AiDD AI+ 软件研发数字峰会
AI+ software Development Digital summit

AI驱动软件研发
全面进入数字化时代

中国·深圳 11.24-25

AI+
software
Development
Digital
summit

代码生成模型基座
在去哪儿网的代码生成应用

王植萌 去哪儿网

科技生态圈峰会 + 深度研习

——1000 + 技术团队的共同选择

KEYLINK ing

K+峰会

K+峰会 深圳站

K+ 全球软件研发行业创新峰会

会议时间：2024.05.24-25

K+峰会 上海站

K+ 全球软件研发行业创新峰会

会议时间：2024.09.20-21

AiDD 峰会

AiDD峰会 深圳站

AI+ 软件研发数字峰会

会议时间：2023.11.24-25

AiDD峰会 北京站

AI+ 软件研发数字峰会

会议时间：2024.07.19-20

AiDD峰会 深圳站

AI+ 软件研发数字峰会

会议时间：2024.11.15-16

演讲嘉宾

王植萌

去哪儿网基础研发团队负责人/技术委员会主席/人工智能委员会公共技术分会负责人

专注于SOP+AIAgents大模型应用

大模型基座在软件工程领域提效

目录
CONTENTS

PART 01

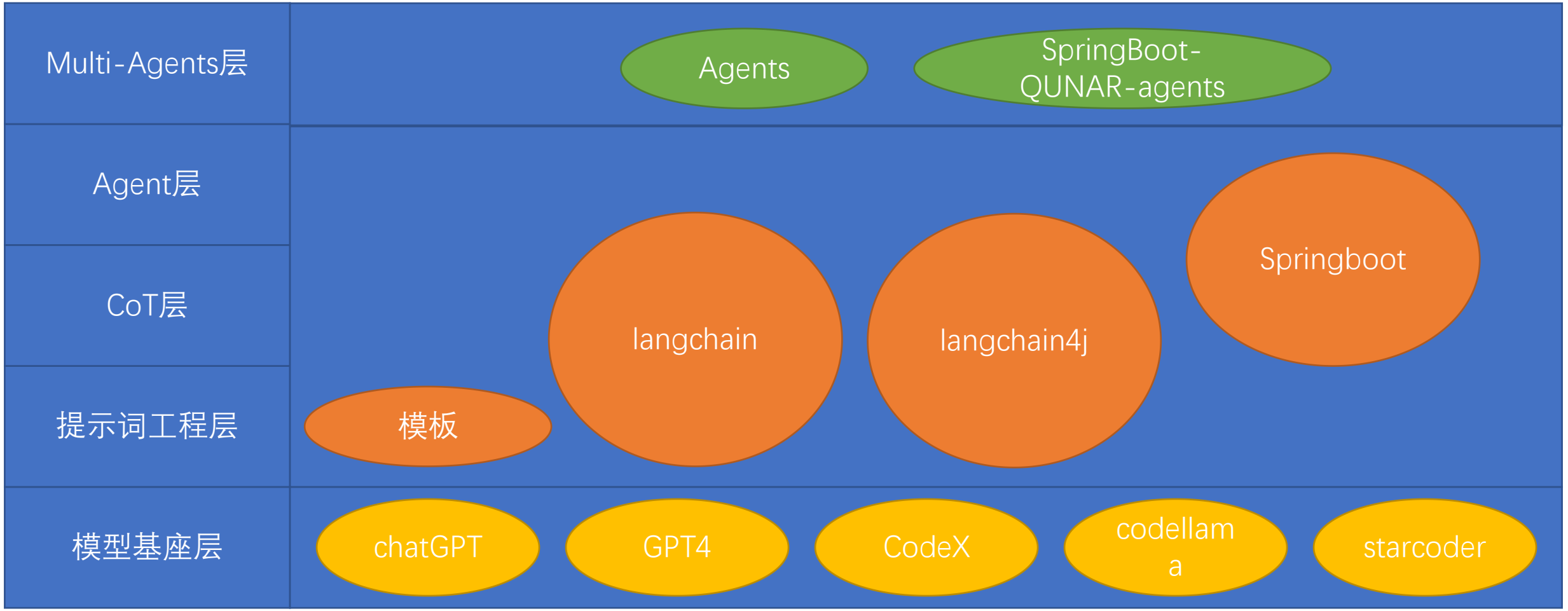
代码生成模型基座应用的目的与目标

代码生成模型基座的目的与目标

- 数字化转型
- 瘦身
- AI引领的全流程提效

代码生成模型基座的目的与目标

代码生成模型基座的目的与目标



PART 02
代码生成模型基座的应用策略

代码生成模型基座的应用策略

DMAIC原则

定义：用户接受率、用户留存率、用户覆盖度

度量：用户接受率(NPS)、用户留存率(周活跃用户)、用户覆盖率(相对于工程师全量的覆盖度)

分析：哪些场景是用户的痛点(TOP3:需求分析阶段、代码阶段、日常解答问题)

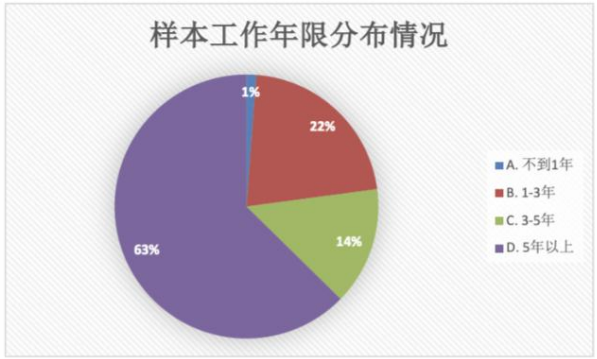
提升：围绕需求阶段、开发测试阶段、日常值班来规划场景

保持：成本可控，ROI可持续的方式来形成正向循环

需求调研与分析

问卷结果汇总与分析：

样本工作年限分布：



开发活动时长占比：

开发活动	占比
需求分析(业务流程梳理与代码逻辑梳理)	30%
代码开发	23.23%
环境创建与编译部署	8.46%
代码审查	9.85%
问题排查与自测联调	13.79%
会议与沟通	16.46%
其他	6.08%

在你的研发过程中，除去需求分析和方案梳理以外，其他活动中的耗时占比是什么样的，不涉及的选项默认为0，各项总和为100 [比重题]

选项	平均分	比例
代码开发与编写 [详细]	33.19	33.19%
环境创建与编译部署 [详细]	12.08	12.08%
代码审查 [详细]	9.86	9.86%
问题排查与自测联调 [详细]	19.71	19.71%
会议与沟通 [详细]	16.47	16.47%
其他 [详细]	8.69	8.69%

代码开发方面

DMAIC原则

定义：生成、补全代码的BenchMark测试，生成、补全代码的用户接受度，生成、补全代码的应用普及度生成、补全代码在工程师提交代码中的占比

度量：用户接受度使用AI开发助手中点击Tab键采纳的数量和生成代码到用户IDE的数量做比来衡量，BenchMark采用QUNAR自定义的业务场景代码问题来作为全集，100分为github-copilot的生成结果，代码提交占比按照工程师接纳行数与其周提交代码行数做比

分析：关键节点包括：模型选择、生成速度、上下文输入、返回结果评估、返回结果完整性校验、用户交互时机选择，用户操作结果回收

提升：围绕能跑通、可补、可用、有场景用、如何证明自己可用来不断提升

保持：监控、统计面板、新模型实验的SOP

PART 03
代码生成模型基座的落地难点

用户对于代码生成工具的负向观点

对正常写代码思路有干扰

功能不够安全

代码补全接受率不高

补全太慢

与工作环境集成不够

补全代码与上下文结合不足

代码生成模型的落地难点

模型基座：开源、商用？

模型部署：大小？是否量化？

提示词工程：模板？上下文？光标周围？引用文件、类？

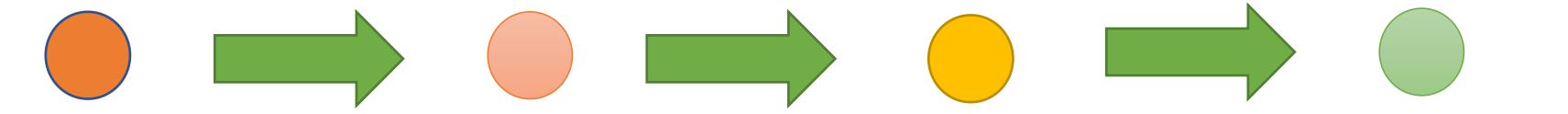
返回结果评估：完整？能否编译通过？

与用户交互：IDEA？Web？VSCode？与GitHub Copilot插件如何兼容？

结果如何度量：试用、推广时机？

模型基座

代码补全-开源模型
代码生成、CR等功能-微软GPT3.5接口



Replit_3b	starcoder_15.5b	codellama_7b	codellama_13b
6月，AI开发助手首次具备代码补全能力	7月，具备中间补全能力	9月，代码补全体验达到可用水平，开始进行推广应用	11月，代码补全BenchMark与github copilot持平，全司内推荐应用

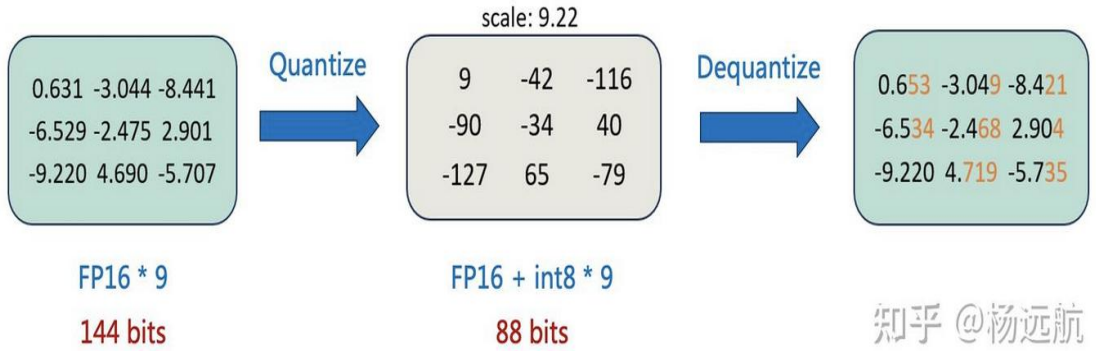
模型部署

GPU内存 量化 8bit/4bit

吞吐能力 TGI&vllm

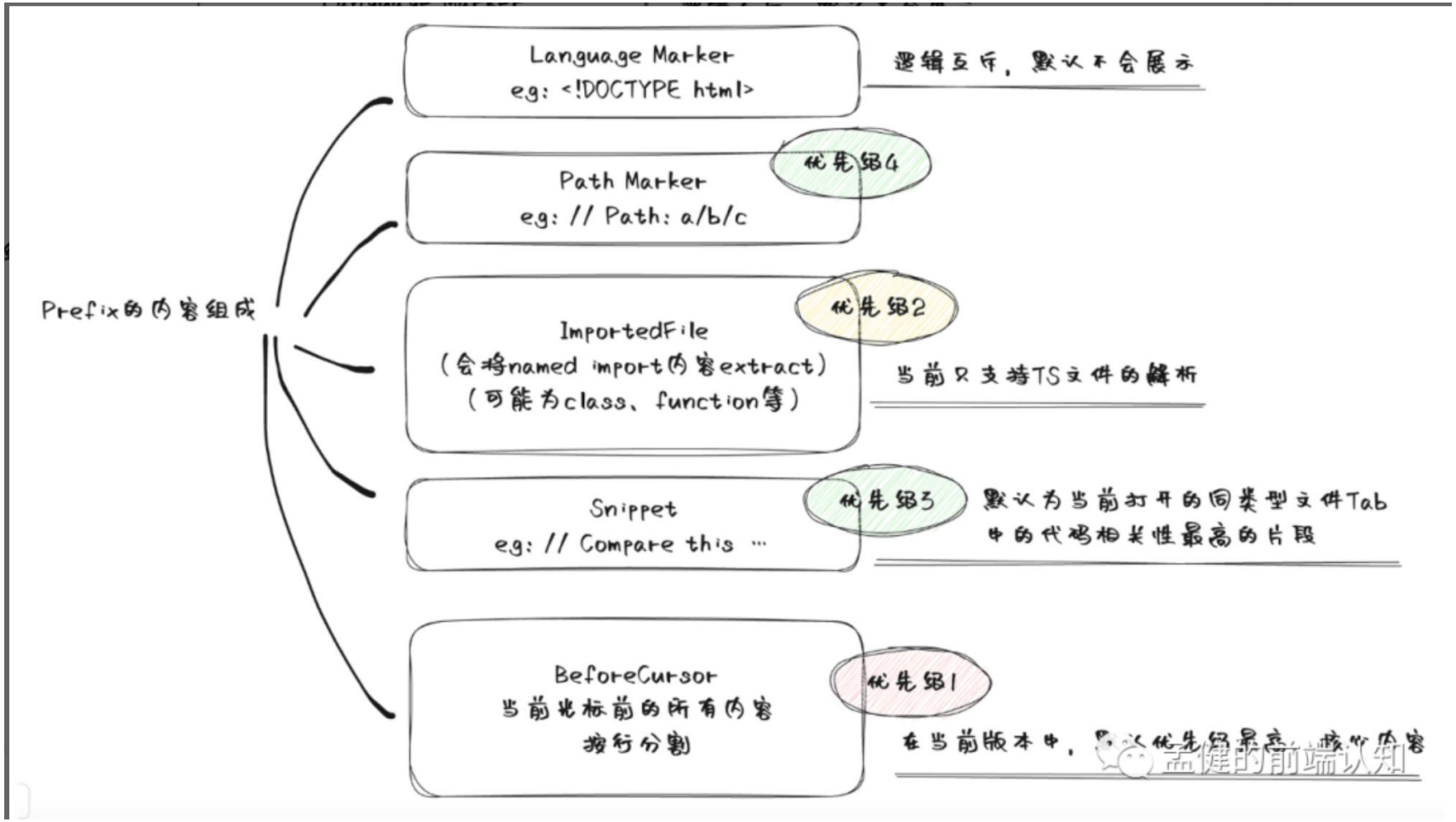
模型量化是将浮点数值转化为定点数值，同时尽可能减少计算精度损失的方法。

具体而言，模型量化是一种压缩网络参数的方式，它将神经网络的参数（weight）、特征图（activation）等原本用浮点表示的量值换用定点（整型）表示，在计算过程中，再将定点数据反量化回浮点数据，得到结果。



提示词工程

如何构建合理的System Prompt?



返回结果评估

构建适合本公司实际情况的评价样本

1.通过方法注释，生成基础的通用算法

2.对于已有方法，生成该方法的方法体注释，包括方法的参数列表注释

3.对于无注释的方法，自动根据方法体内上下文生成业务代码，尽量包含日志和监控打点

4.对于有注释的方法，自动根据方法体内上下文及注释中的提示生成业务代码，尽量包含日志和监控打点

5.打印日志时，自动补全想要输出的占位符变量

6.打印监控时，在try catch块中，根据上文成功的正向打点，在catch块补全失败的反向打点

7.对于同一个类中的方法调用，自动补全参数列表

8.对于同一个类中的方法调用，自动补全完整业务

9.自动识别其他类（实体）中的字段，并正确调用getter/setter

与用户交互

什么时候触发补全?

触发时刻的用户感知，如何让用户感知到同时维持心流状态

用户保持心流状态的最长可等待时间

返回结果自动评分

返回结果自动校验

结果如何度量

BenchMark度量

用户接受率度量

用户补全/提交比度量

用户应用覆盖度

用户周度应用UV

PART 04
问题解决的具体方案和结果

问题解决的具体方案与结果

模型基座选型：两条腿走路

模型	速度耗时	效果
Copilot	1.5-3s	编译错误1 空：0 多补：0 正常：9
Codex	2-6s	编译错误：4（小错误） 空：0 多补：2 正常：4
CodeLlama	3-6s	编译错误：2 空：2 多补：2 正常：3
starcode	3-6s	编译错误：1 空：6 多补：1 正常：1

	商业大模型基座(ChatGPT、GPT4、CodeX)	开源大模型基座(llama2、starcoder、codellama)
模型基座质量	☑	☐
推理效果	☑	☐
应用体验	☑	☐
信息安全	☐	☑
价值观对齐	☐	☑
去除幻觉	☐	☑

问题解决的具体方案与结果

模型部署及初步效果

	生成速度	占用资源	向下补全	中间补全	补全接受率	补全结果展示比
Replit_3b	全体员工开放 4~6s	2*A30 GPU 100%	支持	不支持	5%	3%
StarCoder_15.5b	全体员工开放 4~6s	2*A30 GPU 100%	支持	支持	13%	1%
CodeLlama_7b	全体员工开放 4~6s	2*A30 GPU 100%	支持	支持	17%	1%
CodeLlama_7b 4bit量化	全体员工开放 1~3s	2*A30 GPU 50%	支持	支持	17%	10%
github copilot(codeX)	10%员工开放 1~2s	-	支持	支持	> 17%	-

初始的度量

模型	速度耗时	效果
Copilot	1.5-3s	编译错误1 空：0 多补：0 正常：9
Codex	2-6s	编译错误：4（小错误） 空：0 多补：2 正常：4
CodeLlama	3-6s	编译错误：2 空：2 多补：2 正常：3
starcode	3-6s	编译错误：1 空：6 多补：1 正常：1

初始的度量

定义一个空方法，在方法签名上做好注释，使用插件生成相应的算法代码。

```java
/**
 * 通过深度优先搜索，找到参数root中name为参数target的节点，并返回该节点
 *
 * @param root   根节点
 * @param target 目标名称
 */
public void dfs(Node root, String target) {
}
```

```java
/**
 * 通过深度优先搜索，找到参数root中name为参数target的节点，并返回该节点
 *
 * @param root   根节点
 * @param target 目标名称
 */
public void dfs(Node root, String target) {
    if (root == null) {
        return;
    }
    if (root.name.equals(target)) {
        System.out.println(root.name);
        return;
    }
    for (Node node : root.children) {
        dfs(node, target);
    }
}
```

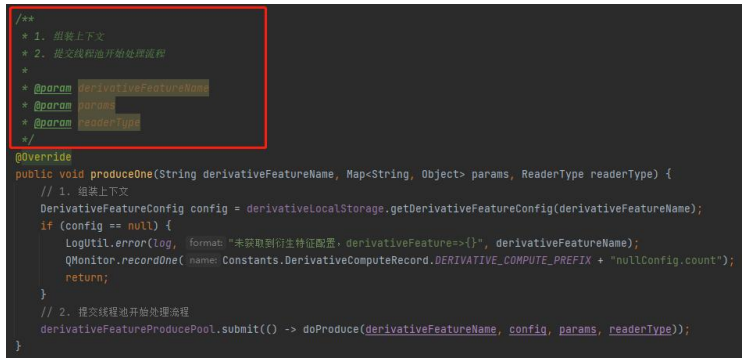
```java
/**
 * 通过深度优先搜索，找到参数root中name为参数target的节点，并返回该节点
 *
 * @param root   根节点
 * @param target 目标名称
 */
public void dfs(Node root, String target) {
    if (root == null) {
        return;
    }
    if (root.name.equals(target)) {
        return;
    }
    for (Node node : root.children) {
        dfs(node, target);
    }
}
```

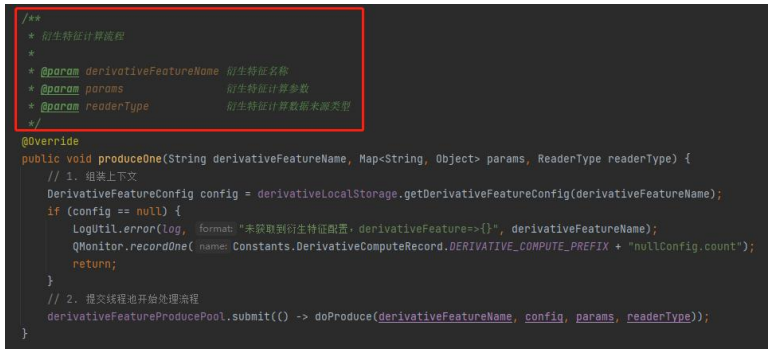
```java
/**
 * 通过深度优先搜索，找到参数root中name为参数target的节点，并返回该节点
 *
 * @param root 根节点
 * @param target 目标名称
 */
public void dfs(Node root, String target) {
    if (root == null) return;
    if (root.name.equals(target)) {
        System.out.println("找到了");
    } else {
        dfs(root.left, target);
        dfs(root.right, target);
    }
}
```

初始的度量

在一个已写完整的方法上，通过键入 /** 来开始提示对方法注释的生成，要求注释中包含对方法内容的逻辑解释，以及方法参数列表的参数含义说明。

```java
@Override
public void produceOne(String derivativeFeatureName, Map<String, Object> params, ReaderType readerType) {
    // 1. 组装上下文
    DerivativeFeatureConfig config = derivativeLocalStorage.getDerivativeFeatureConfig(derivativeFeatureName);
    if (config == null) {
        LogUtil.error(log, format: "未获取到衍生特征配置，derivativeFeature=>{}", derivativeFeatureName);
        QMonitor.recordOne( name: Constants.DerivativeComputeRecord.DERIVATIVE_COMPUTE_PREFIX + "nullConfig.count");
        return;
    }
    // 2. 提交线程池开始处理流程
    derivativeFeatureProducePool.submit(() -> doProduce(derivativeFeatureName, config, params, readerType));
}
```





初始的度量

当我已经完成了一个方法的大部分代码时，我需要对一些边界条件做完善，如此时我需要对某个外部获取的实体判空，并输出日志监控以记录异常。

```java
@Override
public void produceOne(String derivativeFeatureName, Map<String, Object> params, ReaderType readerType) {
    // 1. 组装上下文
    DerivativeFeatureConfig config = derivativeLocalStorage.getDerivativeFeatureConfig(derivativeFeatureName);
    if
    // 2. 提交线程池开始处理流程
    derivativeFeatureProducePool.submit(() -> doProduce(derivativeFeatureName, config, params, readerType));
}
```

```java
@Override
public void produceOne(String derivativeFeatureName, Map<String, Object> params, ReaderType readerType) {
    // 1. 组装上下文
    DerivativeFeatureConfig config = derivativeLocalStorage.getDerivativeFeatureConfig(derivativeFeatureName);
    if (config == null) {
        LogUtil.error(log, format: "衍生特征配置不存在，derivativeFeatureName=>{}", derivativeFeatureName);
        return;
    }
    LogUtil.info(log, "衍生特征配置信息，derivativeFeatureName=>{},config=>{}", derivativeFeatureName, JacksonUtil.toJson(config));
    // 2. 提交线程池开始处理流程
    derivativeFeatureProducePool.submit(() -> doProduce(derivativeFeatureName, config, params, readerType));
}
```

```java
if
// 1. 校验衍生特征是否存在
if (config == null) {
    LogUtil.error(log, format: "衍生特征不存在，featureName=>{}", der
            QMonitor.recordOne( name: Constants.DerivativeComputeF
    return;
}

// 2. 校验衍生特征是否已经计算完成
if (derivativeLocalStorage.isDerivativeFeatureComputed(derivative
    LogUtil.info(log, format: "衍生特征已经计算完成，featureName=>{}",
            QMonitor.recordOne( name: Constants.DerivativeComputeF
    return;
}
```

初始的度量

在方法注释中明确提示边界条件的处理方式，期望按照注释生成相应的监控和日志。

```java
/**
 * 生产一个衍生特征
 * 当衍生特征配置不存在时，打日志及监控，并返回
 *
 * @param derivativeFeatureName 衍生特征名
 * @param params                查询参数
 * @param readerType            读取原子特征源类型
 */
@Override
public void produceOne(String derivativeFeatureName, Map<String, Object> params, ReaderType readerType) {
    // 1. 组装上下文
    DerivativeFeatureConfig config = derivativeLocalStorage.getDerivativeFeatureConfig(derivativeFeatureName);
    if (config == null) {

    }
    // 2. 提交线程池开始处理流程
    derivativeFeatureProducePool.submit(() -> doProduce(derivativeFeatureName, config, params, readerType));
}
```

```java
/**
 * 生产一个衍生特征
 * 当衍生特征配置不存在时，打日志及监控，并返回
 *
 * @param derivativeFeatureName 衍生特征名
 * @param params                查询参数
 * @param readerType            读取原子特征源类型
 */
@Override
public void produceOne(String derivativeFeatureName, Map<String, Object> params, ReaderType readerType) {
    // 1. 组装上下文
    DerivativeFeatureConfig config = derivativeLocalStorage.getDerivativeFeatureConfig(derivativeFeatureName);
    if (config == null) {
        LogUtil.error( usingJson: true, log, format: "衍生特征配置不存在，derivativeFeatureName=>{}", derivativeFeatureName);
        QMonitor.recordOne( name: Constants.DerivativeComputeRecord.DERIVATIVE_COMPUTE_PREFIX + "config.not.exist");
        return;
    }
    // 2. 提交线程池开始处理流程
    derivativeFeatureProducePool.submit(() -> doProduce(derivativeFeatureName, config, params, readerType));
}
```

```java
public void produceOne(String derivativeFeatureName, Map<String, Object> params, ReaderType readerType) {
    // 1. 组装上下文
    DerivativeFeatureConfig config = derivativeLocalStorage.getDerivativeFeatureConfig(derivativeFeatureName);
    if (config == null) {
        LogUtil.warn(log, format: "衍生特征配置不存在，featureName=>{}", derivativeFeatureName);
        QMonitor.recordOne( name: Constants.DerivativeComputeRecord.DERIVATIVE_COMPUTE_PREFIX + "config.not_exist");
        return;
    }
    if (config.getStatus() == 0) {
        LogUtil.warn(log, format: "衍生特征配置已禁用，featureName=>{}", derivativeFeatureName);
        QMonitor.recordOne( name: Constants.DerivativeComputeRecord.DERIVATIVE_COMPUTE_PREFIX + "config.disabled");
        return;
    }
    if (config.getStatus() == 2) {
        LogUtil.warn(log, format: "衍生特征配置已过期，featureName=>{}", derivativeFeatureName);
        QMonitor.recordOne( name: Constants.DerivativeComputeRecord.DERIVATIVE_COMPUTE_PREFIX + "config.expired");
        return;
    }
    // 2. 提交线程池开始处理流程
    derivativeFeatureProducePool.submit(() -> doProduce(derivativeFeatureName, config, params, readerType));
}
```

初始的度量

打印日志时，自动补全想要输出的占位符变量

```
private void doProduce(String derivativeFeatureName, DerivativeFeatureConfig config, Map<String, Object> params, ReaderType readerType) {
    try {
        LogUtil.info(log, format: "衍生特征进入处理流，featureName=>{}");
        // 1. 填充衍生特征计算过程数据
        List<Map<String, Object>> processData = fillProcessDataHandler.handle(readerType, config, params);
        // 2. 调用SDK计算衍生特征结果
        Map<String, Object> result = ComputeService.getInstance().handle(config, params, processData);
        // 3. 将衍生特征计算结果刷入Redis
        flushRedisHandler.handle(config, params, result);
        LogUtil.info(log, format: "衍生特征处理流程完毕，featureName=>{}", derivativeFeatureName);
        QMonitor.recordOne( name: Constants.DerivativeComputeRecord.DERIVATIVE_COMPUTE_PREFIX + "process.success");
    } catch (Exception e) {
        LogUtil.error( usingJson: true, log, format: "衍生特征处理流程发生异常，derivativeFeatureName=>{}", derivativeFeatureName, e);
        QMonitor.recordOne( name: Constants.DerivativeComputeRecord.DERIVATIVE_COMPUTE_PREFIX + "process.failed");
    }
}
```

```
private void doProduce(String derivativeFeatureName, DerivativeFeatureConfig config, Map<String, Object> params, ReaderType readerType) {
    try {
        LogUtil.info(log, format: "衍生特征进入处理流，featureName=>{}", derivativeFeatureName);
        // 1. 填充衍生特征计算过程数据
        List<Map<String, Object>> processData = fillProcessDataHandler.handle(readerType, config, params);
        // 2. 调用SDK计算衍生特征结果
        Map<String, Object> result = ComputeService.getInstance().handle(config, params, processData);
        // 3. 将衍生特征计算结果刷入Redis
        flushRedisHandler.handle(config, params, result);
        LogUtil.info(log, format: "衍生特征处理流程完毕，featureName=>{}", derivativeFeatureName);
        QMonitor.recordOne( name: Constants.DerivativeComputeRecord.DERIVATIVE_COMPUTE_PREFIX + "process.success");
    } catch (Exception e) {
        LogUtil.error( usingJson: true, log, format: "衍生特征处理流程发生异常，derivativeFeatureName=>{}", derivativeFeatureName, e);
        QMonitor.recordOne( name: Constants.DerivativeComputeRecord.DERIVATIVE_COMPUTE_PREFIX + "process.failed");
    }
}
```

```
private void doProduce(String derivativeFeatureName, DerivativeFeatureConfig config, Map<String, Object> params, ReaderType readerType) {
    try {
        LogUtil.info(log, format: "衍生特征进入处理流，featureName=>{}", derivativeFeatureName);
        // 1. 填充衍生特征计算过程数据
        List<Map<String, Object>> processData = fillProcessDataHandler.handle(readerType, config, params);
        // 2. 调用SDK计算衍生特征结果
        Map<String, Object> result = ComputeService.getInstance().handle(config, params, processData);
        // 3. 将衍生特征计算结果刷入Redis
        flushRedisHandler.handle(config, params, result);
        LogUtil.info(log, format: "衍生特征处理流程完毕，featureName=>{}", derivativeFeatureName);
        QMonitor.recordOne( name: Constants.DerivativeComputeRecord.DERIVATIVE_COMPUTE_PREFIX + "process.success");
    } catch (Exception e) {
        LogUtil.error( usingJson: true, log, format: "衍生特征处理流程发生异常，derivativeFeatureName=>{}", derivativeFeatureName, e);
        QMonitor.recordOne( name: Constants.DerivativeComputeRecord.DERIVATIVE_COMPUTE_PREFIX + "process.failed");
    }
}
```

初始的度量

打印监控时，在try catch块中，根据上文成功的正向打点，在catch块补全失败的反向打点

```
private void doProduce(String derivativeFeatureName, DerivativeFeatureConfig config, Map<String, Object> params, ReaderType readerType) {
    try {
        LogUtil.info(log, format: "衍生特征进入处理流，featureName=>{}", derivativeFeatureName);
        // 1. 填充衍生特征计算过程数据
        List<Map<String, Object>> processData = fillProcessDataHandler.handle(readerType, config, params);
        // 2. 调用SDK计算衍生特征结果
        Map<String, Object> result = ComputeService.getInstance().handle(config, params, processData);
        // 3. 将衍生特征计算结果刷入Redis
        flushRedisHandler.handle(config, params, result);
        LogUtil.info(log, format: "衍生特征处理流程完毕，featureName=>{}", derivativeFeatureName);
        QMonitor.recordOne( name: Constants.DerivativeComputeRecord.DERIVATIVE_COMPUTE_PREFIX + "process.success");
    } catch (Exception e) {
        LogUtil.error( usingJson: true, log, format: "衍生特征处理流程发生异常，derivativeFeatureName=>{}", derivativeFeatureName, e);
        QMonitor.recordOne();
    }
}
```

```
private void doProduce(String derivativeFeatureName, DerivativeFeatureConfig config, Map<String, Object> params, ReaderType readerType) {
    try {
        LogUtil.info(log, format: "衍生特征进入处理流，featureName=>{}", derivativeFeatureName);
        // 1. 填充衍生特征计算过程数据
        List<Map<String, Object>> processData = fillProcessDataHandler.handle(readerType, config, params);
        // 2. 调用SDK计算衍生特征结果
        Map<String, Object> result = ComputeService.getInstance().handle(config, params, processData);
        // 3. 将衍生特征计算结果刷入Redis
        flushRedisHandler.handle(config, params, result);
        LogUtil.info(log, format: "衍生特征处理流程完毕，featureName=>{}", derivativeFeatureName);
        QMonitor.recordOne( name: Constants.DerivativeComputeRecord.DERIVATIVE_COMPUTE_PREFIX + "process.success");
    } catch (Exception e) {
        LogUtil.error( usingJson: true, log, format: "衍生特征处理流程发生异常，derivativeFeatureName=>{}", derivativeFeatureName, e);
        QMonitor.recordOne( name: Constants.DerivativeComputeRecord.DERIVATIVE_COMPUTE_PREFIX + "process.failed");
    }
}
```

```
        Map<String, Object> result = ComputeService.getInstance().handle(config, params, processData);
        // 3. 将衍生特征计算结果刷入Redis
        flushRedisHandler.handle(config, params, result);
        LogUtil.info(log, format: "衍生特征处理流程完毕，featureName=>{}", derivativeFeatureName);
        QMonitor.recordOne( name: Constants.DerivativeComputeRecord.DERIVATIVE_COMPUTE_PREFIX + "process.success");
    } catch (Exception e) {
        LogUtil.error( usingJson: true, log, format: "衍生特征处理流程发生异常，derivativeFeatureName=>{}", derivativeFeatureName, e);
        QMonitor.recordOne( name: "\"DERIVATIVE_COMPUTE_PREFIX\" + \"process.failure\"", e);
    }

    @Override
    public void flushAll() {
        List<String> derivativeFeatureNames = derivativeLocalStorage.getDerivativeFeatureNames();
        for (String derivativeFeatureName : derivativeFeatureNames) {
            Map<String, Object> params = derivativeLocalStorage.getDerivativeFeatureParams(derivativeFeatureName);
        }
    }
```

初始的度量

对于同一个类中的方法调用，自动补全参数列表。

```java
@Override
public void produceOne(String derivativeFeatureName, Map<String, Object> params, ReaderType readerType) {
    // 1. 组装上下文
    DerivativeFeatureConfig config = derivativeLocalStorage.getDerivativeFeatureConfig(derivativeFeatureName);
    if (config == null) {
        LogUtil.error(log, format: "未获取到衍生特征配置，derivativeFeature=>{}", derivativeFeatureName);
        QMonitor.recordOne( name: Constants.DerivativeComputeRecord.DERIVATIVE_COMPUTE_PREFIX + "nullConfig.count");
        return;
    }
    // 2. 提交线程池开始处理流程
    derivativeFeatureProducePool.submit(() -> doProduce());
}
```

```java
@Override
public void produceOne(String derivativeFeatureName, Map<String, Object> params, ReaderType readerType) {
    // 1. 组装上下文
    DerivativeFeatureConfig_config = derivativeLocalStorage.getDerivativeFeatureConfig(derivativeFeatureName);
    if (config == null) {
        LogUtil.error(log, format: "未获取到衍生特征配置，derivativeFeature=>{}", derivativeFeatureName);
        QMonitor.recordOne( name: Constants.DerivativeComputeRecord.DERIVATIVE_COMPUTE_PREFIX + "nullConfig.count");
        return;
    }
    // 2. 提交线程池开始处理流程
    derivativeFeatureProducePool.submit(() -> doProduce(derivativeFeatureName, config, params, readerType));
}
```

初始的度量

对于同一个类中的方法调用，自动补全参数列表。

```java
@Override
public void produceOne(String derivativeFeatureName, Map<String, Object> params, ReaderType readerType) {
    // 1. 组装上下文
    DerivativeFeatureConfig config = derivativeLocalStorage.getDerivativeFeatureConfig(derivativeFeatureName);
    if (config == null) {
        LogUtil.error(log, format: "未获取到衍生特征配置，derivativeFeature=>{}", derivativeFeatureName);
        QMonitor.recordOne( name: Constants.DerivativeComputeRecord.DERIVATIVE_COMPUTE_PREFIX + "nullConfig.count");
        return;
    }
    // 2. 提交线程池开始处理流程
    derivativeFeatureProducePool.submit(() -> doProduce(derivativeFeatureName, config, params, readerType));
}

@Override
public void batchProduce(List<String> derivativeFeatureNames, Map<String, Object> params, ReaderType readerType) {
}
```

```java
@Override
public void produceOne(String derivativeFeatureName, Map<String, Object> params, ReaderType readerType) {
    // 1. 组装上下文
    DerivativeFeatureConfig config = derivativeLocalStorage.getDerivativeFeatureConfig(derivativeFeatureName);
    if (config == null) {
        LogUtil.error(log, format: "未获取到衍生特征配置，derivativeFeature=>{}", derivativeFeatureName);
        QMonitor.recordOne( name: Constants.DerivativeComputeRecord.DERIVATIVE_COMPUTE_PREFIX + "nullConfig.count");
        return;
    }
    // 2. 提交线程池开始处理流程
    derivativeFeatureProducePool.submit(() -> doProduce(derivativeFeatureName, config, params, readerType));
}

@Override
public void batchProduce(List<String> derivativeFeatureNames, Map<String, Object> params, ReaderType readerType) {
    derivativeFeatureNames.forEach(derivativeFeatureName -> produceOne(derivativeFeatureName, params, readerType));
}
```

初始的度量

自动识别其他类（实体）中的字段，并正确调用getter/setter

```java
/**
 * 深度优先遍历root节点，找到target节点，并返回该节点
 *
 * @param root   根节点
 * @param target 目标名称
 * @return   目标节点
 */
public Node dfs(Node root, String target) {
}
```

```java
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Node {

    private String name;

    private Node theLeftNode;

    private Node theRightNode;
}
```

```java
/**
 * 深度优先遍历root节点，找到target节点，并返回该节点
 *
 * @param root   根节点
 * @param target 目标名称
 * @return   目标节点
 */
public Node dfs(Node root, String target) {
    if (root == null) {
        return null;
    }
    if (root.getName().equals(target)) {
        return root;
    }
    Node left = dfs(root.getTheLeftNode(), target);
    if (left != null) {
        return left;
    }
    Node right = dfs(root.getTheRightNode(), target);
    if (right != null) {
        return right;
    }
    return null;
}
```

```java
/**
 * 深度优先遍历root节点，找到target节点，并返回该节点
 *
 * @param root   根节点
 * @param target 目标名称
 * @return   目标节点
 */
public Node dfs(Node root, String target) {
    if (root == null) {
        return null;
    }
    if (root.getName().equals(target)) {
        return root;
    }
    Node left = dfs(root.getTheLeftNode(), target);
    if (left != null) {
        return left;
    }
    Node right = dfs(root.getTheRightNode(), target);
    if (right != null) {
        return right;
    }
    return null;
}
```

问题解决的具体方案与结果

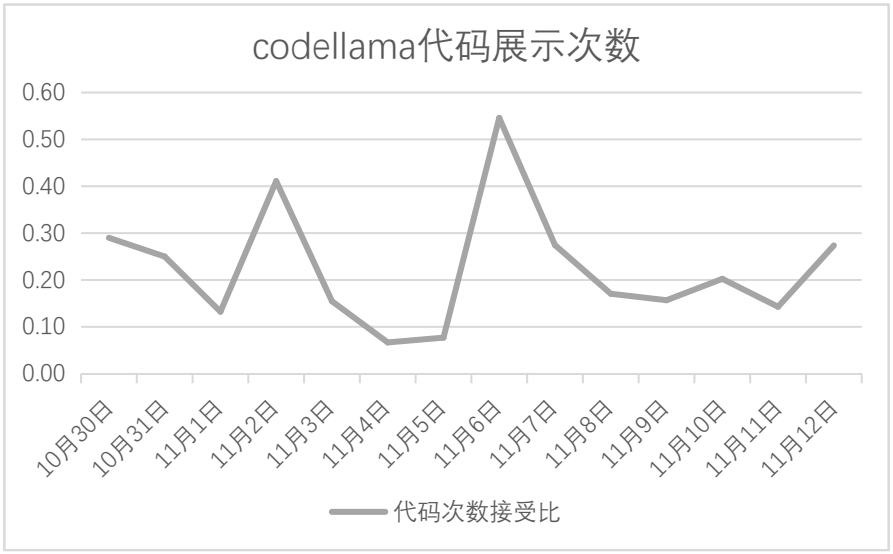
更换模型 codellama_7b -> codellama13b

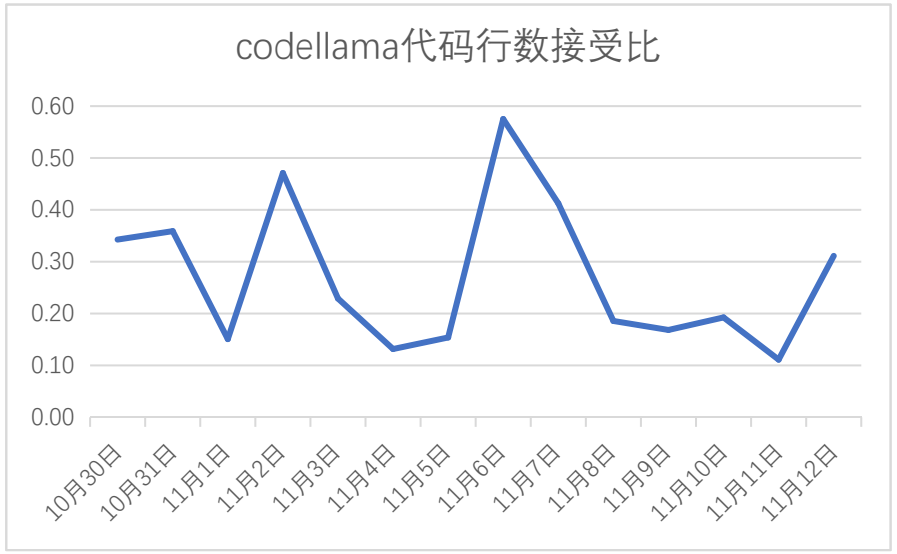
对返回值给与评分，设置展示阈值

做补全完整性校验

限制返回token数，提升处理速度

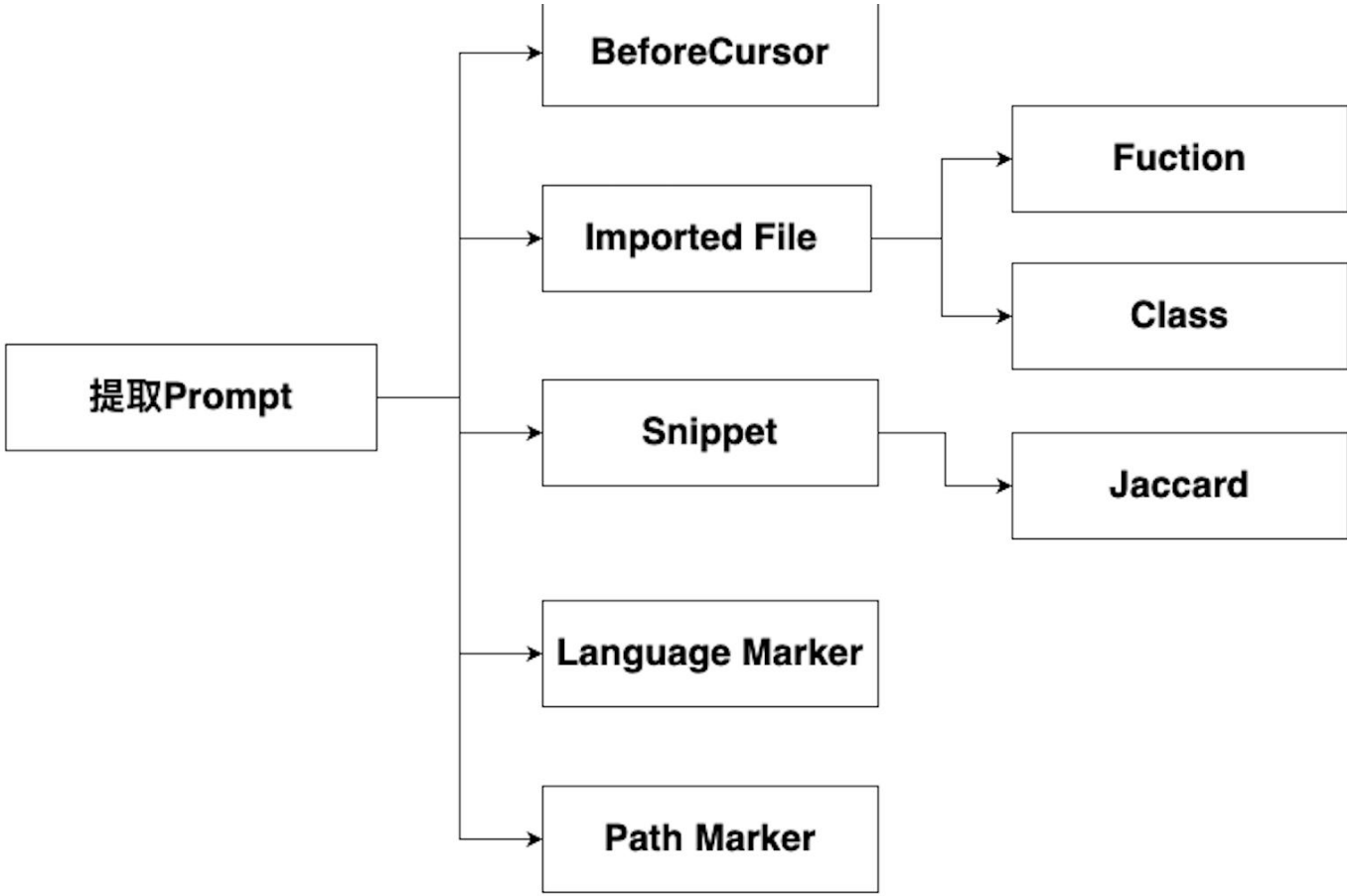
Codellama补全接受度 17% -> 25%





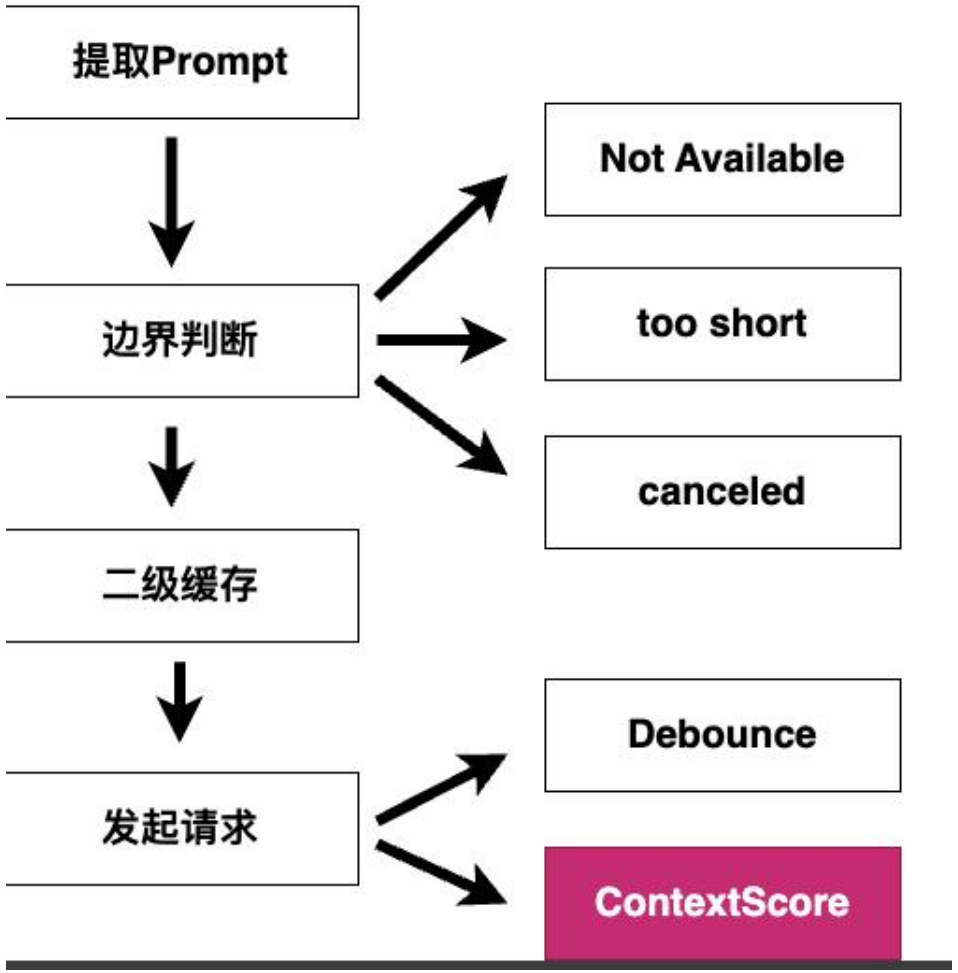
问题解决的具体方案与结果

提示词工程：



问题解决的具体方案与结果

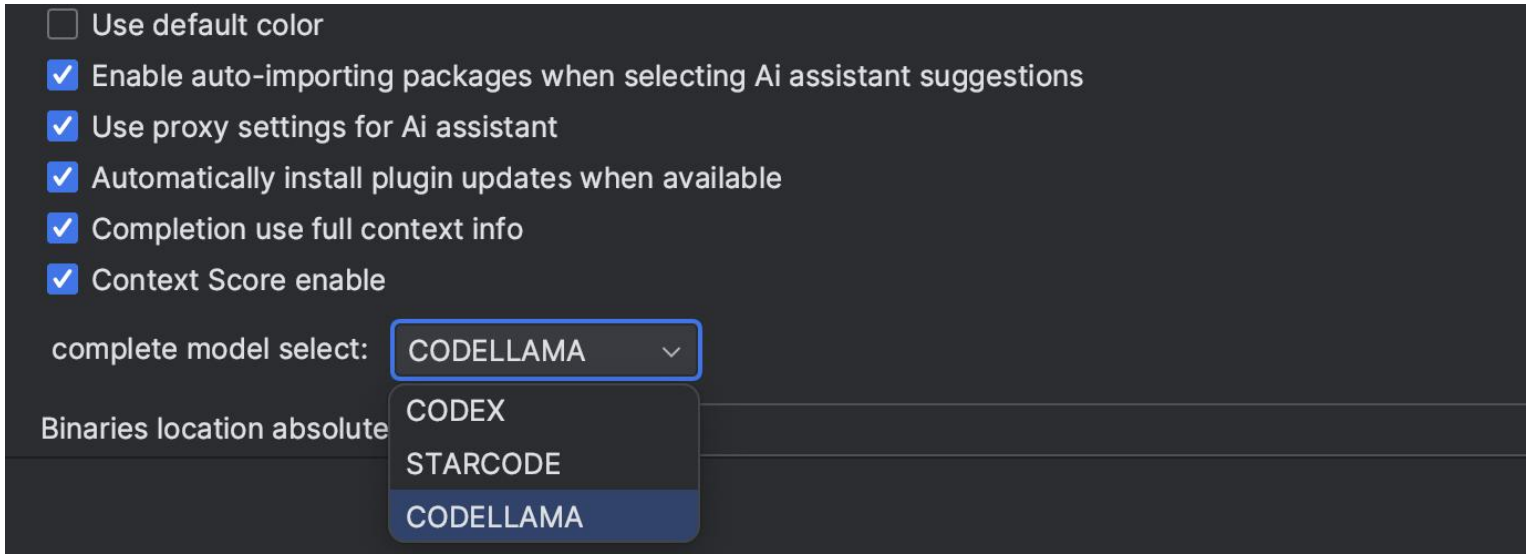
返回结果评估：



问题解决的具体方案与结果

与用户交互：

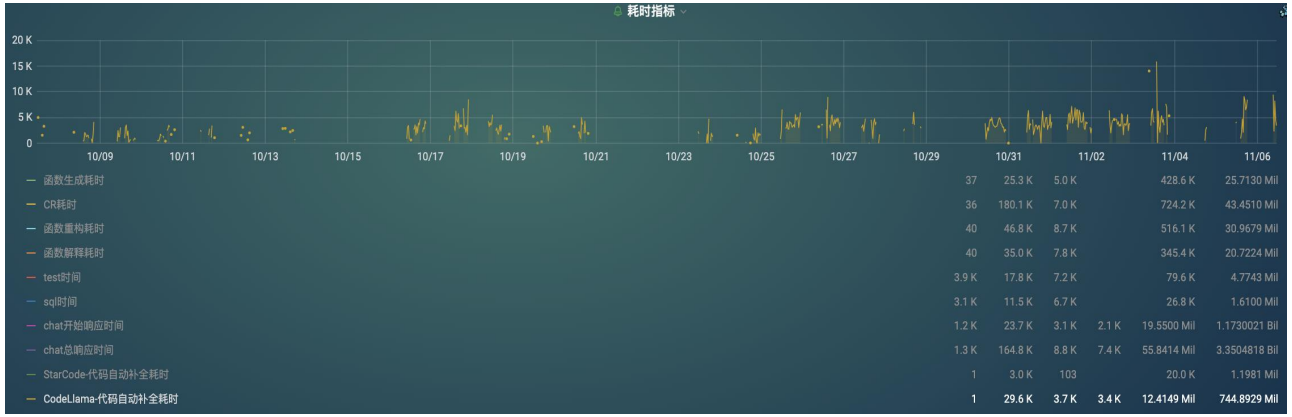
- 用户对于正在补全有感知
- 可自由开关代码补全功能
- 本地有缓存
- 用户可在一定范围内选择模型

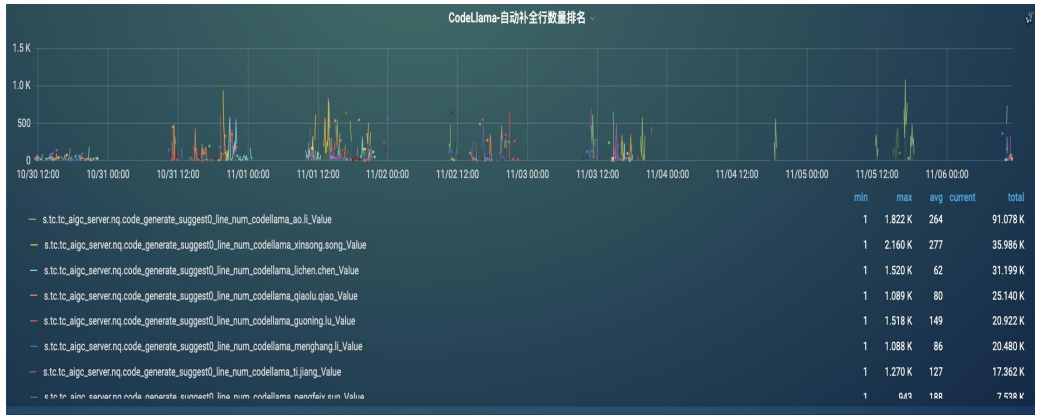


结果的度量

度量常态化：


0.4
0.3
0.2
0.1
0
10/30 12:00
10/31 00:00
10/31 12:00
11/01 00:00
11/01 12:00
11/02 00:00
11/02 12:00
11/03 00:00
11/03 12:00
11/04 00:00
11/04 12:00
11/05 00:00
11/05 12:00
异常总量
函数注释请求量
sql生成量
test生成量
chat失败量
chat成功量
StarCode-代码自动补全
StarCode-代码自动补全失败
StarCode-代码自动补全异常
StarCode-代码补全渲染展示次数
StarCode-代码补全接受次数
StarCode-代码补全丢弃次数
CodeLlama-代码自动补全
0 0.850 0.022 0.070


耗时指标
20 K
15 K
10 K
5 K
0
10/09
10/11
10/13
10/15
10/17
10/19
10/21
10/23
10/25
10/27
10/29
10/31
11/02
11/04
11/06
函数生成耗时 37 25.3 K 5.0 K 428.6 K 25.7130 Mil
CR耗时 36 180.1 K 7.0 K 724.2 K 43.4510 Mil
函数重构耗时 40 46.8 K 8.7 K 516.1 K 30.9679 Mil
函数解释耗时 40 35.0 K 7.8 K 345.4 K 20.7224 Mil
test时间 3.9 K 17.8 K 7.2 K 79.6 K 4.7743 Mil
sql时间 3.1 K 11.5 K 6.7 K 26.8 K 1.6100 Mil
chat开始响应时间 1.2 K 23.7 K 3.1 K 2.1 K 19.5500 Mil 1.1730021 Bil
chat总响应时间 1.3 K 164.8 K 8.8 K 7.4 K 55.8414 Mil 3.3504818 Bil
StarCode-代码自动补全耗时 1 3.0 K 103 20.0 K 1.1981 Mil
CodeLlama-代码自动补全耗时 1 29.6 K 3.7 K 3.4 K 12.4149 Mil 744.8929 Mil


CodeLlama-自动补全行数量排名
1.5 K
1.0 K
500
0
10/30 12:00
10/31 00:00
10/31 12:00
11/01 00:00
11/01 12:00
11/02 00:00
11/02 12:00
11/03 00:00
11/03 12:00
11/04 00:00
11/04 12:00
11/05 00:00
11/05 12:00
11/06 00:00
min max avg current total
s.tc.tc_aigc_server.nq.code_generate_suggest0_line_num_codellama_ao.li_Value 1 1.822 K 264 91.078 K
s.tc.tc_aigc_server.nq.code_generate_suggest0_line_num_codellama_xinsong.song_Value 1 2.160 K 277 35.986 K
s.tc.tc_aigc_server.nq.code_generate_suggest0_line_num_codellama_lichen.chen_Value 1 1.520 K 62 31.199 K
s.tc.tc_aigc_server.nq.code_generate_suggest0_line_num_codellama_qiaolu.qiao_Value 1 1.089 K 80 25.140 K
s.tc.tc_aigc_server.nq.code_generate_suggest0_line_num_codellama_guoning.lu_Value 1 1.518 K 149 20.922 K
s.tc.tc_aigc_server.nq.code_generate_suggest0_line_num_codellama_menghang.li_Value 1 1.088 K 86 20.480 K
s.tc.tc_aigc_server.nq.code_generate_suggest0_line_num_codellama_ti.jiang_Value 1 1.270 K 127 17.362 K


自动补全监控
1.5 K
1.0 K
500
0
10/30 12:00
10/31 00:00
10/31 12:00
11/01 00:00
11/01 12:00
11/02 00:00
11/02 12:00
11/03 00:00
11/03 12:00
11/04 00:00
11/04 12:00
11/05 00:00
11/05 12:00
11/06 00:00
Replit-总生成行数
StarCode-总接受行数
StarCode-总展示行数
StarCode-总生成行数
CodeLlama-总接受行数 1 128 8 1.576 K
CodeLlama-总展示行数 1 128 6 4.779 K
CodeLlama-总生成行数 1 2.168 K 166 285.193 K
CodeLlama-总生成行数 1 2.168 K 166 285.193 K

PART 05
后续工作展望

后续工作展望

模型基座：更便宜的SFT(QLora?)、更精炼的模型

模型部署：更小的量化损失

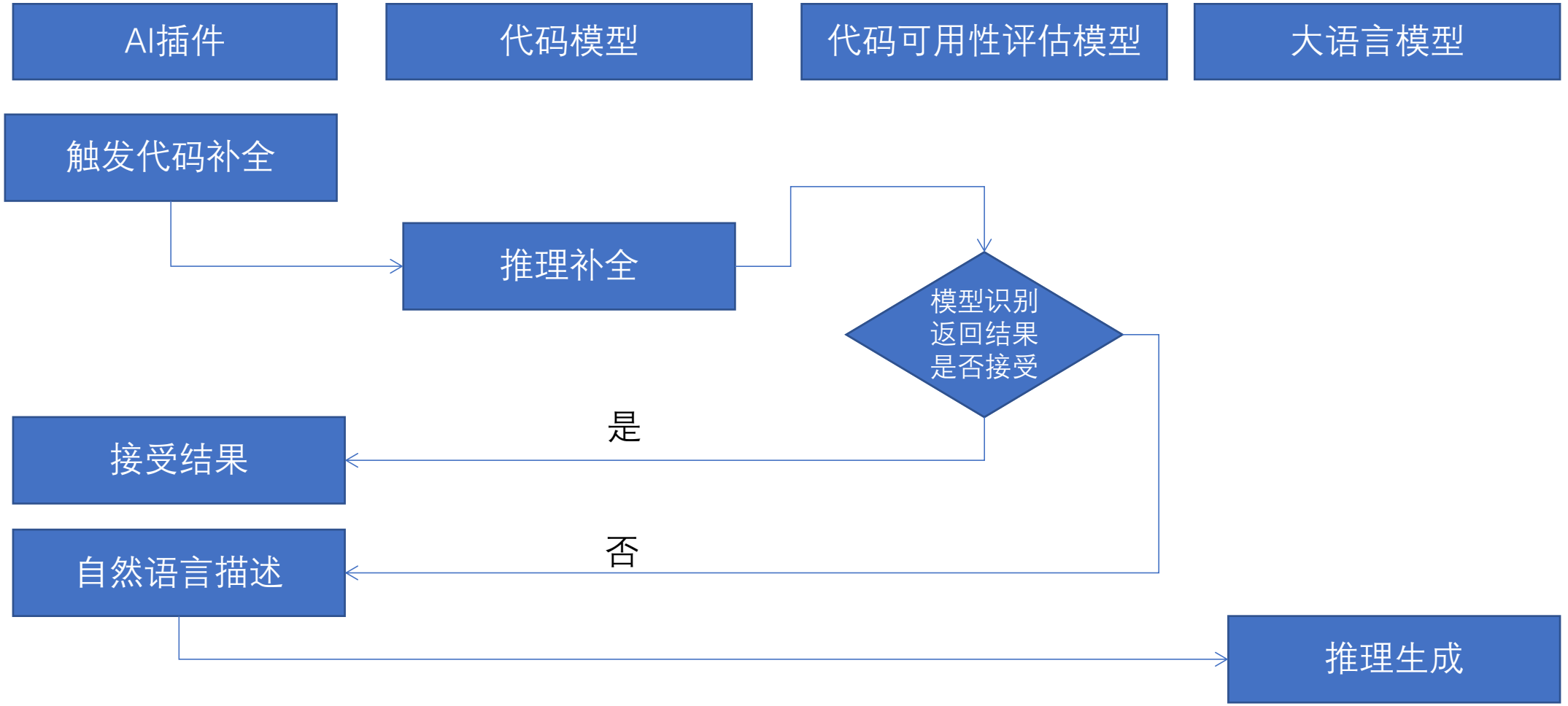
提示词工程：结合Agent框架，代码生成工程化、SOP化

返回结果评估：更迅速、更全面

与用户交互：IDE集成能力更强，更少的跳出诉求，更友好的提示用户完善注释

结果如何度量：效果度量标准更加标准化，权威性更强

后续工作展望-直接补全与自然语言生成结合



AiDD AI+ 软件研发数字峰会
AI+ software Development Digital summit

THANKS