



# 仓颉编程语言与苍穹AI智能体框架

杨海龙 博士

2024年10月23日



# 仓颉编程语言

仓颉编程语言是华为编译器与编程语言实验室主导开发的编程语言。仓颉作为一款面向全场景应用开发的现代编程语言：

- 设计很多现代语言优秀特性，包括跨语言、宏编程、内嵌DSL等
- 支持编译器和运行时，CJNative和CJVM
- 开箱即用的 IDE 工具链支持，VSCode插件，CangjieStudio



<https://gitcode.com/Cangjie>

15:08:19



<https://gitcode.com/Cangjie-TPC>





# 仓颉的跨语言调用

## 仓颉与C语言的互操作

为了拥抱C语言生态，仓颉提供了C语言互操作机制。

- 仓颉支持调用 C 语言的函数，也支持 C 语言调用仓颉的函数。
- 在仓颉中要调用 C 的函数，需要在仓颉语言中用 `@C` 和 `foreign` 关键字声明这个函数，

```
/* duckdb包说明
 * duckdb数据库仓颉语言驱动
 *
 */
//引入外部函数
foreign func duckdb_open(path_p: CPointer<UInt8>, outdatabase: CPointer<duckdb_database>): duckdb_state
foreign func duckdb_close(database: CPointer<duckdb_database>): Unit
foreign func duckdb_connect(database: duckdb_database, out_connection: CPointer<duckdb_connection>): duckdb_state
foreign func duckdb_disconnect(connection: CPointer<duckdb_connection>): Unit
foreign func duckdb_query(connection: duckdb_connection, query: CPointer<UInt8>, result: CPointer<duckdb_result>): duckdb_state
foreign func duckdb_column_name(result: CPointer<duckdb_result>, col: idx_t): CPointer<UInt8>
```



# 仓颉的跨语言调用

## 仓颉与Python的互操作

为了兼容强大的计算和 AI 生态，仓颉支持与 Python 语言的互操作调用。Python 的互操作通过 std 模块中的 ffi.python 库为用户提供能力。

### 限制：

- 目前 Python 互操作仅支持在 Linux 平台使用，并且仅支持仓颉编译器的 CJNative 后端。
- 建议用户在执行 Python 互操作相关代码前，配置仓颉默认栈大小至少为 1MB。
- 仅支持 Python 3 版本。
- Python 回调仓颉代码需要通过 C 作为介质进行调用。



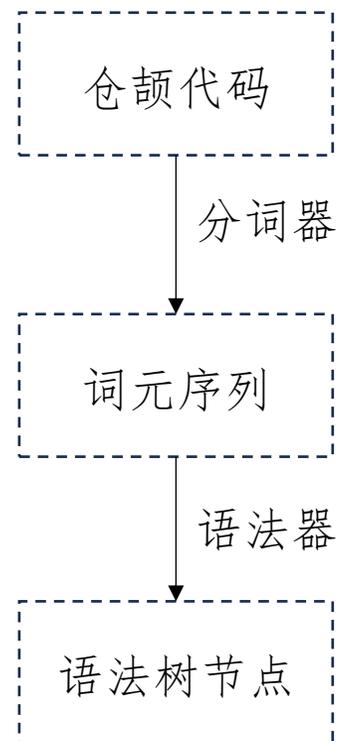


# 仓颉的宏编程

宏可以理解作为一种特殊的函数。一般的函数在输入的值上进行计算，然后输出一个新的值，而宏的输入和输出都是程序本身。这段输出的程序随后用于编译和执行。为了把宏的调用和函数调用区分开来，我们在调用宏时使用@加上宏的名称。



.....





# 仓颉的宏编程

仓颉语言提供了一些预定义的编译标记，可以通过这些编译标记控制仓颉编译器的编译行为。

源码位置

`@sourcePackage()` 当前宏所在的源码的包名

`@sourceFile()` 当前宏所在的源码的文件名

`@sourceLine()` 当前宏所在的源码的代码行

条件编译

条件编译使用 `@When` 标记，包括平台适应、功能选择、调试支持、性能优化

C优化

`@FastNative`标记用于优化对 C 函数的调用

```
@FastNative foreign func strlen(str: CPointer<UInt8>): UIntNative
```



# 仓颉的内嵌DSL

通过内嵌的DSL特性方便实现：

- 单智能体，实现代码和自然语言的有机融合；
- 多智能体的协作，包括线性协作、主从协作、自由协作；

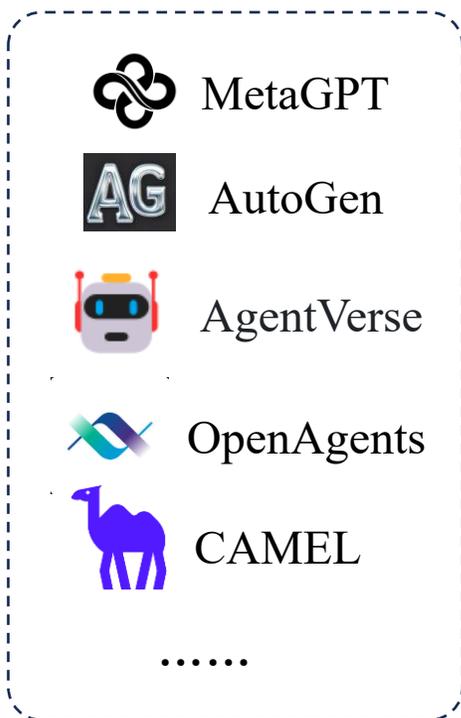
当然，用户也可以使用仓颉宏编程实现自己的DSL特性；

```
@agent class Planner {  
  @prompt[pattern=APE] (  
    action: "帮助用户制定旅行路线",  
    purpose: "让用户在计划时间内多参观景点并得到充分休息",  
    expectation: "生成一条合理的旅游景点路线, 包括时间、景点、通勤等信息"  
  )  
}
```





# AI智能体框架



这些智能体框架都是基于Python 

那么是否有基于仓颉语言的多智能体框架? 



# 苍穹：基于仓颉的AI多智能体框架

苍穹（CangChain）是基于仓颉（Cangjie）通用的多智能体框架，由仓颉语言社区支持、高校参与建设的独立的开源项目，包括可解释性、多模型混合、知识增强、逻辑增强、自适应角色定义、可执行图等特性。



- 2023年7月，构建苍穹的骨架。  
- 2023年9月，基础设施建设，包括LLMAPI、Tools。   
- 2023年11月，单智能体以及私有化部署。 
- 2024年5月，多智能体系统。  
- 2024年6月，代码开发多智能体系统Demo。  



# 苍穹的架构

应用层

代码开发

教育教学

生物信息

模拟仿真

其他应用...

框架层

多智能体系统自主构建机制

规划+多角色+执行图

知识增强+逻辑增强

基础通用单智能体

知识库

逻辑求解

Agent机制

LLM API

工具箱

数据连接

原生RAG

逻辑增强

工具/LLM选择  
人工监督  
反思、基于规则的语义对抗

GLM  
GPT +混合  
Llama

Google  
Wiki  
Math...

向量数据库  
关系数据库  
图数据...

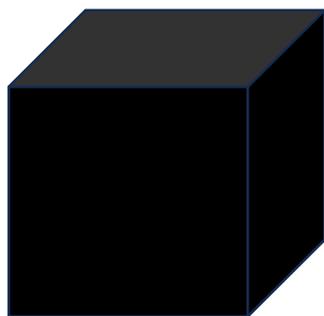
知识分割  
预检索  
后检索  
...

基于规则的逻辑嵌入  
一阶逻辑 +外部求解器...

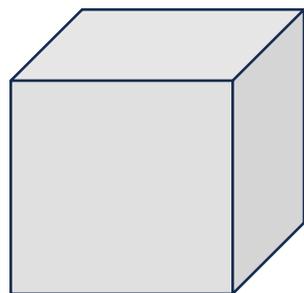


# 苍穹的可解释性

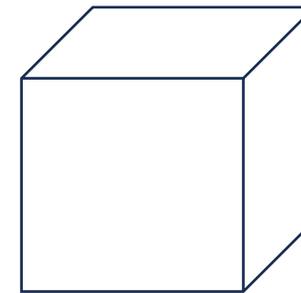
数据驱动的黑盒模型



数据驱动和知识驱动的灰盒模型



透明模型（未来）

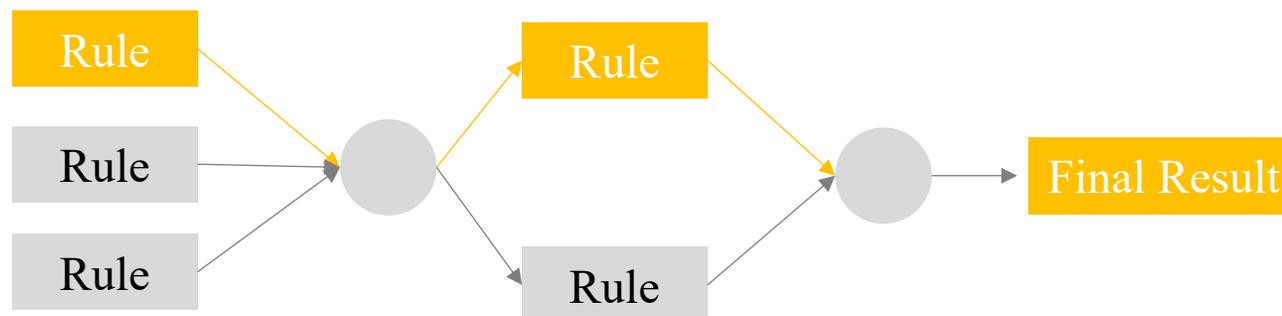
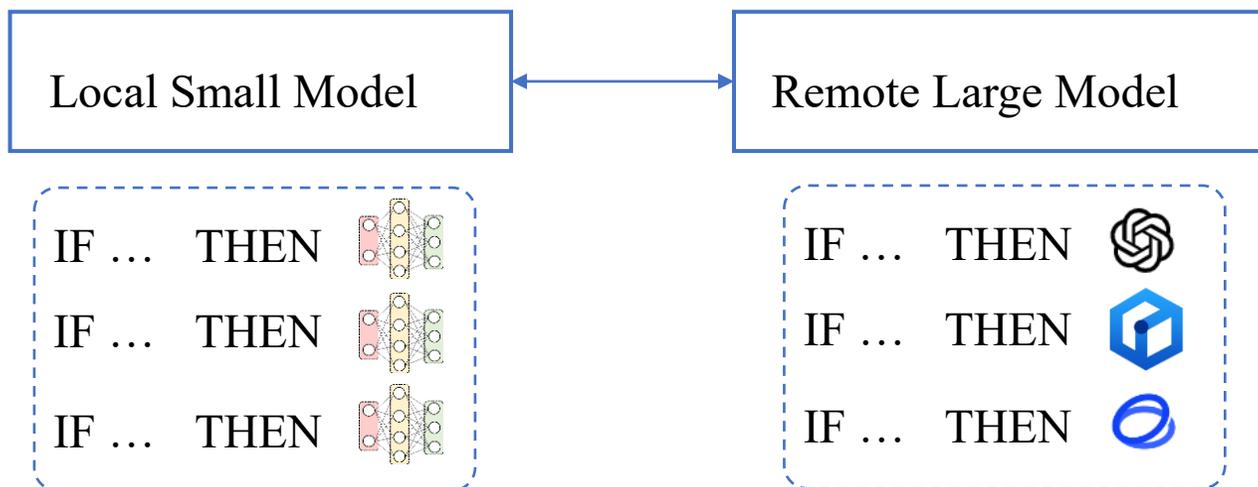


```
num = float(input("输入一个特征:"))  
if num > 0:  
      
elif num == 0:  
      
else:  
    
```

```
num = float(input("输入一个特征:"))  
if num > 0:  
    print("正样本")  
elif num == 0:  
    print("零")  
else:  
    print("负样本")
```

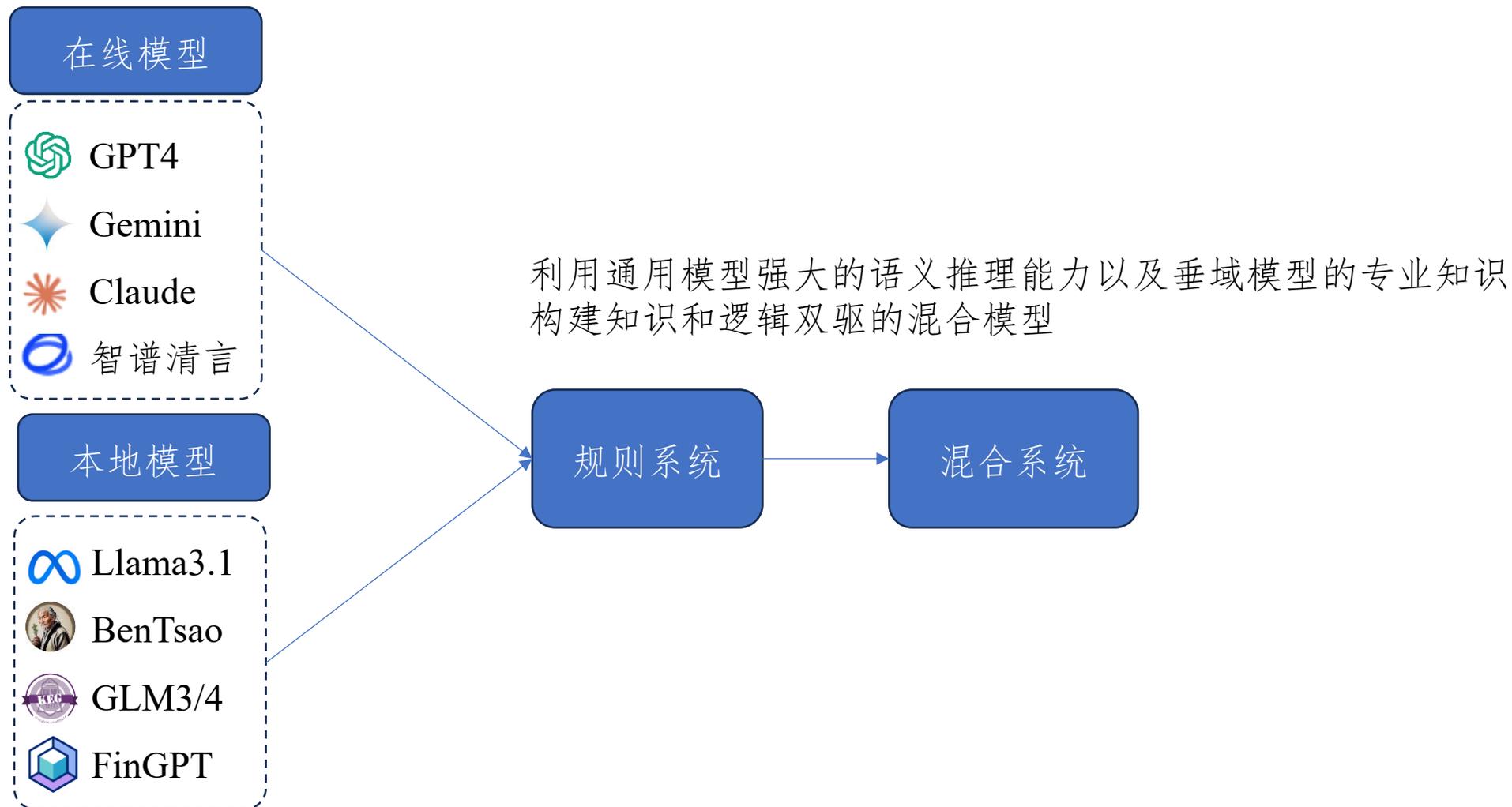


# 苍穹的可解释性





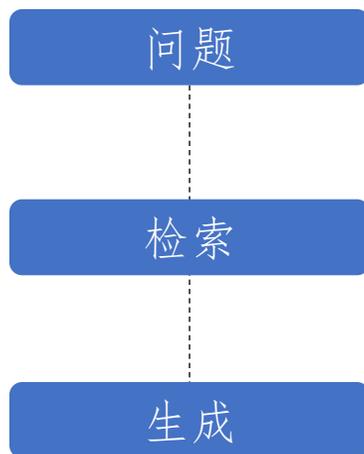
# 苍穹的多模型混合



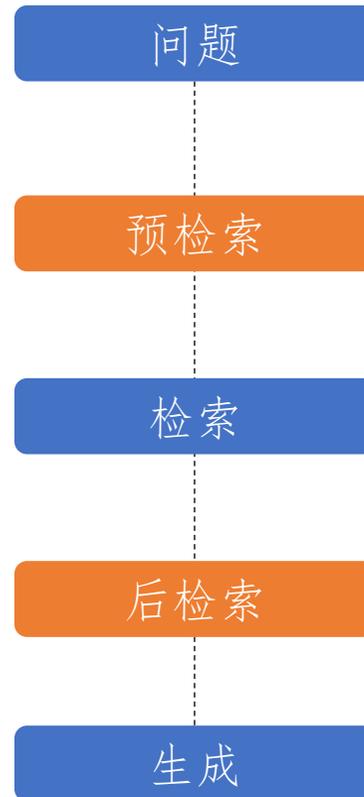


# 苍穹的知识增强

原生的RAG



RAG的增强

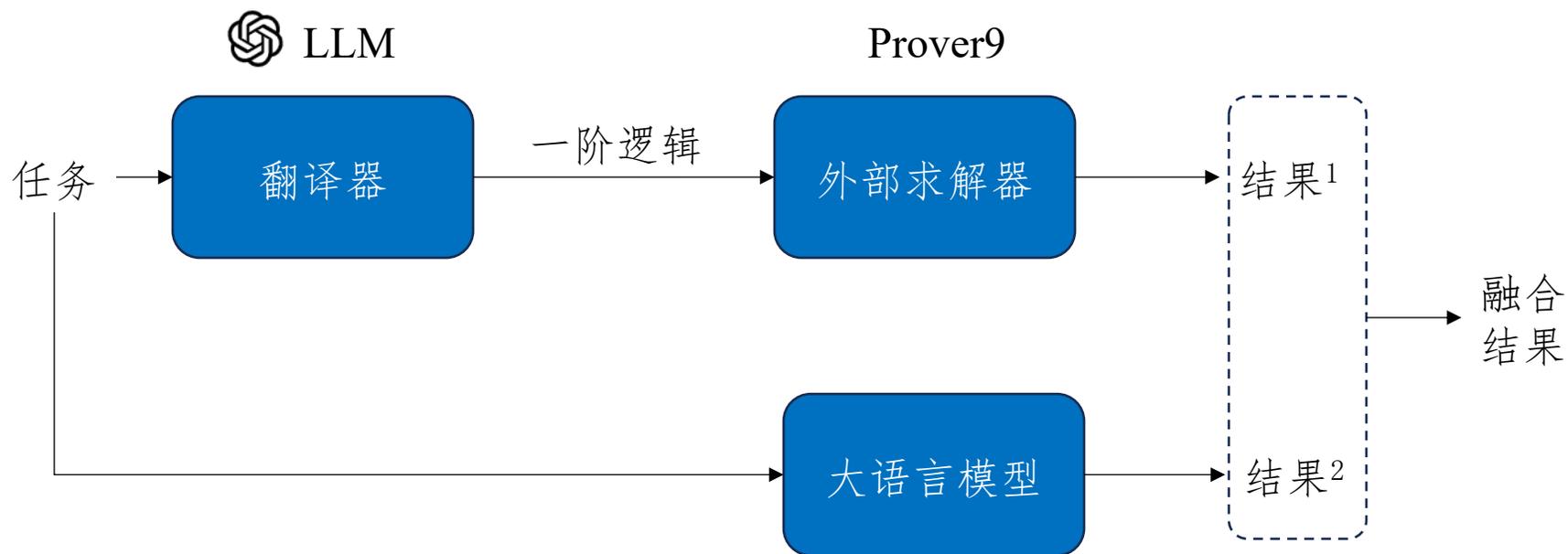


改进原始查询以提高检索效果，解决诸如查询表述不清、语言复杂性和歧义等问题。

对检索到的文档块进行后处理，以更好地利用上下文信息。

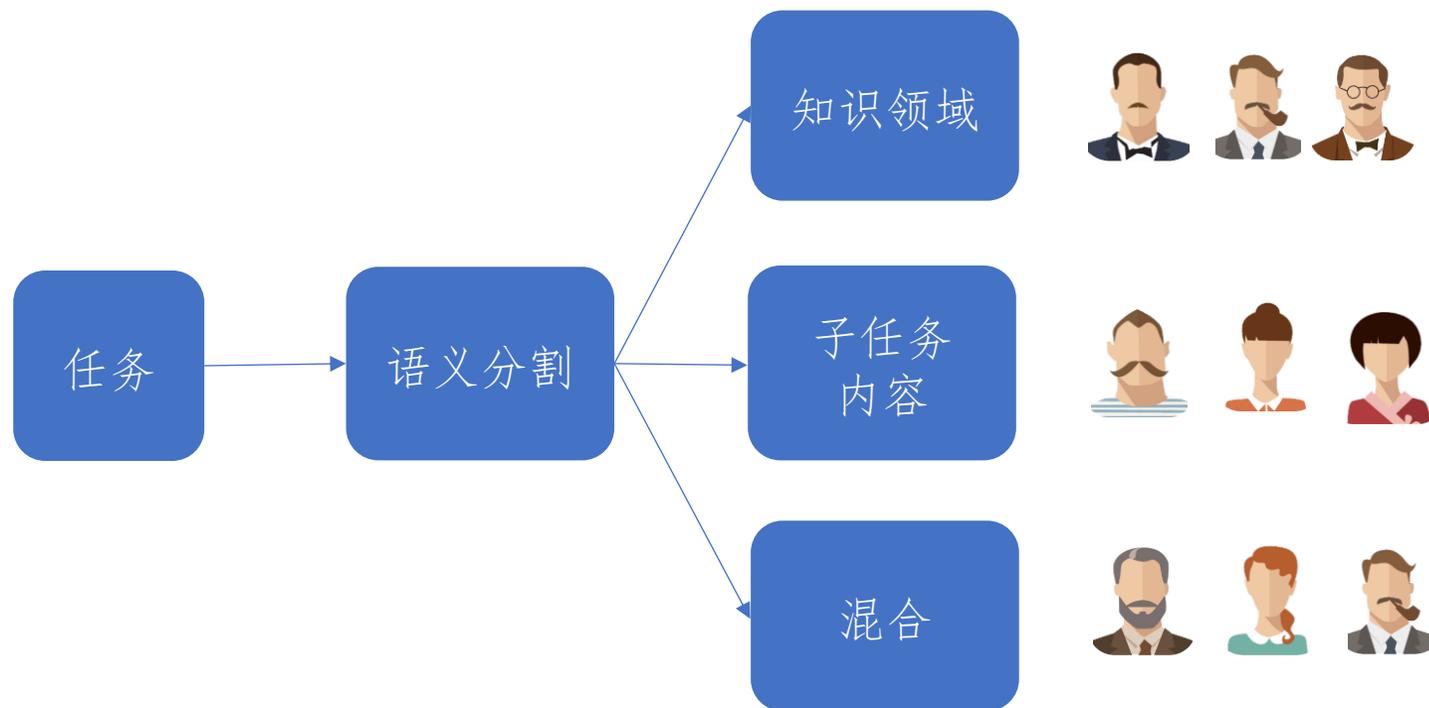


# 苍穹的逻辑增强



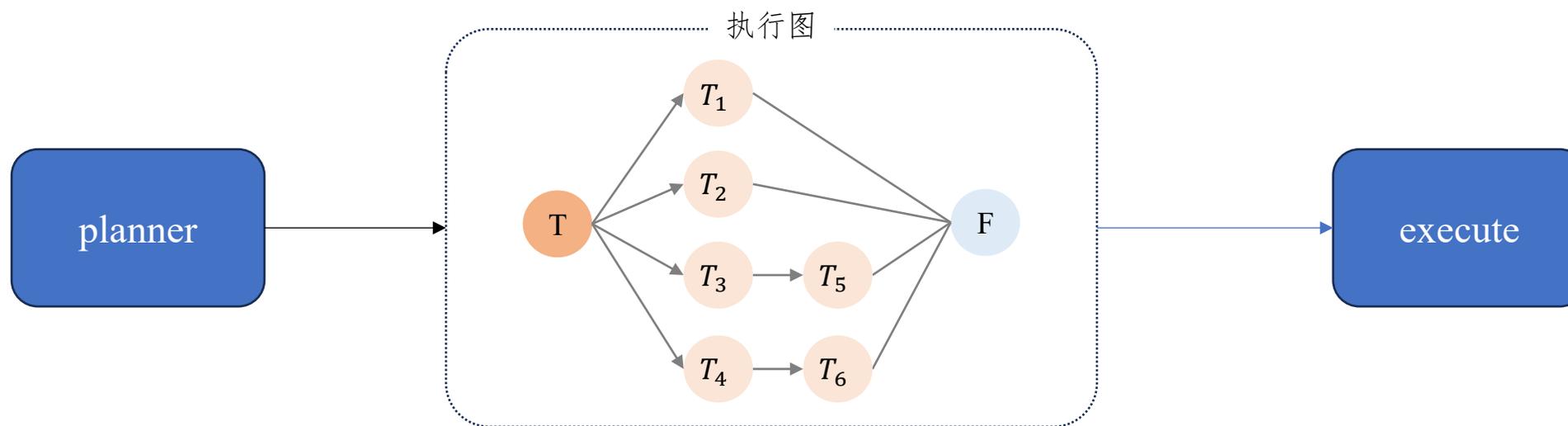


# 苍穹的自适应角色定义





# 苍穹的可执行图





# 苍穹的应用

## 代码生成

根据开发任务分解，模拟真实的开发流程，涉及的智能体角色包括文档撰写，代码开发，代码测试，代码检视等。

## 多智能体邮件处理系统

帮助客户高效处理邮件，包含钓鱼邮件识别、翻译、摘要、回信等不同角色的智能体。

## xx人才文档系统

辅助人才公司书写人才推荐材料，特别是领域专业知识，普通的职员无法胜任时，作为领域专家的智能体会根据领域前研的研究输出材料内容。



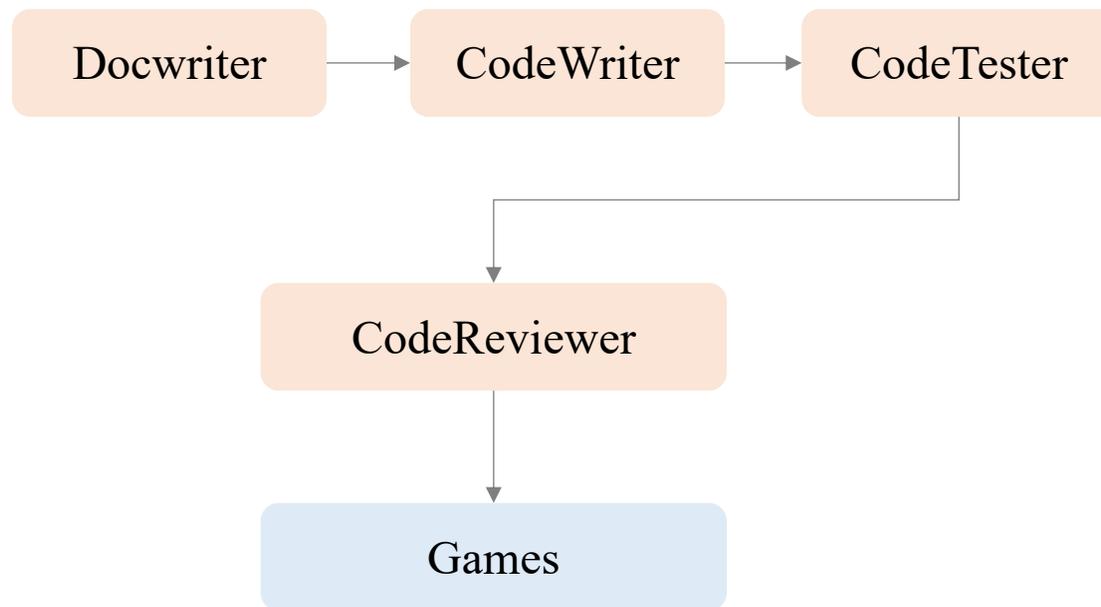
# 苍穹的应用：代码生成

```
planner.cj x
src > agent > planner.cj > PlannerPromptTemplate > format(HashMap<String, String>): String
45 // Planner主要是用来挑选下一步的Agent角色
46 public class PlannerAgent <: Role {
47     public init(tools!: ArrayList<BaseTool> = ArrayList<BaseTool>())
48         super("Planner", "The manager of multi agent to decide which
49             println("PlannerAgent init done.")
50     }
51
52     public func generate(input: Dict): String{
53         let output = super.predict(input["text"])[0]["text"]
54         output
55     }
56
57     public func shouldContinue(res: JsonObject): Bool {
58         match (res.get("message")) {
59             case None => return true
60             case Some(_) => return false
61         }
62     }
63 }
```

```
OpenAI init done.
=====
agent_name: Doc_Writer
query:编写一个2048游戏的文档，其中包括游戏规则、功能需求和实现细节。
=====
Executing role: Doc_Writer
OpenAI init done.
2048_Game_Documentation.txt is create successfully in current directory.

OpenAI init done.
=====
OpenAI init done.
=====
OpenAI init done.
=====
agent_name: Code_Writer
query:根据2048游戏的文档，编写2048游戏的代码。
=====
Executing role: Code_Writer
OpenAI init done.
2048_game.py is create successfully in current directory.
```

团队设计并实现了一个多智能体（Multi-Agent）的系统，包含以下四个关键角色：Docwriter、CodeWriter、CodeTester和CodeReviewer。





# 苍穹的应用：多智能体邮件处理系统

基于GLM构建，使用MindIE套件，运行在昇腾310芯片上。

原始邮件

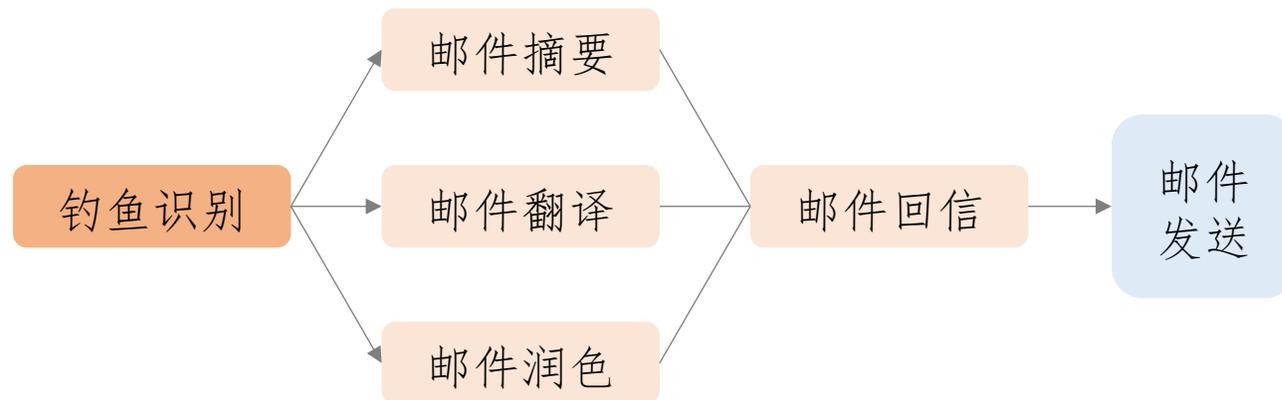
请输入原始邮件内容

请选择功能

邮件摘要  邮件翻译  邮件回信  钓鱼邮件识别  邮件润色

清空 提交

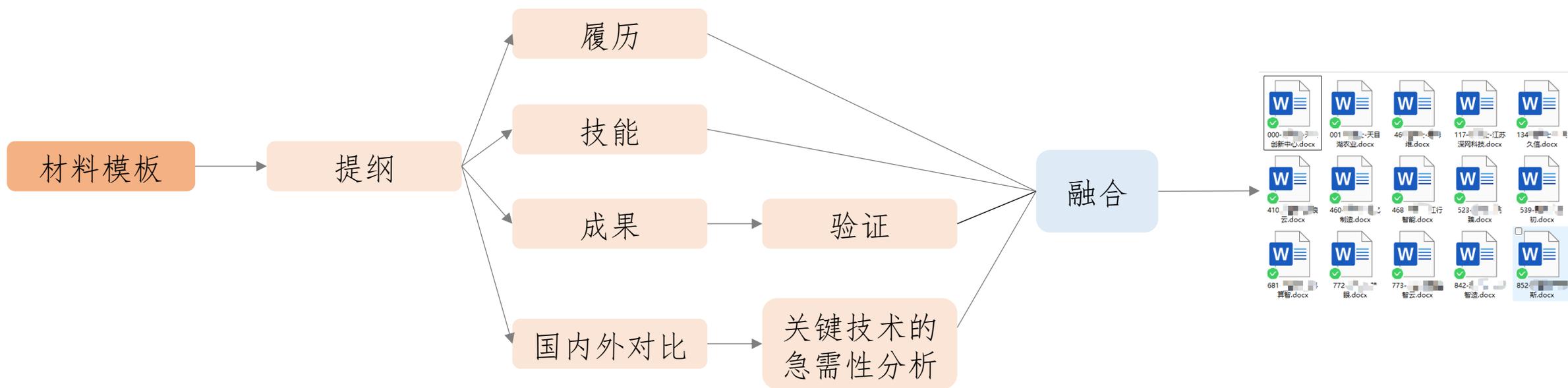
邮件处理执行图





# 苍穹的应用：XX人才文档系统

人才材料撰写执行图





# 苍穹的总结和规划

## 总结:

- 经历单智能体到多智能体的转变;
- 可解释智能体领域的探索;
- 自适应的角色定义机制;

## 规划:

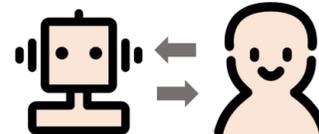
- 加强交互,除了人-智能体,还有智能体-智能体,智能体-人;
- 智能体平台,智能体标准化;
- 群体智能系统,智能体自我学习,私有的知识库;
- 拥抱仓颉生态,深耕教育、软件开发、智能管理等领域;



Single Agent



Agent-Agent



Agent-Human



THANKS

