 js、py、rust 三种语言的接口，支持多模态
- 为了方便大家理解背后的原理，我们比 langchain.js 官方 demo 实现的更加深入细节。大家可以分享后，阅读官方 demo。



```
1 const schema = new arrow.Schema([
2     // index 标记是文章的第几块
3     new arrow.Field("index", new arrow.Int32()),
4     // vector 是文章的 embedding 后的 vector
5     new arrow.Field(
6         "vector",
7         new arrow.FixedSizeList(
8             EMBEDDING_VECTOR_SIZE,
9             new arrow.Field("item", new arrow.Float32(), true)
10        )
11    ),
12    new arrow.Field("lines_from", new arrow.Int32()),
13    new arrow.Field("lines_to", new arrow.Int32()),
14    // content 该块的内容
15    new arrow.Field("content", new arrow.Utf8()),
16]);
17
18 const empty_tbl = await db.createEmptyTable(TABLE_NAME, schema);
```

创建向量数据库 (lanceDB) 中的表，其中 vector 用来储存向量，这里要设置向量的 size，去对应 embedding model 的文档中查阅。

content 存储原始文档，其他的字段存储相关的 metadata，可以根据业务需要去设置



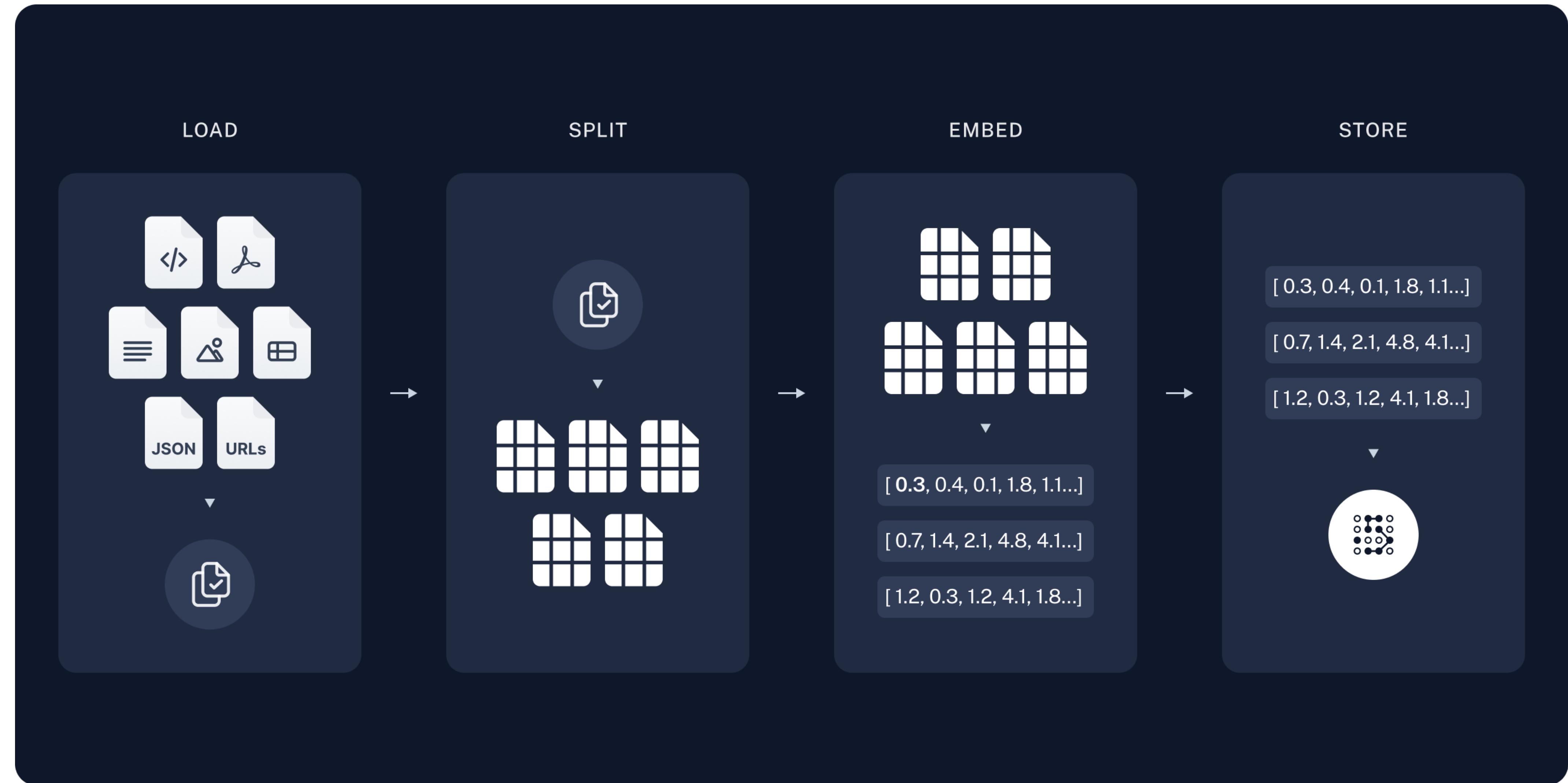
```
1 // 加载数据源
2 const loader = new TextLoader(path.join(currentDir, "../data/kong.txt"));
3 const docs = await loader.load();
4
5 // 对长文章进行切块
6 const splitter = new RecursiveCharacterTextSplitter({
7   // 每块的长度
8   chunkSize: 500,
9   // 块之间的重叠长度
10  chunkOverlap: 100,
11 })
```

- 加载文档，这里使用 Langchain.js 提供的 TextLoader 去加载 txt 文件
- 对原始文章进行切块，切成 500 字符一块，块之间有 100 字符的重叠，可以根据业务情况进行设置



```
1 const splitDocs = await splitter.splitDocuments(docs);
2
3 // 将切块后的 content 进行 embedding
4 for (let i = 0; i < splitDocs.length; i++) {
5   const doc = splitDocs[i];
6
7   const vector = await embeddings.embedQuery(doc.pageContent);
8   await table.add([
9     {
10       index: i,
11       vector: vector,
12       lines_from: doc.metadata?.loc?.lines?.from,
13       lines_to: doc.metadata?.loc?.lines?.to,
14       content: doc.pageContent,
15     },
16   ]);
17 }
```

对切割后的每块文档进行向量化，并存储到向量数据库中，也会存储一些 meta data



RAG - 检索实现



```
1 const contextRetrieverChain = RunnableSequence.from([
2   (input) => input.question,
3   async (question) => {
4     const queryVector = await embeddings.embedQuery(question);
5     const relatedDocs = await
6       table.vectorSearch(queryVector).limit(4).toArray();
7     return relatedDocs.map((doc) => doc.content).join("\n ----- \n");
8   },
9 ]);
```

- 创建一个用于 retriever 的 chain，使用 RunnableSequence.from 构建
- 第一个节点从输入中提取出 question
- 第二个节点是一个函数，函数也是合法的 Runnable 块，先把用户的 question 向量化，再从向量数据库（table 实例）中进行相似性检索，并返回纯文本结果

```
2 const prompt = ChatPromptTemplate.fromMessages([
3   [
4     "system",
5     `你是一个熟读鲁迅的《孔乙己》的终极原著党，精通根据作品原文详细解释和回答问题，你在回答时会引用作品原文。
6 并且回答时仅根据原文，尽可能回答用户问题，如果原文中没有相关内容，你可以回答“原文中没有相关内容”，
7   ],
8   [
9     "user",
10    `

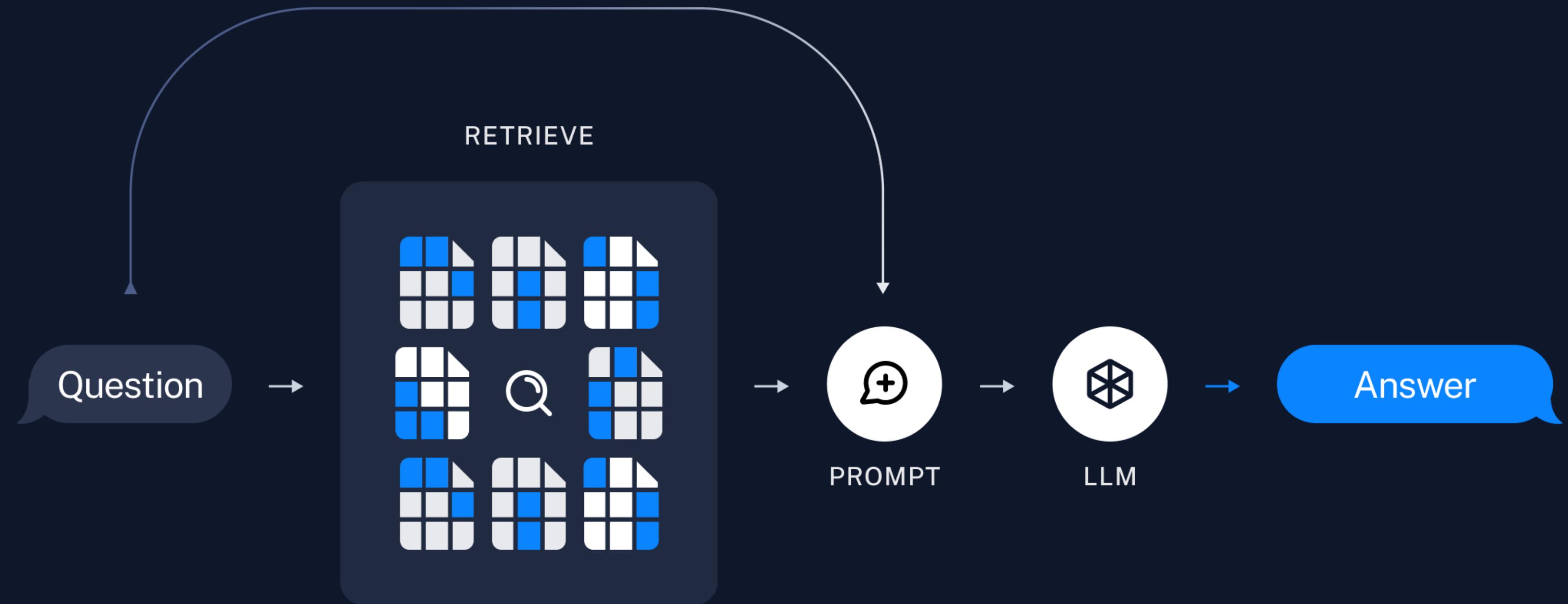
11 以下是原文中跟用户回答相关的内容：
12 {context}
13
14 现在，你需要基于原文，回答以下问题：
15 {question}
16 `,
17 ],
18 ]);
```

- 构建 prompt, system prompt 给 LLM 定义角色扮演和核心指令
- user prompt 中，用户的输入是 question 变量，检索的上下文是 context 指令。运行中由上游块提供



```
1 const ragChain = RunnableSequence.from([
2   // chain 中的第一个节点
3   {
4     // 根据用户的输入获取 context。 输入是整个 chain 的输出，返回值赋值给 context
5     context: contextRetrieverChain,
6     // 将输入中的 question 赋值给 question，也就是将用户的输入传递给下一个节点
7     question: (input) => input.question,
8   },
9   // 这个节点的输入是一个对象，包含了 context 和 question
10  prompt,
11  model,
12  // 提取模型中的文本输出
13  new StringOutputParser(),
14]);
15
```

- 最后，将所有块链接成一个 chain
- 第一个块，调用 contextRetrieverChain，并将结果赋值给 context。提取出用户输入的 question，赋值给 question（相当于透传）
- 第二个块，调用 prompt template，根据用户的输入的变量渲染成 prompt
- 第三个块，调用 LLM
- 第四个块，提取 LLM 输出中的文本，输出



reAct Agent 实现

理解 Agent

- Agent 本质上是一个能够自主感知环境、做出决策 并 执行行动的智能系统。

做出决策 => 决定调用哪些 tool (函数)

执行行动 => 传入参数并调用函数

感知环境 => 观察函数调用结果

- reAct (Reasoning and Action) 是非常早期的 Agent 系统，在 2022 年被提出
- reAct 能力并不强，仅支持单个参数和单个返回值的函数
但其非常适合理解 Agent 的本质，和观察 AI 的思考过程

回答以下问题，尽你所能。你可以使用以下工具：

{tools}

请使用以下格式：

Question: 你需要回答的输入问题

Thought: 你应该始终思考该做什么

Action: 你要采取的行动，应该是 [{tool_names}] 中的一个

Action Input: 行动所需的输入

Observation: 行动的结果

..... (此“Thought/Action/Action Input/Observation”循环可以重复 N 次)

Thought: 我现在知道最终答案了

Final Answer:: 对最初输入问题的最终回答

开始吧！

Question: {input}

Thought: {agent_scratchpad}

- tools 中会传入人类提供的工具和介绍
- 核心是引导 AI 一步步，思考 => 决策要使用的工具 => 调用工具 => 传入参数 => 观察结果，如此循环，直到最后得到结果
- 其中 agent_scratchpad 会记录整个思考过程，帮助 AI 迭代思考
- Agent 的能力范围取决于 模型的基础能力为我们提供的工具的能力

代码实现



```
1 // 创建提供给 agent 的工具。 reAct 只支持 DynamicTool, 也就是只有一个输入的工具
2 const exchangeRateTool = new DynamicTool({
3   name: "exchange-rate-calculator",
4   description: "获取人民币和美元之间的最新汇率。输入应为你当前的货币, 'CNY' 或 'USD'。",
5   func: async (input) => {
6     // 硬编码的汇率
7     const exchangeRate = 7.12;
8
9     let result;
10    if (input === "CNY") {
11      result = `1 CNY = ${((1 / exchangeRate).toFixed(4))} USD`;
12    } else if (input === "USD") {
13      result = `1 USD = ${exchangeRate} CNY`;
14    } else {
15      throw new Error("输入格式不正确。请使用 'CNY' 或 'USD'。");
16    }
17
18    return result;
19  },
20});
```

- 一个计算汇率的工具
- description 是最终提供给 LLM 的函数描述, 帮助 LLM 在合适的时候选择该工具
- 返回值会被转换成 String 输出给 LLM



```
1 const tools = [new Calculator(), exchangeRateTool];
2
3 const llm = new ChatOpenAI({
4   temperature: 0,
5 });
6
7 const agent = await createReactAgent({
8   llm,
9   tools,
10  prompt,
11 });
12
13 const agentExecutor = new AgentExecutor({
14   agent,
15   tools,
16 });
```

- 一个计算汇率的工具
- `description` 是最终提供给 LLM 的函数描述，帮助 LLM 在合适的时候选择该工具
- 返回值会被转换成 `String` 输出给 LLM



```
1 const tools = [new Calculator(), exchangeRateTool];
2
3 const llm = new ChatOpenAI({
4   temperature: 0,
5 });
6
7 const agent = await createReactAgent({
8   llm,
9   tools,
10  prompt,
11 });
12
13 const agentExecutor = new AgentExecutor({
14   agent,
15   tools,
16 });
```

- 构建 Agent
- 其中 Calculator 是 Langchain 提供的计算器 tool 的实现
- prompt 是前面介绍过的 prompt



```
1 const result = await agentExecutor.invoke({  
2   input: "我有 17 美元，现在相当于多少人民币？",  
3 });
```

Question: 我有 17 美元，现在相当于多少人民币？

Thought: To determine how much 17 USD is in CNY, I need to get the latest exchange rate between USD and CNY.

Action: exchange-rate-calculator

Action Input: USD

Observation: 1 USD = 7.12 CNY

Thought: To find out how much 17 USD is in CNY, I need to multiply 17 by the exchange rate of 7.12.

Action: calculator

Action Input: $17 * 7.12$

Observation: 121.04

Thought: I now know the final answer.

Final Answer: 17 美元相当于 121.04 人民币。

OpenAI Agent 实现

OpenAI Agent 实现

- 最靠谱的 Agent，官方支持 复杂输入 Tool 的调用，基于 Json schema 描述
- OpenAI Model 的 follow instruction 能力最强
- 基于官方 Function calling 功能
- 如果是做正事，选这个没错

Your code

LLM

1. Your application calls the API with your prompt and definitions of the functions the LLM can call

2. The model decides whether to respond to the user or whether one or more functions should be called

3. The API responds to your application specifying the function to be called and the arguments to call it with

4. Your application executes the function with the given arguments

5. Your application calls the API providing your prompt and the result of the function call your code just executed

...



```
1 // 创建自定义 tool, openAI-agent 支持 DynamicStructuredTool 和
  DynamicTool, 也就是支持多输入和单输入的工具,
2 // 提供了构建更复杂的工具的能力
3 const dateDiffTool = new DynamicStructuredTool({
4   name: "date-difference-calculator",
5   description: "计算两个日期之间的天数差",
6   schema: z.object({
7     date1: z.string().describe("第一个日期, 以YYYY-MM-DD格式表
      示"),
8     date2: z.string().describe("第二个日期, 以YYYY-MM-DD格式表
      示"),
9   }),
10  func: async ({ date1, date2 }) => {
11    const d1 = new Date(date1);
12    const d2 = new Date(date2);
13    const difference = Math.abs(d2.getTime() -
      d1.getTime());
14    const days = Math.ceil(difference / (1000 * 60 * 60 *
      24));
15    return days.toString();
16  },
17});
```

- 使用 json schema 定义输入函数, 支持对参数进行 类型、数量、必填 等约束
- 定义 Tool 更加灵活



```
1 const tools = [new Calculator(), exchangeRateTool,  
  dateDiffTool];  
2  
3 const prompt = ChatPromptTemplate.fromMessages([  
4   ["system", "You are a helpful assistant"],  
5   ["user", "{input}"],  
6   ["placeholder", "{agent_scratchpad}"],  
7 ]);
```

- prompt 非常简单，因为 reAct 中思考、观察、执行这些步骤，由 OpenAI 解决了
- agent_scratchpad 记录了 LLM 调用函数和结果的过程，帮助 AI 深度思考



```
1 const llm = new ChatOpenAI({  
2   temperature: 0,  
3 });  
4 const agent = await createOpenAIToolsAgent({  
5   llm,  
6   tools,  
7   prompt,  
8 });  
9  
10 const agentExecutor = new AgentExecutor({  
11   agent,  
12   tools,  
13 });
```



```
1 const res = await agentExecutor.invoke({  
2   input:  
3     "你要回答我两个问题，一个是我有 20 美元这相当于多少人民币？另一个是 今年  
4       是 2024 年，今年 5.1 和 10.1 之间有多少天？",  
4 });
```



```
1 {
2   "type": "function",
3   "function": {
4     "name": "exchange-rate-calculator",
5     "description": "获取人民币和美元之间的最新汇率。输入应为你当前的货
币, 'CNY' 或 'USD'。",
6     "parameters": {
7       "type": "object",
8       "properties": {
9         "input": {
10           "type": "string"
11         }
12       },
13       "additionalProperties": false,
14       "$schema": "http://json-schema.org/draft-07/schema#"
15     }
16   }
17 }
```

```
1 {
2   "type": "function",
3   "function": {
4     "name": "date-difference-calculator",
5     "description": "计算两个日期之间的天数差",
6     "parameters": {
7       "type": "object",
8       "properties": {
9         "date1": {
10           "type": "string",
11           "description": "第一个日期，以YYYY-MM-DD格式表示"
12         },
13         "date2": {
14           "type": "string",
15           "description": "第二个日期，以YYYY-MM-DD格式表示"
16         }
17       },
18       "required": [
19         "date1",
20         "date2"
21       ],
22       "additionalProperties": false,
23       "$schema": "http://json-schema.org/draft-07/schema#"
24     }
25 }
```



```
1 "tool_calls": [
2   {
3     "function": {
4       "arguments": "{\"input\": \"USD\"}",
5       "name": "exchange-rate-calculator"
6     },
7     "id": "call_xtqW8jAFYEY1giX3v7uuqRYHT",
8     "index": 0,
9     "type": "function"
10   },
11   {
12     "function": {
13       "arguments": "{\"date1\": \"2024-05-01\", \"date2\": \"2024-
14       \"name": "date-difference-calculator"
15     },
16     "id": "call_ldLVRjAqdkMMMG6TUMQ5CQ58",
17     "index": 1,
18     "type": "function"
19   }
20 ]
```

```
1 {
2   "lc": 1,
3   "type": "constructor",
4   "id": [
5     "langchain_core",
6     "messages",
7     "ToolMessage"
8   ],
9   "kwargs": {
10    "tool_call_id": "call_xtqW8jAFYEY1giX3v7uuqRYHT",
11    "content": "1 USD = 7.12 CNY",
12    "additional_kwargs": {
13      "name": "exchange-rate-calculator"
14    },
15    "response_metadata": {}
16  }
17 },
```



```
1 {
2   "lc": 1,
3   "type": "constructor",
4   "id": [
5     "langchain_core",
6     "messages",
7     "ToolMessage"
8   ],
9   "kwargs": {
10    "tool_call_id": "call_ldLVRjAqdkMMMG6TUMQ5CQ58",
11    "content": "153",
12    "additional_kwargs": {
13      "name": "date-difference-calculator"
14    },
15    "response_metadata": {}
16  }
17 }
```



```
1 {  
2   input: '你要回答我两个问题，一个是我有 20 美元这相当于多少人民币？ 另一个是  
今年是 2024 年，今年 5.1 和 10.1 之间有多少天？' ,  
3   output: '1. 根据最新汇率，1 美元 = 7.12 人民币。因此，20 美元相当于 20  
* 7.12 = 142.4 人民币。\\n' +  
4   '\\n' +  
5   '2. 2024 年 5 月 1 日和 2024 年 10 月 1 日之间有 153 天。'  
6 }
```

Lang Smith

The screenshot displays a developer tool interface with two main panels. The left panel, titled "TRACE", shows a hierarchical tree of function calls. The root node is "AgentExecutor" (3.23s, 242 invocations). It branches into "OpenAIToolsAgent" (1.66s), "RunnableAssign" (0.01s), "RunnableMap" (0.01s), "RunnableLambda" (0.01s), "ChatPromptTemplate" (0.00s), "ChatOpenAI gpt-3.5-turbo" (1.64s), "OpenAIToolsAgentOutputParser" (0.00s), "date-difference-calculator" (0.00s), "exchange-rate-calculator" (0.00s), and another "OpenAIToolsAgent" node (1.57s). This second "OpenAIToolsAgent" node also has children: "RunnableAssign" (0.00s), "RunnableMap" (0.00s), "RunnableLambda" (0.00s), "ChatPromptTemplate" (0.00s), "ChatOpenAI gpt-3.5-turbo" (1.57s), and "OpenAIToolsAgentOutputParser" (0.00s). The right panel is titled "OpenAIToolsAgent" and contains tabs for "Run", "Feedback", and "Metadata". The "Run" tab is selected, showing the "Input" section with a JSON code snippet. The code is a step-based action definition for an "exchange-rate-calculator" tool, which includes invoking it with an input of "USD" and logging the invocation details. The JSON code is as follows:

```
1 v {
2   "input": "你要回答我两个问题，一个是我有 20 美元这相当于多少人民币？另一个是 今年是 2024 年，今年 5.1 和 10.1 之间有多少天？",
3   "steps": [
4     {
5       "action": {
6         "tool": "exchange-rate-calculator",
7         "toolInput": {
8           "input": "USD"
9         },
10        "toolCallId": "call_KMYnDJF1rHGIJerYzNbdbceA",
11        "log": "Invoking \"exchange-rate-calculator\" with {\"input\": \"USD\"}\n",
12        "messageLog": [
13          {
14            "lc": 1,
15            "type": "constructor",
16            "id": [
17              "langchain_core",
18              "messages",
19              "AIMessageChunk"
20            ],
21            "kwargs": {
22              "content": "",
23              "additional_kwargs": {
24                "tool_calls": [
25                  {
26                    "function": {
27                      "arguments": "{\"input\": \"USD\"}",
28                      "name": "exchange-rate-calculator"
29                    },
30                    "id": "call_KMYnDJF1rHGT1erYzNbdbceA"
31                  }
32                ]
33              }
34            }
35          }
36        ]
37      }
38    }
39  ]
40}
```

Q & A

- 🧑‍💻 Kai
- 主页: <https://kaiyi.cool>
- 课程代码: <https://github.com/RealKai42/lm-course-basic>
- 记得点个 Star ⭐



