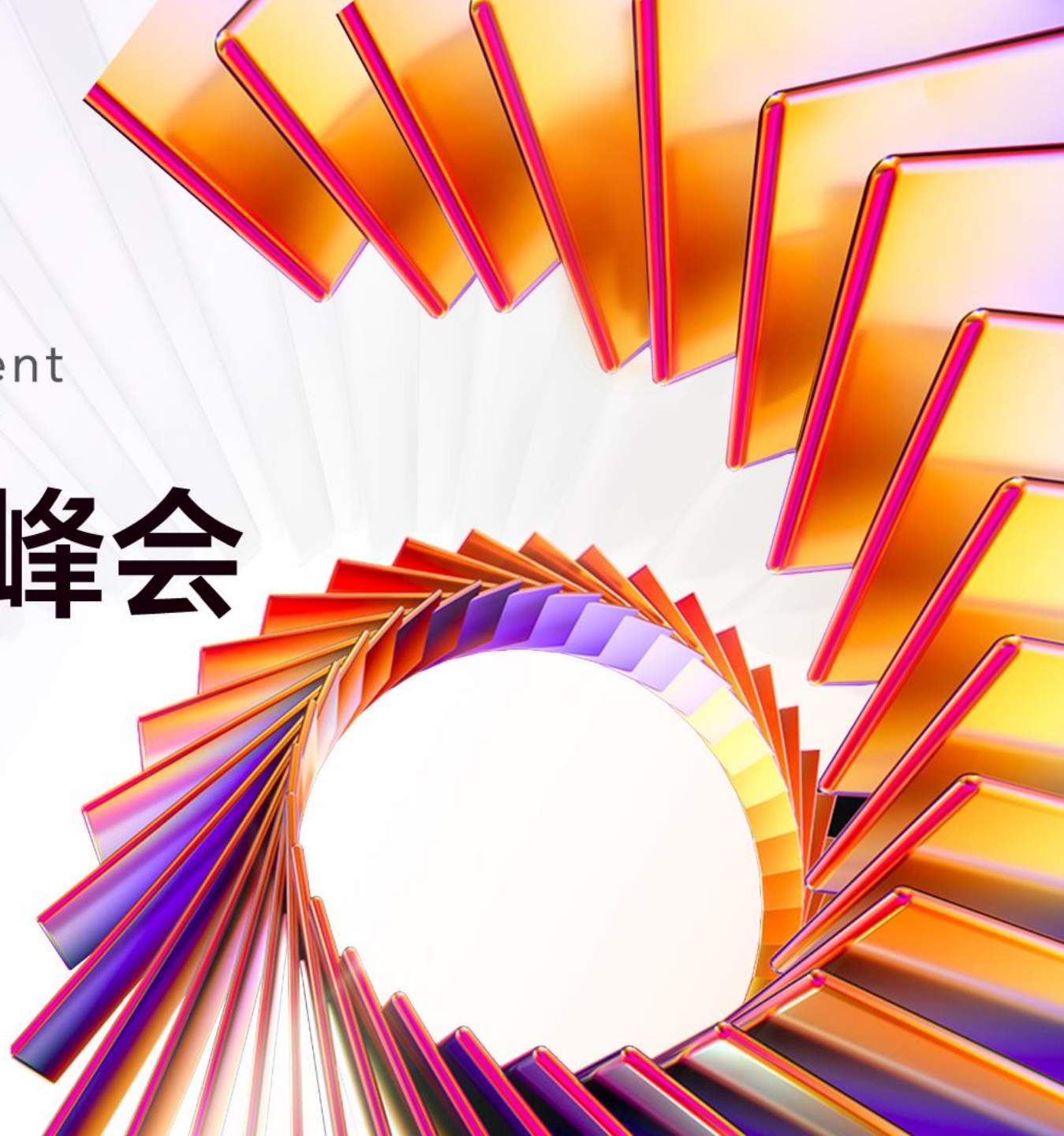
AiDD 8th 2025

第8届 AI+ Development Digital Summit AI+研发数字峰会

拥抱AI 重塑研发

11月14-15日 | 深圳



AiDD × K

2026 AI+ PRODUCT INNOVATION SUMMIT

01.16-17 · ShangHai

AI+产品创新峰会

Track 1: AI 产品战略与创新设计

从0到1的AI原生产品构建

论坛1: AI时代的用户洞家与需求发现

论坛2: AI原生产品战路与商业模式重构

论坛3: AgenticAI产品创新与交互设计

Track 2: AI 产品开发与工程实践

从1到10的工程化落地实践

论坛1: 面向Agent智能体的产品开发

论坛2: 具身智能与AI硬件产品

论坛3: AI产品出海与本地化开发

Track 3: AI 产品运营与智能演化

从10到100的AI产品运营

论坛1: AI赋能产品运营与增长黑客

论坛2: AI产品的数据飞轮与智能演化

论坛3: 行业爆款AI产品案例拆解

2-hour Speech: 回归本质

用户洞察的第一性

——2小时思维与方法论工作坊

在数字爆炸、AI迅速发展的时代，仍然考验“看见”的“同理心”

Panel 1: 出海前瞻

“出海避坑地图”圆桌对话

——不止于翻译:

AI时代的出海新范式

Panel 2: 失败复盘

为什么很多AI产品“叫好不叫座”?

——从伪需求到真价值:

AI产品商业化落地的关键挑战

智能重构产品 数据驱动增长

Reinventing Products with Intelligence, Driven by Data

80+ 业界大咖 汇聚产品大咖的真知灼见

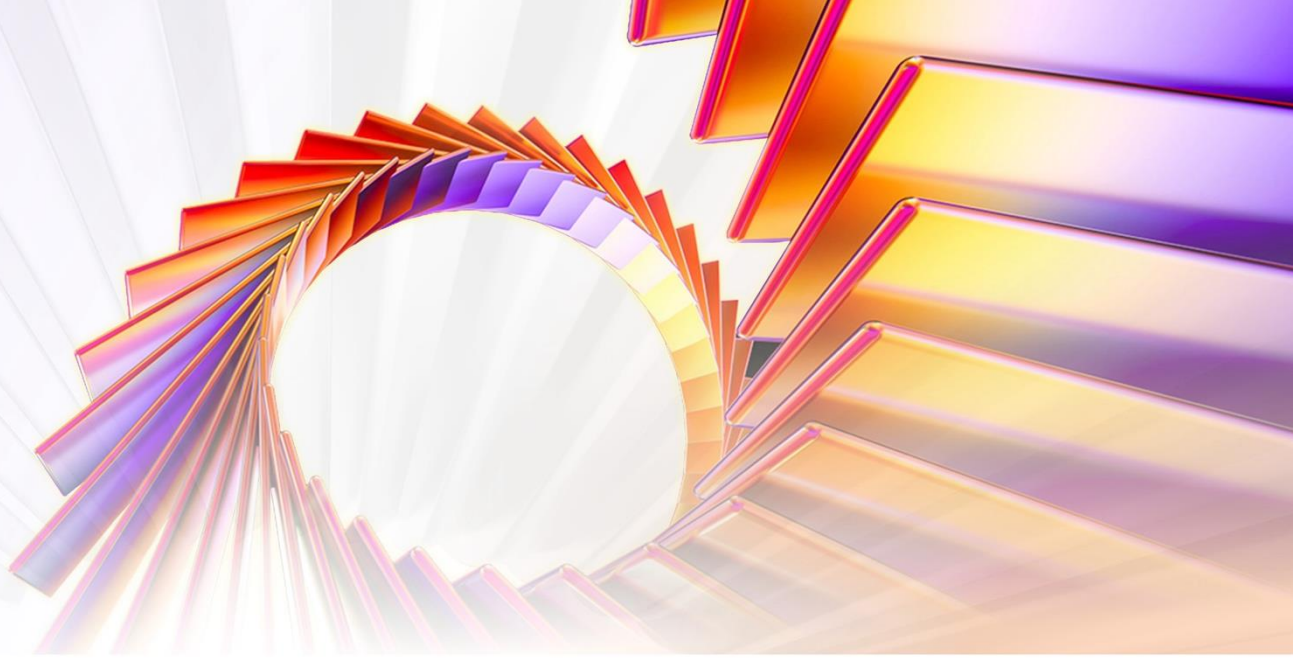
40+ 创新案例 开拓全新AI+产品视角

10+ 分论坛 涵盖产品到商业增长的全链路

800+ 听众规模 共同探索产品的智能重构

扫码查看会议详情

AiDD 8th 2025 | 11月14-15日 | 深圳
第8届 AI+ Development Digital Summit
AI+研发数字峰会
拥抱AI 重塑研发



从数据治理到基于知识图谱脚本生成探索之路

华凯建 | oppo

华凯建

oppo高级软件测试开发工程师

多年测试开发工作经验，专注于利用ai技术推动测试创新与实践。在AI助力动效智能测试，测试框架及流水线构建，缺陷分析等领域有深入研究。

目录
CONTENTS

问题引入

AIGC测试脚本生成：为何陷入“高开低走”困境？

理想

一键生成
输出即为高质量脚本

解放人力
转向高附加值的工作

工程化落地
成为可靠的自动化的环节

现实

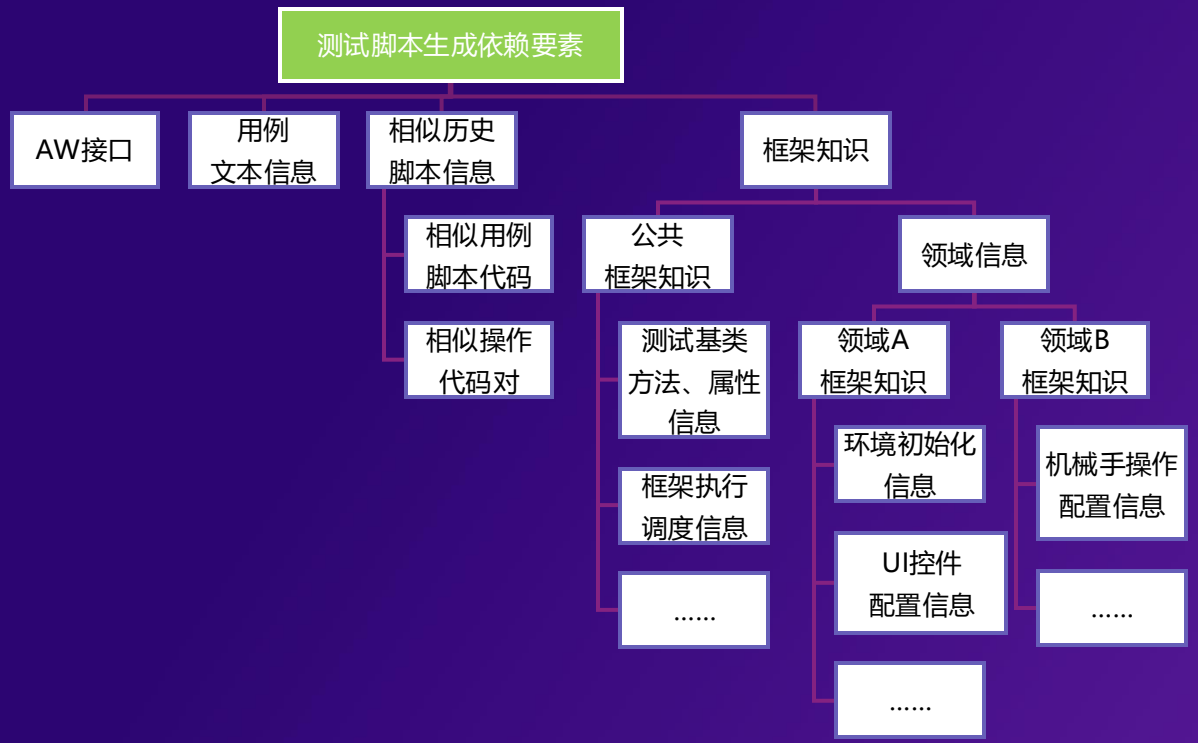
脚本质量不稳定
生成结果时好时坏，难以信任

调试成本高昂
修改脚本的时间甚至超过手动编写

提效难以衡量
投入大量资源，提效不明显

技术已就绪，但实践已证明，直接将原始数据喂给大模型，难以稳定地生成符合工程要求的测试脚本。

追根溯源：四大因素，导致生成脚本质量差



测试AW 质量堪忧

1.问题：注释缺失、功能边界模糊。“脆弱”的适配性。

2.影响：生成的脚本从根基上就是坏的。

文本用例“噪音”充斥

1.问题：错别字、专业词汇滥用。需求描述存在二义性、操作步骤与预期结果不匹配。

2.影响：ai无法理解需求，生成的脚本只能依靠瞎猜。

脚本与用例严重脱节

1.问题：脚本的实际验证点，与用例测试期望不相符。

2.影响：AI无法从历史中的脚本信息，学习如何正确生成新脚本。

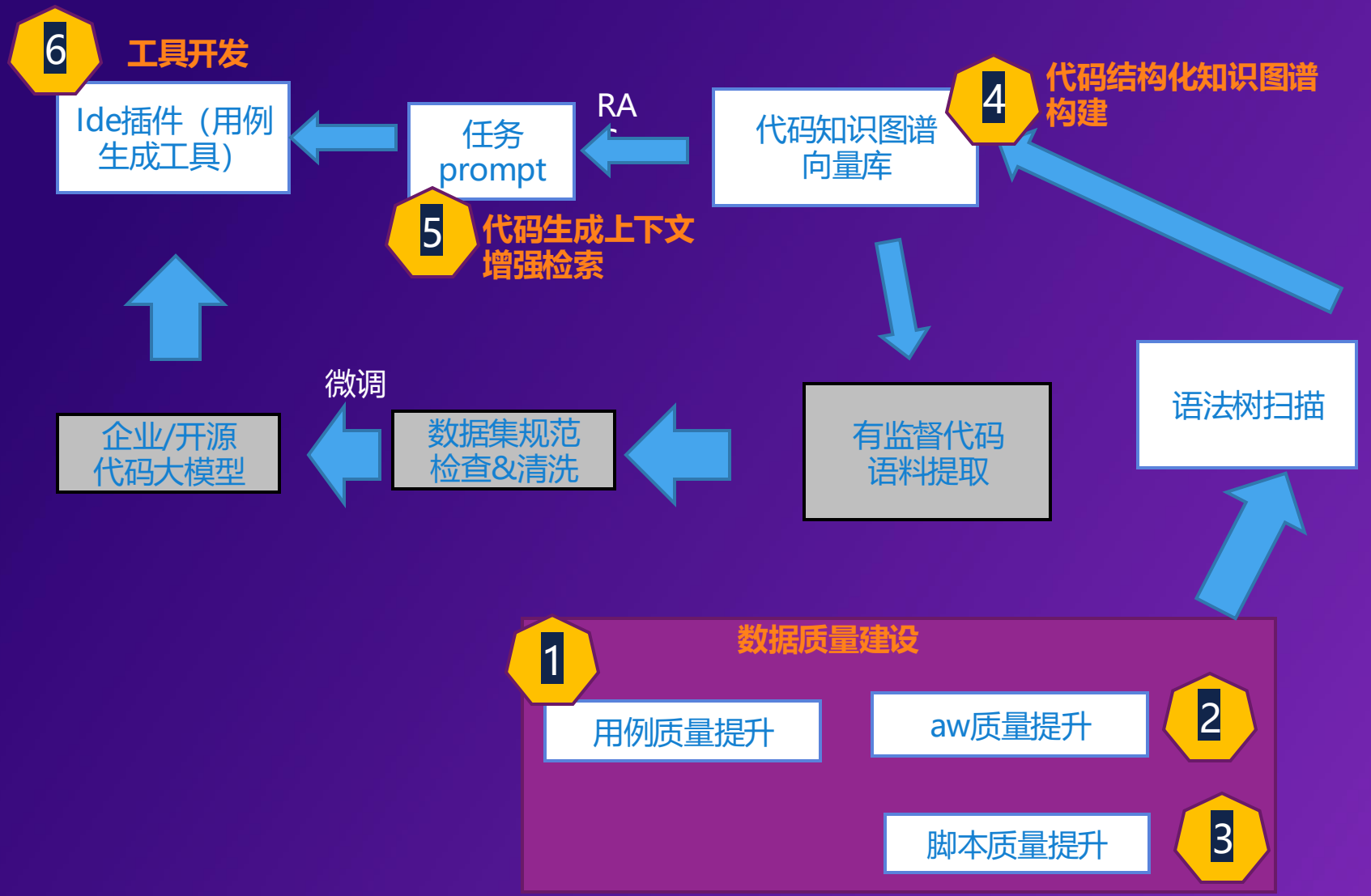
框架知识未被有效利用

1.问题：由于缺乏对代码结构和调用关系的理解，模型只能进行浅层模仿。

2.影响：ai无法感知深层次的知识来生成脚本。

未经治理的数据，AI在生成脚本无法理解内在逻辑与规范，从而产出不可靠的脚本。

破局之道：构建从“数据治理”到“智能生成”的高质量闭环

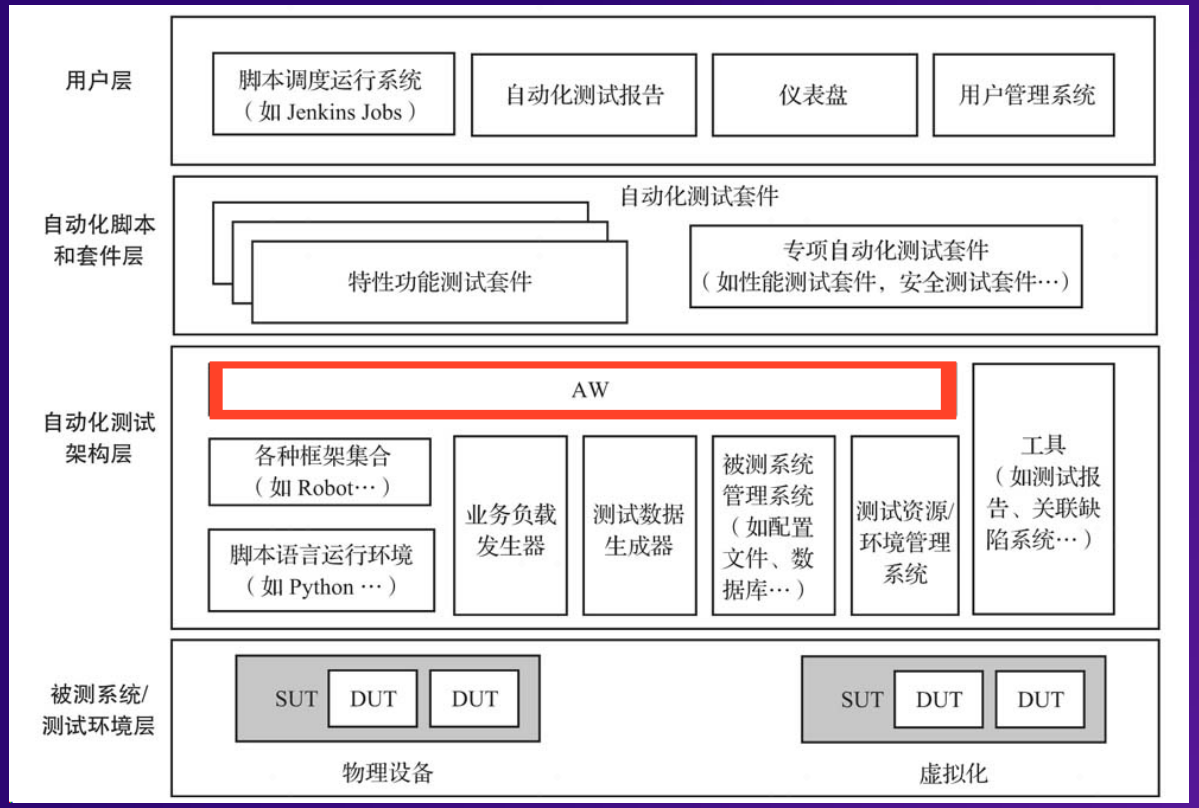


AiDD 8th 2025

PART 01

测试aw治理

AW质量与AIGC脚本生成的“垃圾进与垃圾出”



图片摘自：刘琛梅 .测试架构师修炼之道[M]机械教育初版社,2021:12.

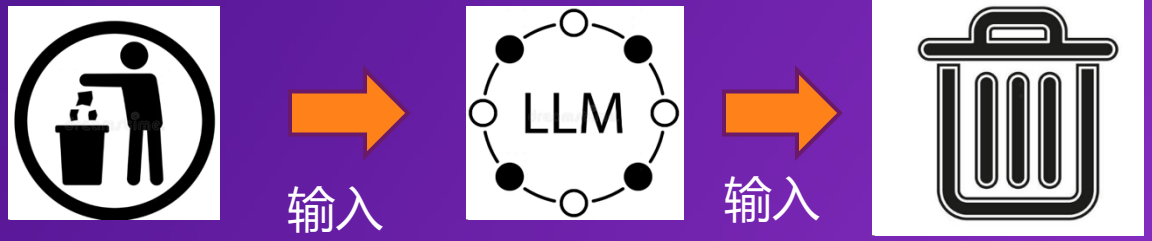
AW作为脚本直接调用的、有意义的测试接口。是脚本构成的基石。

AW注释主要问题：

- 1.注释缺失
- 2.参数类型不正确
- 3.表述模糊不清晰
- 4.专业词汇不规范
- 5.功能标注不清晰
- 6.风格不一

AW代码主要问题：

- 1.功能未实现
- 2.功能不健壮
- 3.功能逻辑错误
- 4.性能低下
- 5.安全性缺陷
- 6.集成/兼容性问题



不规范的AW　　大模型　　脚本

AW治理：设计大模型背景下的代码注释规范

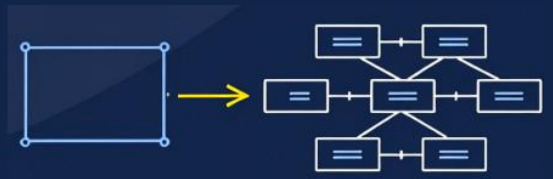
从“为人而注”到“为人与ai共读”的范式转变

大模型通过代码理解aw的挑战

为何代码本身难以让大模型理解?

复杂的依赖链

函数内部调用其他函数，形成依赖网



Token长度限制

数百/数千行的函数实现无法完整放入上下文窗口

解决方案：通过注释理解，面向Ai的规范注释

高质量注释是最高效的“Prompt”

函数作用

简要描述函数的功能预用法

参数说明

明确入参/出参的类型、含义与约束

使用限制

指明使用场景、前置条件

异常处理

说明可能抛出的异常及含义。

示例驱动

提供1-2个典型用法示例，作为Few-shot Learning。

优秀的注释，是人与AI之间最可靠的契约。它不仅是文档，更是嵌入代码的‘系统Prompt’，直接决定了模型的理解效能与代码生成质量。

AW治理：AIGC辅助代码注释的规模化落地实践

从大量存量aw到增量aw的标准化注释全流程落地

规模化注释的困境

✨**数量庞大**

大量AW函数需要标注

✨**人工成本极高**

纯人工方式耗时耗力，效率低下

✨**一致性难以保证**

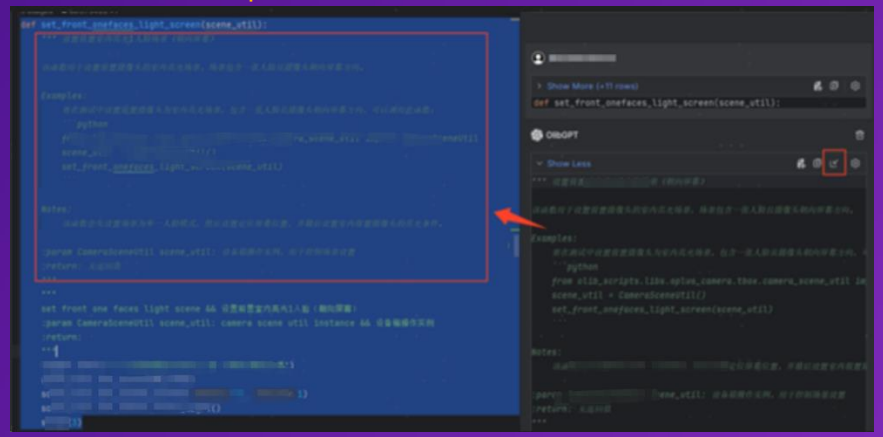
不同人员标注风格不一，质量参差

AI时代注释标准建设 → Prompt设计 → 开发工具预标注 → 人工校验 → Ci校验合入

人工审核修正

ci校验存量/新增aw是否标准化注释


```
你是测试开发工程师，现在对不规范的代码注释重新生成注释文档。
### 注释规范
xxxxxx
###  相关知识
xxxxxx
### 注释示例
#### 示例1输入
xxxxxx
#### 示例1输出
xxxxxx
.....

### 输入任务
xxxxx
### 输出要求
现在我向你输入任务，"相关知识"中的内容对生成注释也许有帮助，
请严格按照"注释规范"要求和"注释示例"来输出中文注释。
只输出注释文档，不需要输出其他内容。
```



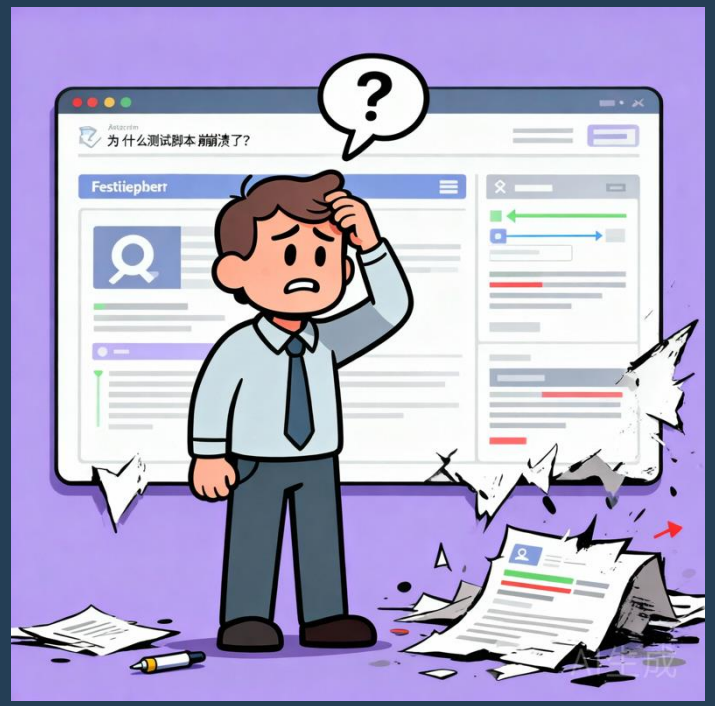
基于大模型代码注释，生成的代码注释质量较高，人工评估可用性在95%以上。

AW健壮性：基于大模型增强ui变更快速适配

UI变更致使用例失败的困境

✨传统解法方案的瓶颈

•依赖固定的控件属性（ID、文本）进行定位。
•一旦属性变更，自动化脚本立即失败，需要人工排查和修复。



解决方案：大模型驱动的智能定位与适配

✨核心思想

当传统定位方式失效时，将UI的“视觉语义”与“代码结构”交由大模型理解，动态生成兼容性定位策略

✨技术流程

1.捕获现场： aw执行找不到操作元素时，捕获当前页面的信息（如XML）。
2.信息裁减：对xml文本进新裁减，只保留关键信息。例如：{‘text’：‘20:41’，‘desc’：‘20:41’，‘class’：‘TextView’，‘center’：‘(150,75)’}‘。
3.智能匹配： 大模型理解UI语义，找到最匹配目标控件的新定位方式。
4.自愈执行： 执行模型返回的新定位策略，完成测试操作。

✨案例（手机登录）

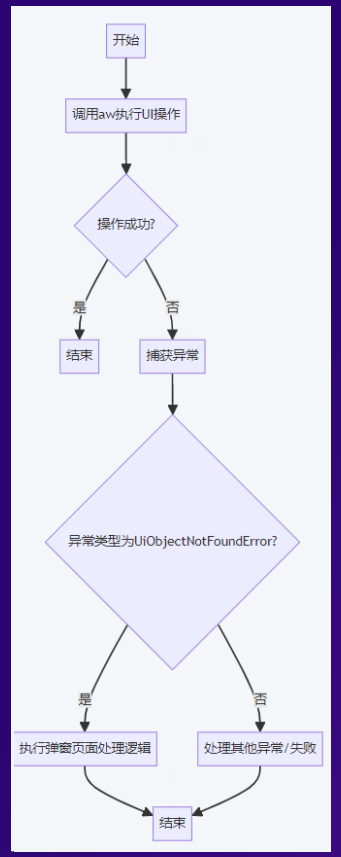
```
from olib_aw.teams.ai.ai_api import call_simple_agent

xml = """<?xml version='1.0' encoding='UTF-8' standalone='yes' ?><hierarchy rotation="0"><node index="
task_content = f"""
请根据我提供的手机账号登录页面xml信息，分析出账号输入框位置，密码输入框位置，确认按钮输位置。
xml信息如下:
{xml}
"""
tips = ""
output_format_dict = {
    "account": "[x1,y1][x2,y2]:账号输入框位置,其中x1,y1是边框左上角坐标，x2,y2是边框右下角坐标,只输出位置",
    "password": "[x1,y1][x2,y2]:密码输入框位置,其中x1,y1是边框左上角坐标，x2,y2是边框右下角坐标,只输出位置",
    "button": "[x1,y1][x2,y2]:确认按钮位置,其中x1,y1是边框左上角坐标，x2,y2是边框右下角坐标,只输出位置"
}

data = call_simple_agent(task_content=task_content, tips="", output_format_dict=output_format_dict)
```

```
account
[297,653][984,808]

password
[96,857][984,1012]

button
[96,1250][984,1382]
```

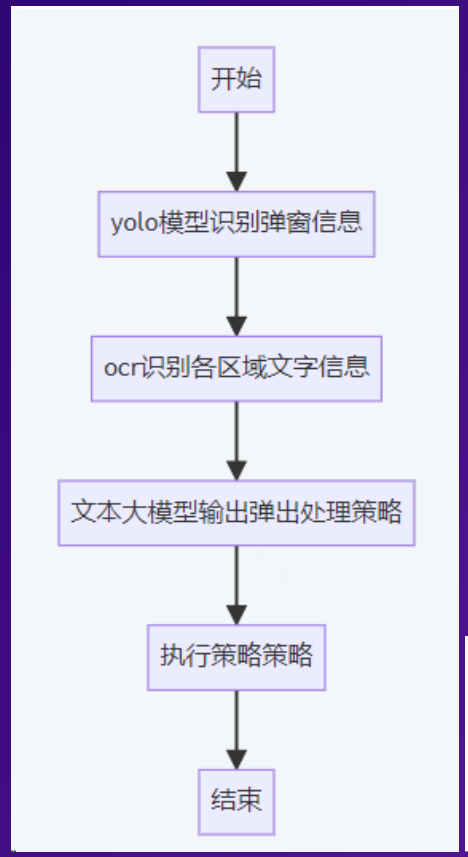
因ui变更的导致的登录问题下降80%。

AW健壮性：基于大模型ui操作弹窗问题处理

从”yolo+ocr+文本大模型”到”视觉大模型”智能体的方案演进



弹窗问题处理触发时机




```
弹窗内的文字：广告,天猫双11，9.9元，超级大优惠，呀！土豆，立即查看
操作按钮：[{'button_area': [135.07762145996094, 647.7536010742188, 295.
7921447753906, 700.2354736328125], 'text_info': {'text': '立即查看'}}]
弹窗关闭按钮：  [{'button_area': [135.07762145996094, 647.7536010742188,
295.7921447753906, 700.2354736328125]}]
```

```
### 弹窗类型
广告
### 弹窗处理策略
由于弹窗属于广告类型，建议点击关闭按钮以消除干扰
### 点击坐标
(215,674)
```

yolo + ocr +文本大模型方案

GUI VL Agent

```
#### 是否存在弹窗
是

#### 弹窗类型
广告/推广

#### 处理弹窗动作决策说明
这是一个典型的广告弹窗，目的是吸引用户注册并领取优惠券。它不属

#### 点击坐标
(590, 285)
```

视觉大模型方案（gui-owl-32b，qwen-vl）

处理准确率： 75% → 92%；泛化能力： 一般 → 极强

从“看见文字”到”看懂场景“，实现根本性的健壮性跨越。随着视觉大模型推理成本不断下降，后续优势会愈加明显。

PART 02

用例质量提升

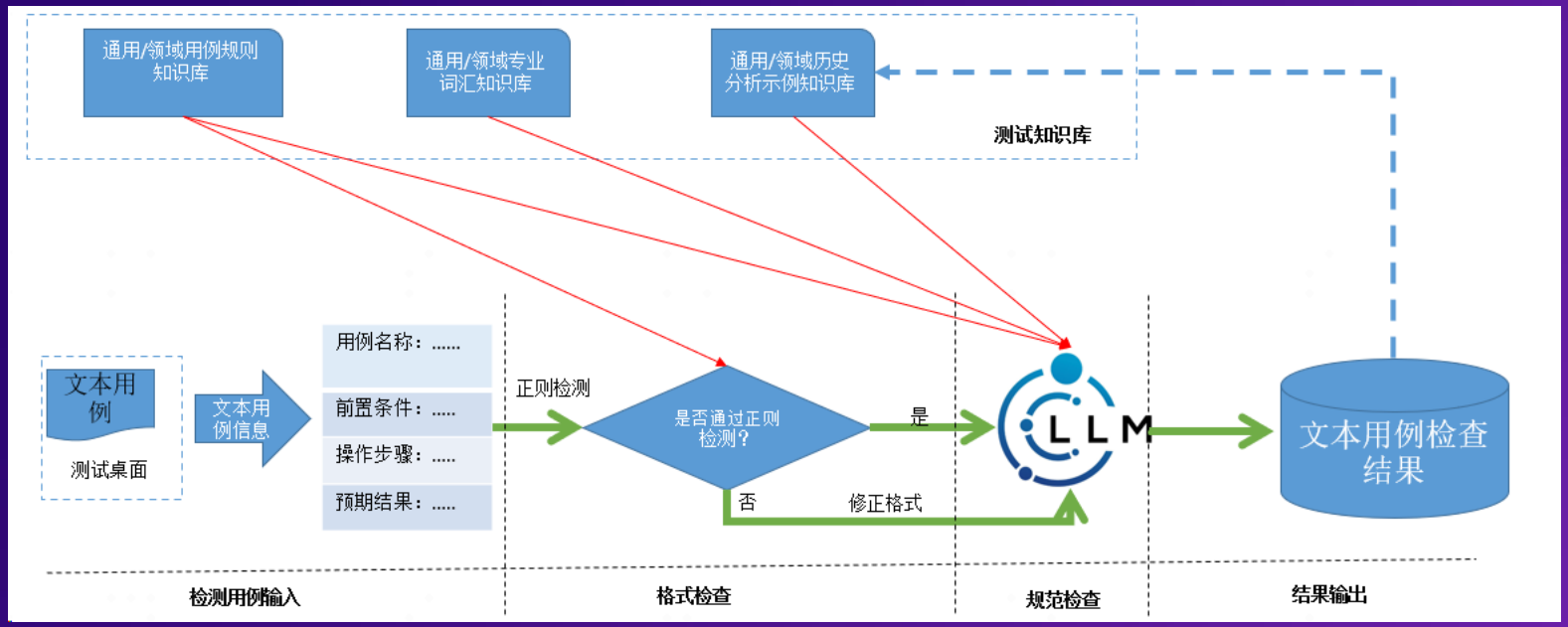
用例质量提升：现阶段哪些问题可以由大模型来检测？

大模型检查用例问题，可以细分为以下几类：

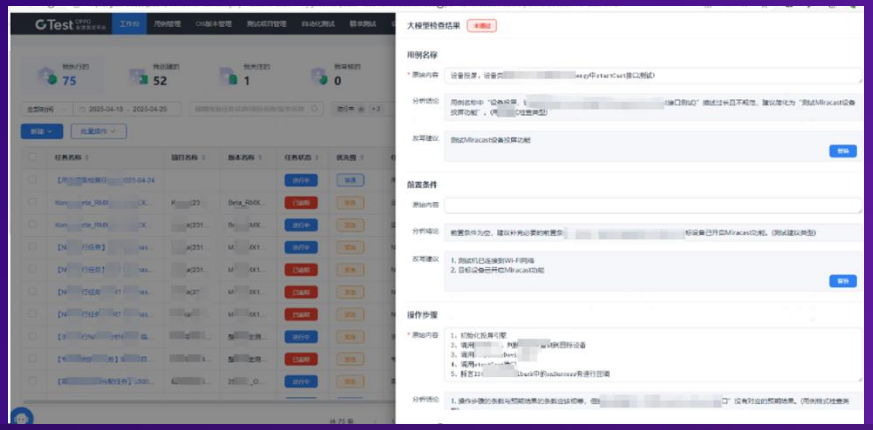
检查项	例子	方案
1.用例格式类	分点描述统一用数字序号“1.” “2.” “3.”。	正则处理+用例规范
2.错别字	/	大模型语义理解自动校正
3.二义性	模糊描述如“多次尝试”	大模型+用例规范
4.步骤与期望对应	操作步骤与预期结果逻辑关联	大模型+用例规范
5.独立性	用例无相互依赖	大模型+用例规范
6.用词专业性	统一使用“测试机”等术语	大模型+领域专业词汇知识库+用例规范
7.领域自定义规则检查	/	大模型+领域规范
8.规范建议类型	/	大模型+用例规范

用例质量提升：技术方案实施与落地

整体技术方案



用例检测 嵌入到工作流程中（用例检查任务/用例提交评审节点中）



67%

识别完全正确无的用例数（无错检，漏检）/ 总用例数

25%

识别不完全正确用例数/ 总用例数

AiDD 8th 2025

PART 03

代码结构化知识图谱构建

结构化代码知识图谱构建的优势

结构化代码知识图谱RAG和普通RAG的对比

维度	普通RAG	结构化代码知识图谱RAG
信息表示方式	向量化文本数据，依赖文本相似性	图结构（节点（实体），边（关系）），直观呈现代码关联
信息检索方式	向量相似度搜索，简单匹配有效	图遍历和子图检索，支持复杂查询和多层次关系追踪
关系理解能力	有限，难以捕捉复杂交互和层次结构	强大，可推理实体间关系，提供丰富上下文

代码结构化知识图谱优势

01

灵活查询

直接通过图结构查询函数/模块，无需遍历文本，提升效率

例如查找“函数A调用的所有子函数”

02

深入关系挖掘

可以挖掘和理解不同函数之间的调用关系，包括间接调用和递归关系，有助于快速掌握整个代码结构。

例如查找测试脚本的aw调用链

代码结构化知识图谱实体和关系设计

实体设计

实体	属性	备注
Repository	id,name	仓库名
File	id,name	文件（模块）名
Import	id,name	导包
Class	id,name,base_class	类
NestedClass	id, name	类中嵌套类
AWFunction	id, name, code, parameters, Docstring, aw_descriptio (embedding)	独立函数（aw）
AWMethod	id, name, code, parameters, docstring aw_descriptio (embedding)	类中方法(aw)
Script	id，name, code	测试脚本
CodePair	id，name(embedding)	步骤操作代码对

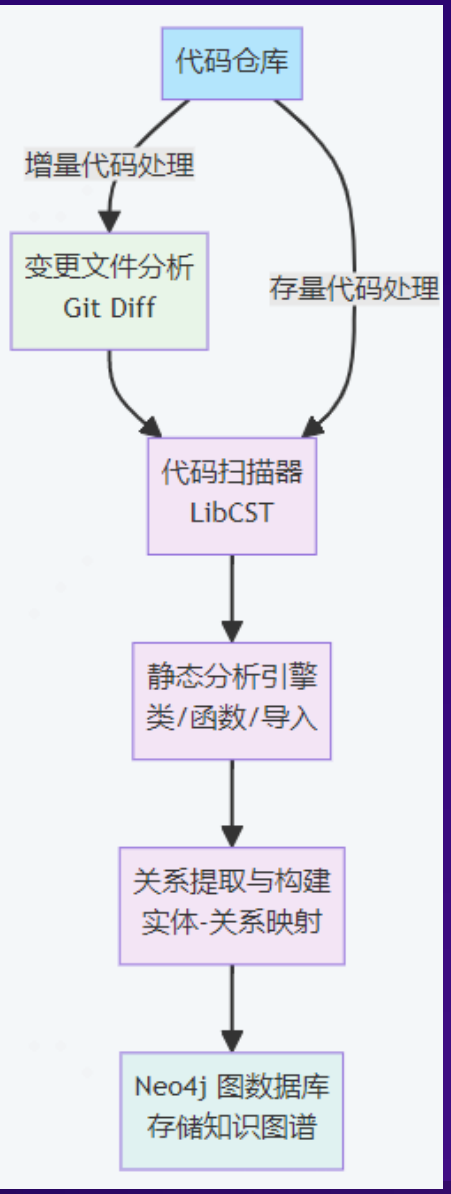
实体和关系，可以根据实际情况灵活设计。

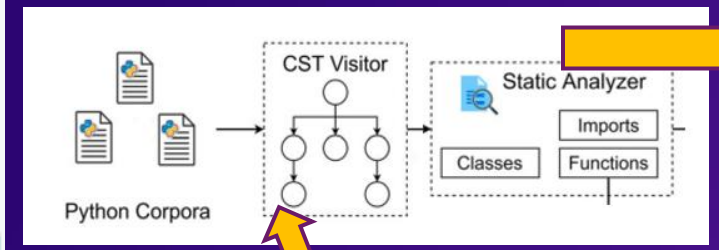
关系设计

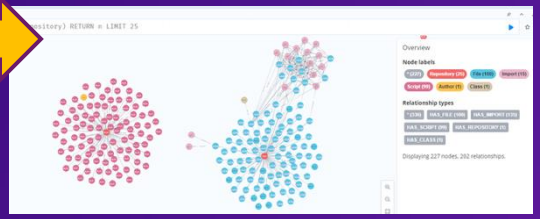
关系	主实体	辅实体	说明
HasModule	Repository	File	仓库中有哪些模块（文件）
HasImport	File/Script	Import	模块/脚本中有哪些导包
HasClass	File	Class	模块中有哪些类
HasNestedClass	Class	NestedClass	类中有哪些嵌套类
HasAwFunction	File	AwFunction	模块中有哪些独立的函数
HasAwMethod	Class	AWMethod	类中有哪些方法
HasScripts	Repository	Script	仓库中包含哪些脚本
HasCodepair	Script	Codepair	脚本中包含哪些代码对
UseAwMethod	Script	AWMethod	脚本中使用了哪些方法aw
UseAwFunction	Script	AWFunction	脚本中包含哪些函数aw
……	……	……	……

方案参考自： Lu Liang, Yong Li, Ming Wen & Ying Liu (2022) KG4Py: A toolkit for generating Python knowledge graph and code semantic search, Connection Science, 34:1, 1384-1400, DOI: 10.1080/09540091.2022.2072471

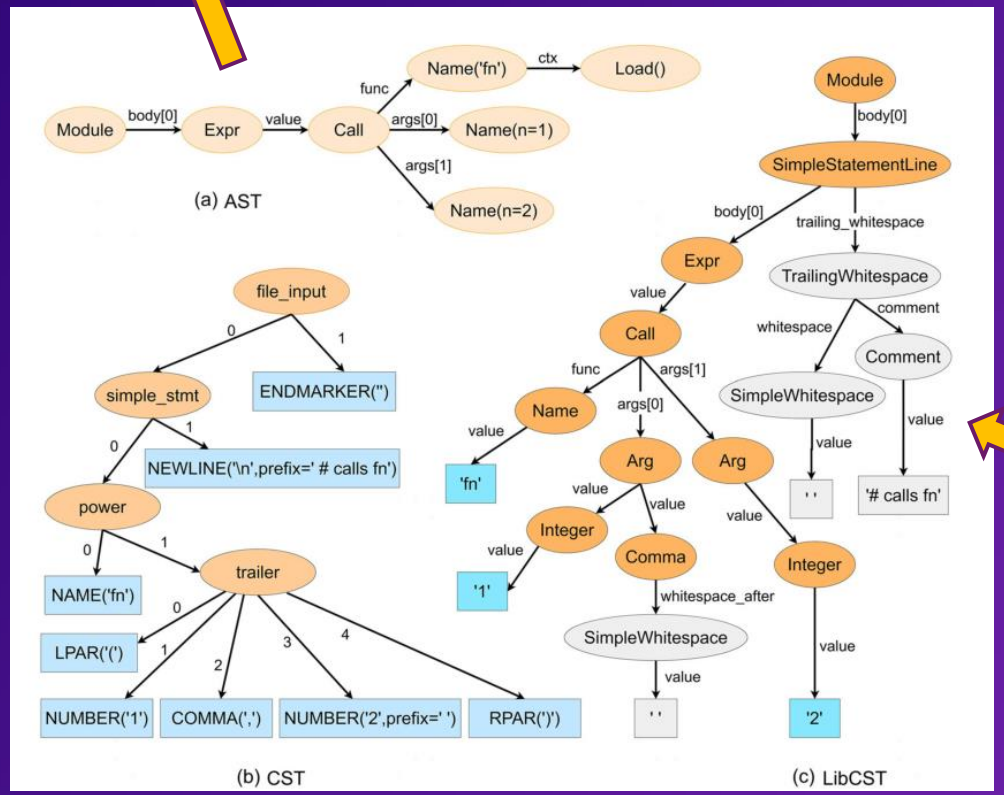
从代码中提取结构构建知识图谱







数据存储到Neo4j知识图谱数据库中



结构化代码知识提取核心注意点

问题	影响	最佳实践
隐式导入（如 import *）	模块来源模糊，	使用显式导入
动态反射（如 getattr()）	静态分析失效	避免动态调用，优先使用显式结构

项目	特点
AST（抽象语法树）	表示语义结构，忽略语法细节
CST（具体语法树）	完整保留所有语法符号
LibCST	Python库，高效处理CST

✅ 推荐使用 LibCST 进行代码扫描，提取节点信息

PART 04

脚本质量提升

脚本用例一致性检测的必要性与挑战

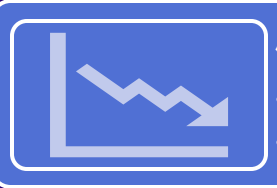
必要性

- **用例描述（我们想要的）：**
  - **前置条件：** 用户已登录且账户余额充足。
  - **操作步骤：** 购买价值100元的商品，使用余额支付。
  - **预期结果：** 支付成功，用户余额减少100元，生成正确的订单记录。

```python
def test_balance_payment():
    login()  # 登录了，但未检查余额是否充足
    purchase(100)
    select_payment("balance")
    assert payment_success()  # 支付成功
    # 遗漏：没有检查余额变化！
    # 遗漏：没有验证订单金额！
```

风险一：测试覆盖度“虚假繁荣”

- 自动化测试报告一片绿色，实际大量关键业务逻辑未被验证，质量风险被掩盖。



风险二：缺陷检测能力大幅衰减

- 脚本没有检查到的部分，一旦代码变更引入缺陷，无法被及时发现。
- 回归测试形同虚设，缺陷泄漏到生产环境的概率激增。

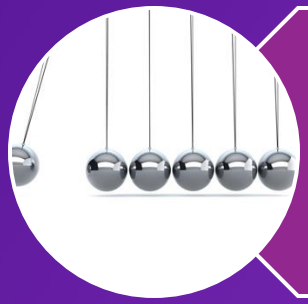
风险三：维护成本与信任危机

- 随着时间推移，用例与脚本的差异越来越大，测试套件逐渐“腐化”。
- 团队对自动化测试结果失去信任，最终又倒退到低效的手工测试。

挑战

深度调用链追溯

一个简单操作背后是复杂的函数调用网，必须深度分析才能发现断言点。



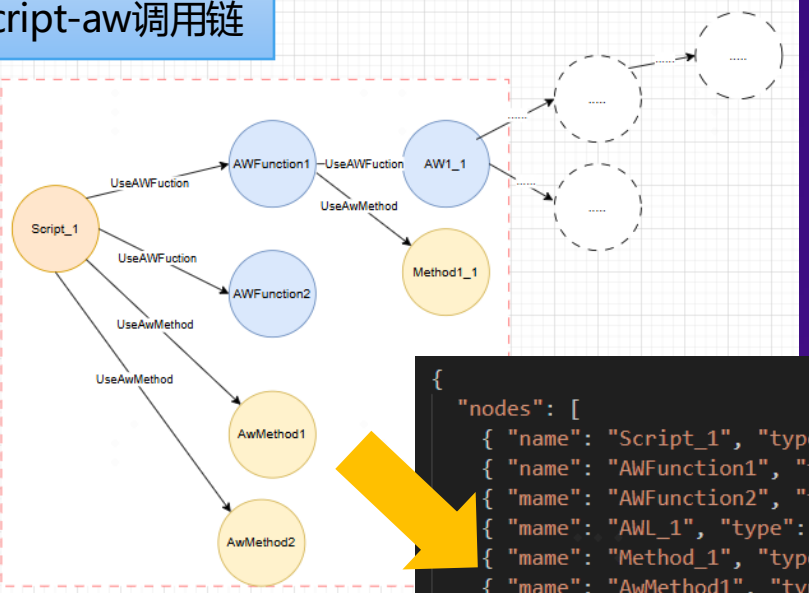
精度与误报的平衡

规则过松则漏报，规则过严则误报频出，需要专家级语义理解能力。

脚本用例一致性检测的实施方案与落地

由结构化代码知识图谱，我们可以获取script-aw调用关系链。调用链可能很大，可以采取一些减枝策略。（如深度上限制层数，或者更精细的层数控制）

script-aw调用链



将script-aw调用链归纳成大模型可理解的文本

```
{
  "nodes": [
    { "name": "Script_1", "type": "script" },
    { "name": "AWFunction1", "type": "aw_function","code":"xxx", docstring:"xxx" },
    { "mame": "AWFunction2", "type": "aw_function", "code":"xxx", docstring:"xxx"},
    { "mame": "AWL_1", "type": "aw_function","code":"xxx", docstring:"xxx" },
    { "mame": "Method_1", "type": "aw_method","code":"xxx", docstring:"xxx" },
    { "mame": "AwMethod1", "type": "aw_method","code":"xxx", docstring:"xxx" }
  ],
  "edges": [
    { "from": "Script_1", "to": "AWFunction1", "relation": "UseAWFunction" },
    { "from": "AWFunction1", "to": "AWL_1", "relation": "UseAWFunction" },
    { "from": "AWL_1", "to": "Method_1", "relation": "UseAwMethod" },
    { "from": "Script_1", "to": "AWFunction2", "relation": "UseAWFunction" },
    { "from": "Script_1", "to": "AwMethod1", "relation": "UseAwMethod" },
    { "from": "Script_1", "to": "AwMethod2", "relation": "UseAwMethod" }
  ]
}
```

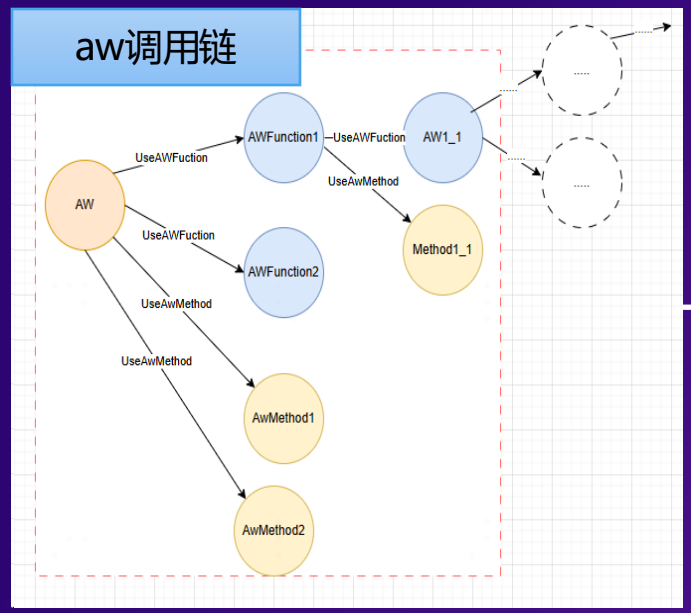
Prompt设计

```
# Prompt设计
## 背景
你是一个测试开发专家，擅长分析测试用例文本描述和测试脚本代码的一致性。
你能拆解测试脚本的函数调用链（以JSON格式提供），并基于此对比测试用例的各
个部分（前置条件、操作步骤、期望结果），识别脚本中缺失或未完整实现的测试点。
## 任务
请对以下测试用例文本描述和测试脚本代码（包括脚本本代码和脚本函数调用链），
分析它们的一致性，重点关注测试用例描述中是否有在脚本中代码或函数调用链中缺少的测试点，
并根据提供的评分规则进行评分和建议。
## 输入数据
### 测试用例文本描述
#### 前置条件:
{{pre_condition}}
#### 操作步骤:
{{test_steps}}
#### 期望结果:
{{expectation_result}}
### 测试脚本:
#### 脚本代码
{{script_code}}
#### 脚本函数调用链
{{script_call_graph}}
（格式为JSON，包含`nodes`和`edges`，`nodes`包括`name`，`type`，`code`，`docstring`等属性
表示脚本中的函数、方法或步骤；`edges`包括`from`，`to`，`relation`，表示调用关系。
这有助于映射测试步骤和期望结果到具体代码实现）
## 分析要求
1. **拆解测试用例文本**：将前置条件、操作步骤、期望结果分解为可检查的独立条目（例如，列表形式）
2. **识别测试脚本**：结合脚本代码和函数调用链，识别脚本中实现的前置条件、操作步骤和期望结果（
3. **对比一致性**：
   - 检查测试用例描述中的每个条目是否在脚本代码或函数调用链中有对应实现。
   - 识别描述中有但脚本中缺少的测试点（例如，未调用的关键函数、缺失的断言）。
4. **应用扣分规则**：
{{deduction_rules}}
## 输出格式
### 一次性评分（0-100）
`【评分值】`
### 扣分项
- `【列表】`
### 一致的部分
- **前置条件列表**：`【列表】`
- **操作步骤列表**：`【列表】`
- **期望结果列表**：`【列表】`
### 不一致的部分
- **前置条件列表**：`【列表】`
- **操作步骤列表**：`【列表】`
- **期望结果列表**：`【列表】`
### 改进建议
- `【建议列表】`
```

各领域自定义扣分规则

脚本用例一致性基础：高质量aw质量检测体系构建

高层的脚本质量依赖于基础的aw质量检测。基于aw的调用链，可以构建aw代码质量检测机制。



将aw调用链归纳成大模型可理解的文本

Prompt设计

aw质量检测agent

aw代码问题检测

- **1.功能未实现**
- **2.功能不健壮**
- **3.功能逻辑错误**
- **4.性能低下**
- **5.安全性缺陷**
- **6.集成/兼容性问题**

通过构建基于结构化知识图谱的aw调用链，实现代码依赖关系的可视化、可推理与可追溯，为脚本质量检测提供精准、高效、可扩展的智能分析基础。

脚本质量提升落地效果

在脚本管理平台脚本质量检测流程中接入用例脚本一致性检测。

83%

识别完全正确的用例数（无错检，漏检）/ 总用例数

10%

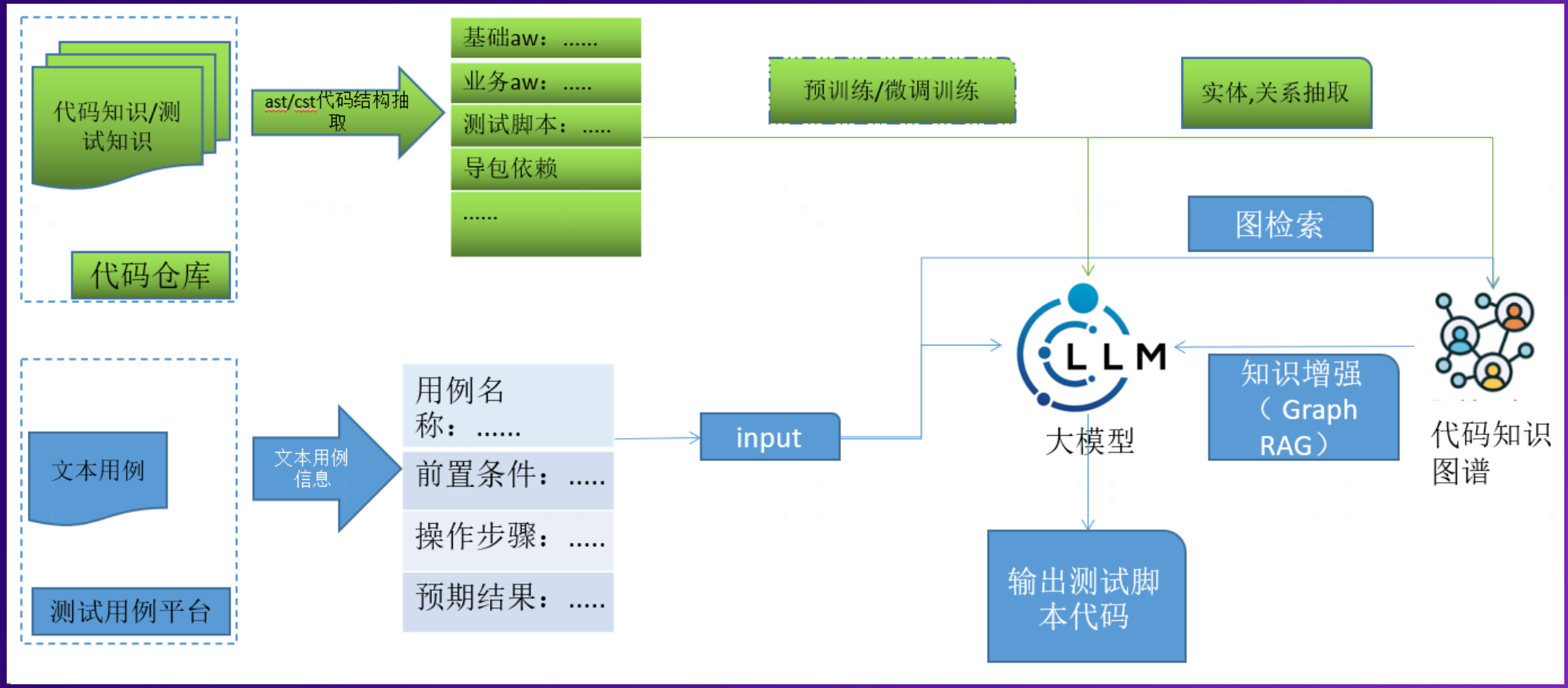
识别不完全正确用例数/ 总用例数

AI驱动的脚本用例一致性检测，显著提升脚本质量管控效率

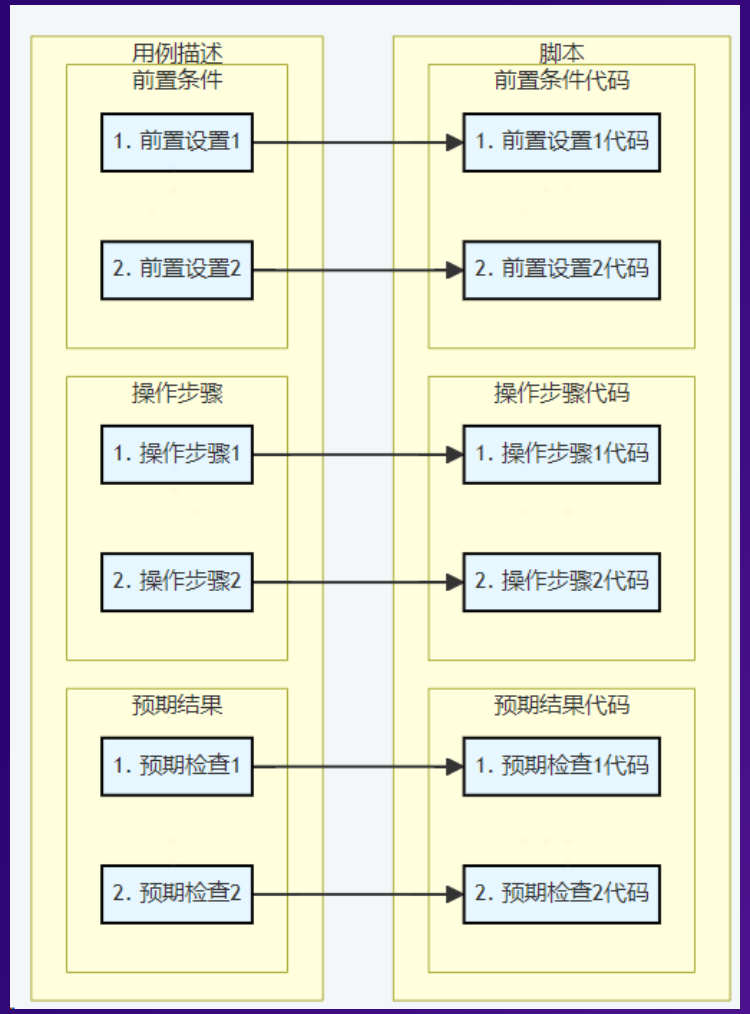
PART 05

脚本生成落地

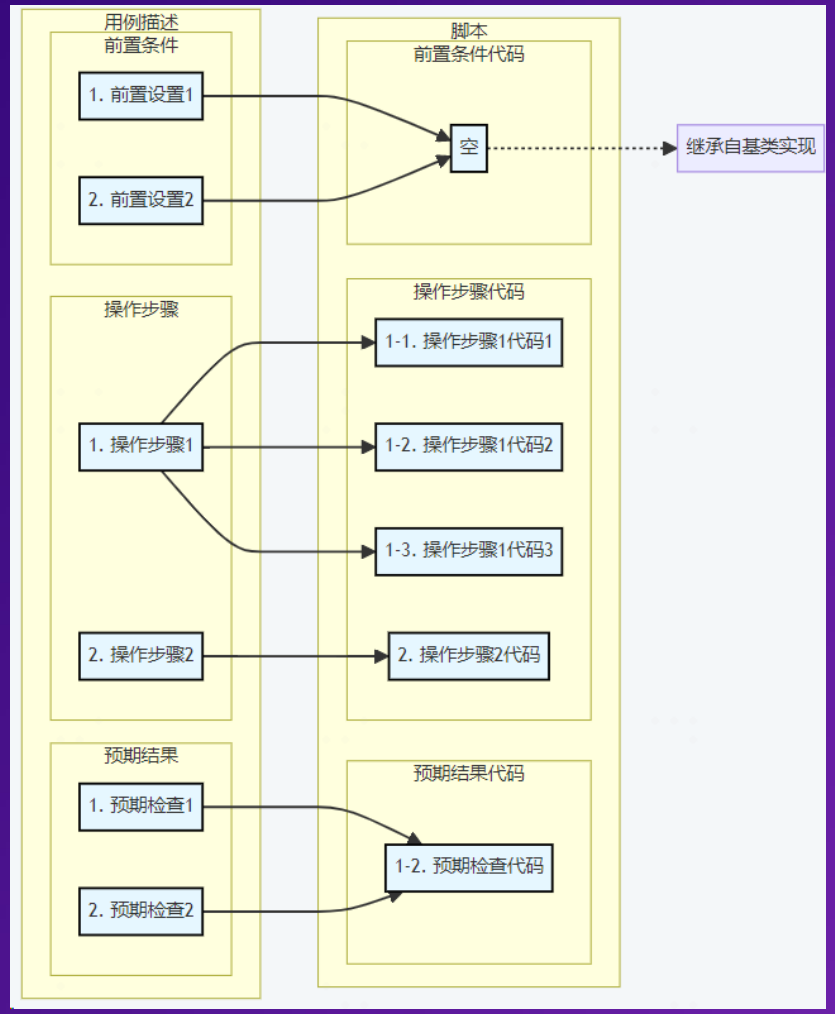
脚本生成整体方案


代码知识/测试知识
代码仓库
ast/cst代码结构抽取
基础aw：.....
业务aw：.....
测试脚本：.....
导包依赖
.....
预训练/微调训练
实体,关系抽取
图检索
文本用例
测试用例平台
文本用例信息
用例名称：.....
前置条件：.....
操作步骤：.....
预期结果：.....
input
LLM
大模型
知识增强（Graph RAG）
代码知识图谱
输出测试脚本代码

脚本生成:用例描述与代码实现不匹配的挑战和方案



用例描述与代码实现一一对应

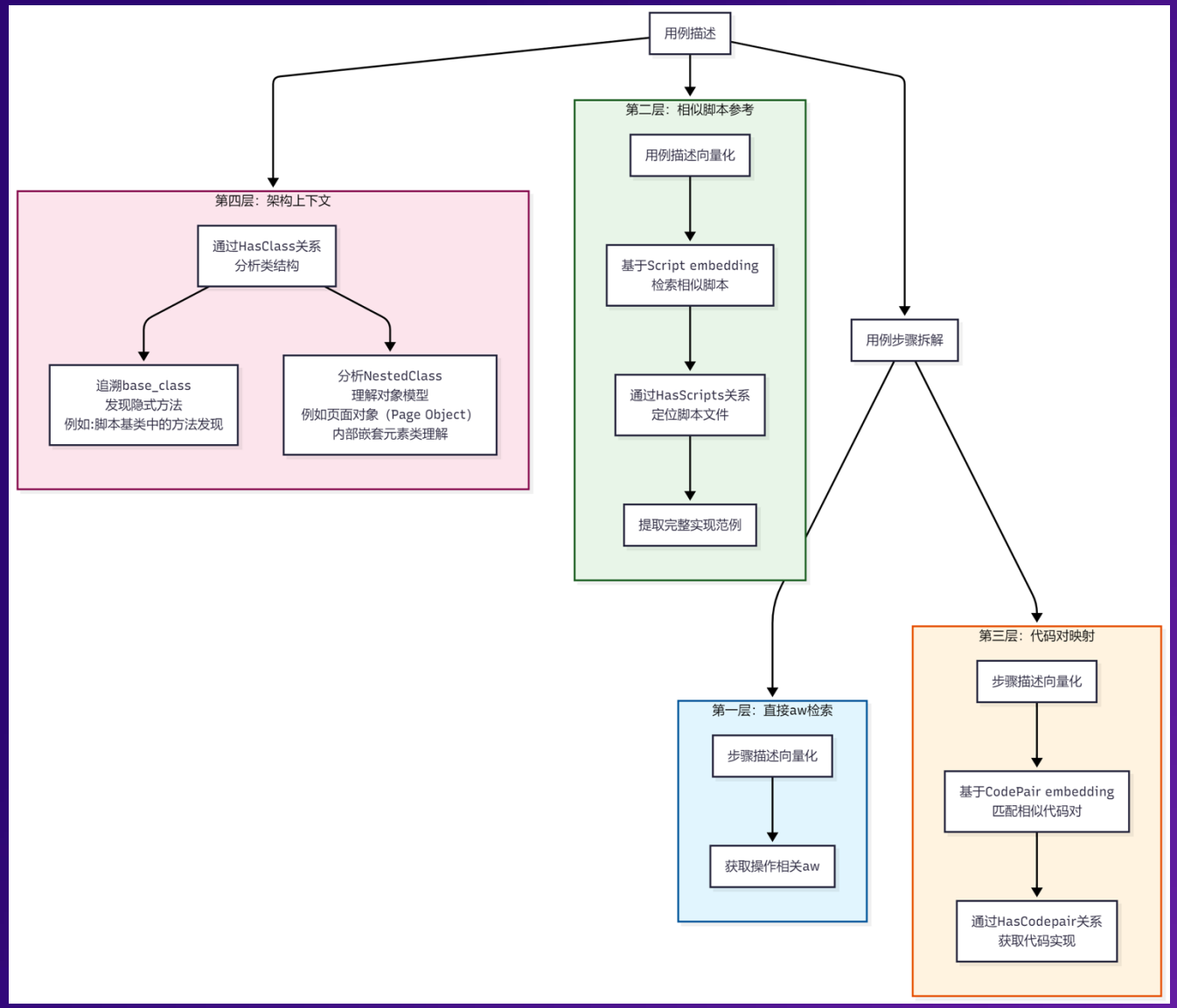


用例描述与代码实现非一一对应

挑战：用例-代码不匹配场景

- **一对多映射**：单个用例步骤需要多个AW组合实现
- **隐式依赖**：功能已在基类实现，无需显式调用
- **上下文缺失**：用例描述简略，缺乏技术实现细节

脚本生成:用例描述与代码实现不匹配的挑战和方案

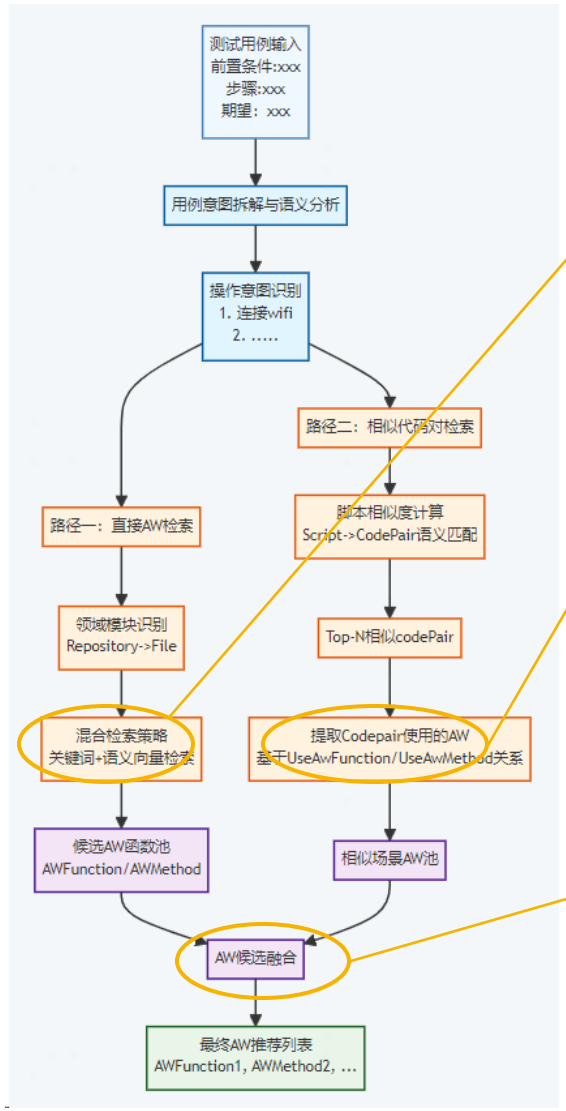


方案：四层增强检索设计

- **1.aw检索**：适用于一一对应场景
- **2.相似脚本**：提供宏观蓝图，提供完整实现范例
- **3.相似代码对**：处理步骤级的多AW组合代码实现
- **4.架构上下文**：发现隐式可用方法

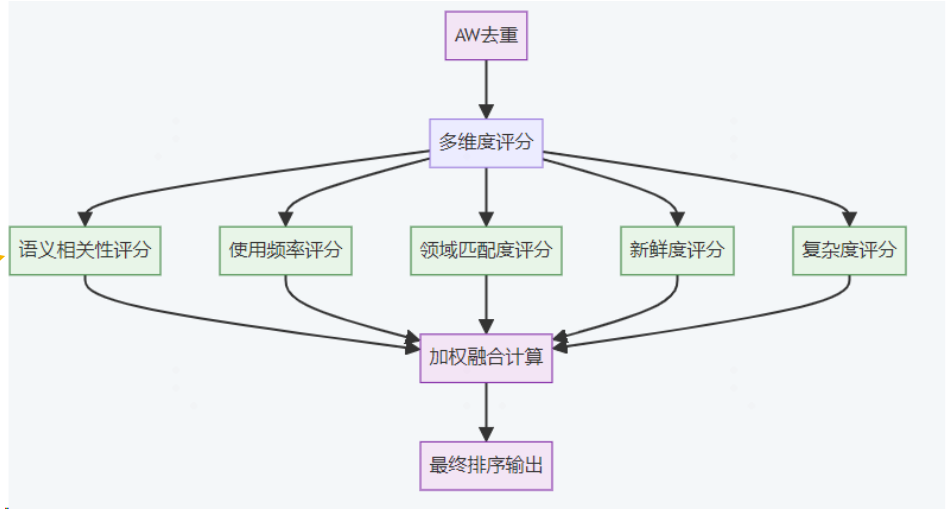
通过结构化知识图谱的多层次检索，有效应对用例与代码的非一一对应挑战，提升脚本生成的准确性和效率。

脚本生成：基于code-graph的AW检索设计




```
# aw语义检索伪代码
MATCH (aw:AWFunction)
WHERE aw.aw_description_embedding ~ "连接WiFi的向量"
RETURN aw ORDER BY similarity_score DESC
```

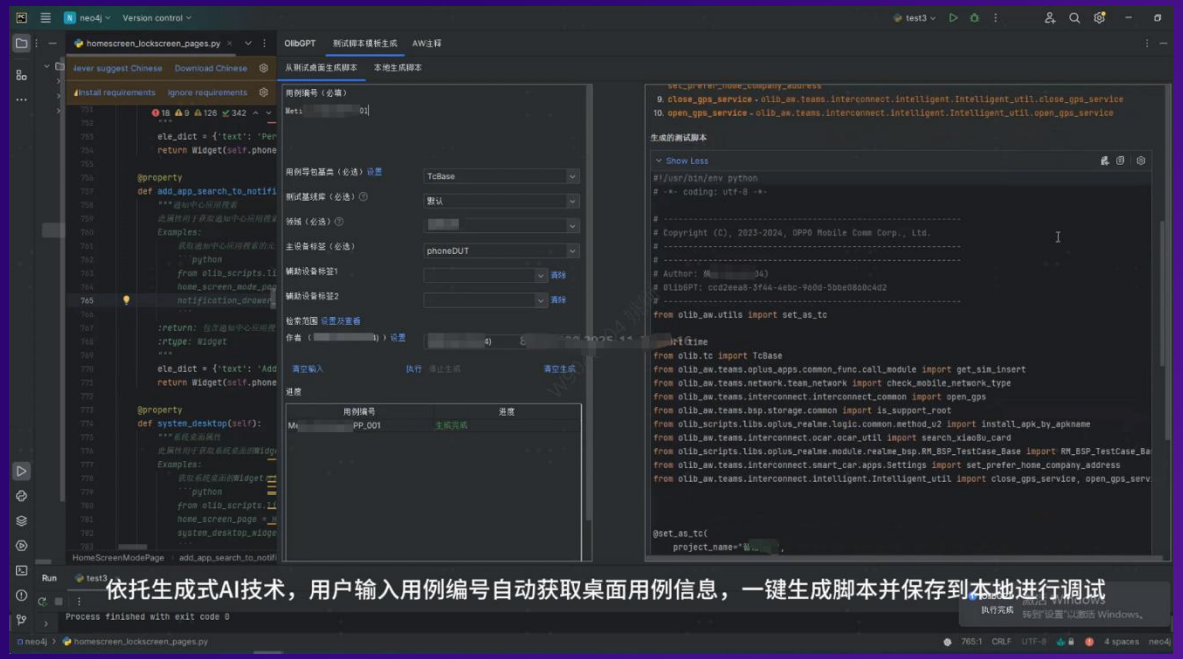
```
# 基于CodePair语义匹配检索aw伪代码
MATCH (s:Script)-[:HasCodepair]->(cp:CodePair)
MATCH (s:Script)-[:UseAwFunction]->(aw:AWFunction)
WHERE cp.name EMBEDDING SIMILAR $test_case_text
RETURN aw, COUNT(DISTINCT s) as script_frequency
ORDER BY script_frequency DESC
```



基于结构化代码知识图谱的灵活性，可以因地制宜地设计最佳的aw检索算。

脚本生成工具落地与成效

在ide中集成脚本生成插件



- 输入用例编号，选择基线
- 自动从测试桌面获取文本用例
- 基于大模型生成脚本到代码仓
- 直接在ide中调试

整体落地效果

AI生成脚本占比		AI生成代码占比	
78.91%	AI脚本入库次数 总脚本入库次数	44.01%	AI代码采纳行数 入库总代码行数

领域效果

AI生成脚本占比		AI生成代码占比	
94.42%	AI脚本入库次数 总脚本入库次数	57.76%	AI代码采纳行数 入库总代码行数
99.57%	AI脚本入库次数 总脚本入库次数	51.56%	AI代码采纳行数 入库总代码行数

PART 06

总结与展望

总结与展望

1.总结

代码知识图谱

aw-aw调用链

script-aw调用链

增强代码知识检索

AW资产质量↑

- 注释质量提升
- ui操作健壮性提升
- AW 代码质量检测

文本用例质量↑

用例质量检查

脚本质量↑

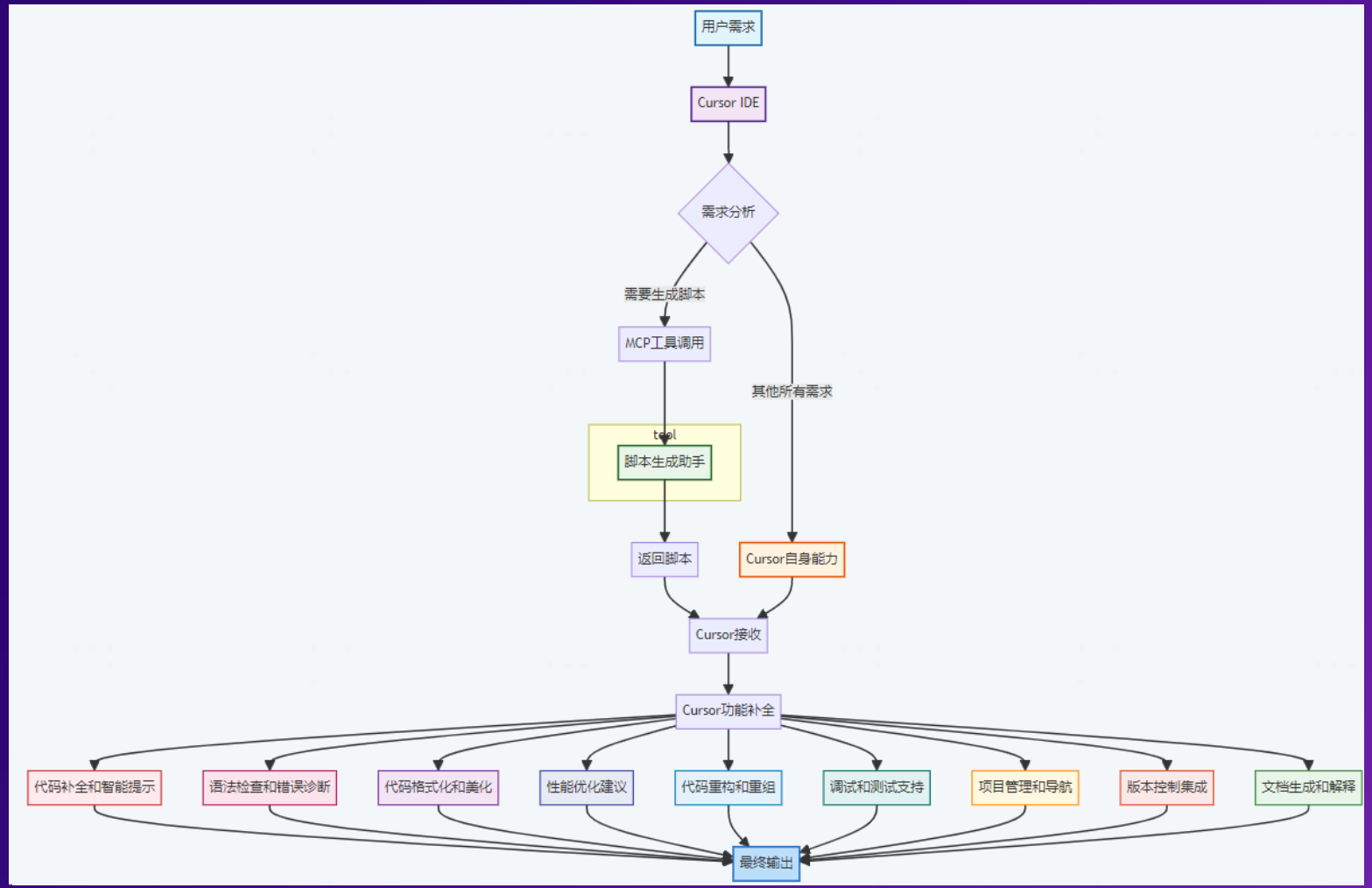
脚本用例一致性检测

脚本生成agent

高质量新脚本生成

总结与展望

2.展望



脚本生成工具
- 专注脚本生成核心功能
- 内置行业最佳实践
- 标准化代码输出

智能编程助手
- 提供完整开发环境
- 智能代码补全和提示
- 实时错误检测和调试
- ……

脚本生陈工具与智能编程助手深度集成，为测试脚本开发提供全流程的智能辅助。

科技生态圈峰会 + 深度研习

——1000 + 技术团队的共同选择

KEYLINK ing

AiDD 峰会

AiDD峰会 上海站
AI+研发数字峰会
时间：2026.05.22-23

AiDD峰会 北京站
AI+研发数字峰会
时间：2026.08.21-22

AiDD峰会 深圳站
AI+研发数字峰会
时间：2026.11.20-21

AiDD峰会详情

AiDD × AI+产品峰会

AiDD × 上海站
AI+产品创新峰会
时间：2026.01.16-17

AiDD × 北京站
AI+产品创新峰会
时间：2026.08.23-24

产品峰会详情

AiDD × K

2026 AI+ PRODUCT INNOVATION SUMMIT

01.16-17 · ShangHai

AI+产品创新峰会

Track 1: AI 产品战略与创新设计

从0到1的AI原生产品构建

论坛1: AI时代的用户洞家与需求发现

论坛2: AI原生产品战路与商业模式重构

论坛3: AgenticAI产品创新与交互设计

Track 2: AI 产品开发与工程实践

从1到10的工程化落地实践

论坛1: 面向Agent智能体的产品开发

论坛2: 具身智能与AI硬件产品

论坛3: AI产品出海与本地化开发

Track 3: AI 产品运营与智能演化

从10到100的AI产品运营

论坛1: AI赋能产品运营与增长黑客

论坛2: AI产品的数据飞轮与智能演化

论坛3: 行业爆款AI产品案例拆解

2-hour Speech: 回归本质

用户洞察的第一性

——2小时思维与方法论工作坊

在数字爆炸、AI迅速发展的时代，仍然考验“看见”的“同理心”

Panel 1: 出海前瞻

“出海避坑地图”圆桌对话

——不止于翻译:

AI时代的出海新范式

Panel 2: 失败复盘

为什么很多AI产品“叫好不叫座”?

——从伪需求到真价值:

AI产品商业化落地的关键挑战

智能重构产品 数据驱动增长

Reinventing Products with Intelligence, Driven by Data

80+ 业界大咖 汇聚产品大咖的真知灼见

40+ 创新案例 开拓全新AI+产品视角

10+ 分论坛 涵盖产品到商业增长的全链路

800+ 听众规模 共同探索产品的智能重构

扫码查看会议详情

第8届AI+研发数字峰会
AI+ Development Digital Summit

感谢聆听！

扫码领取会议PPT资料