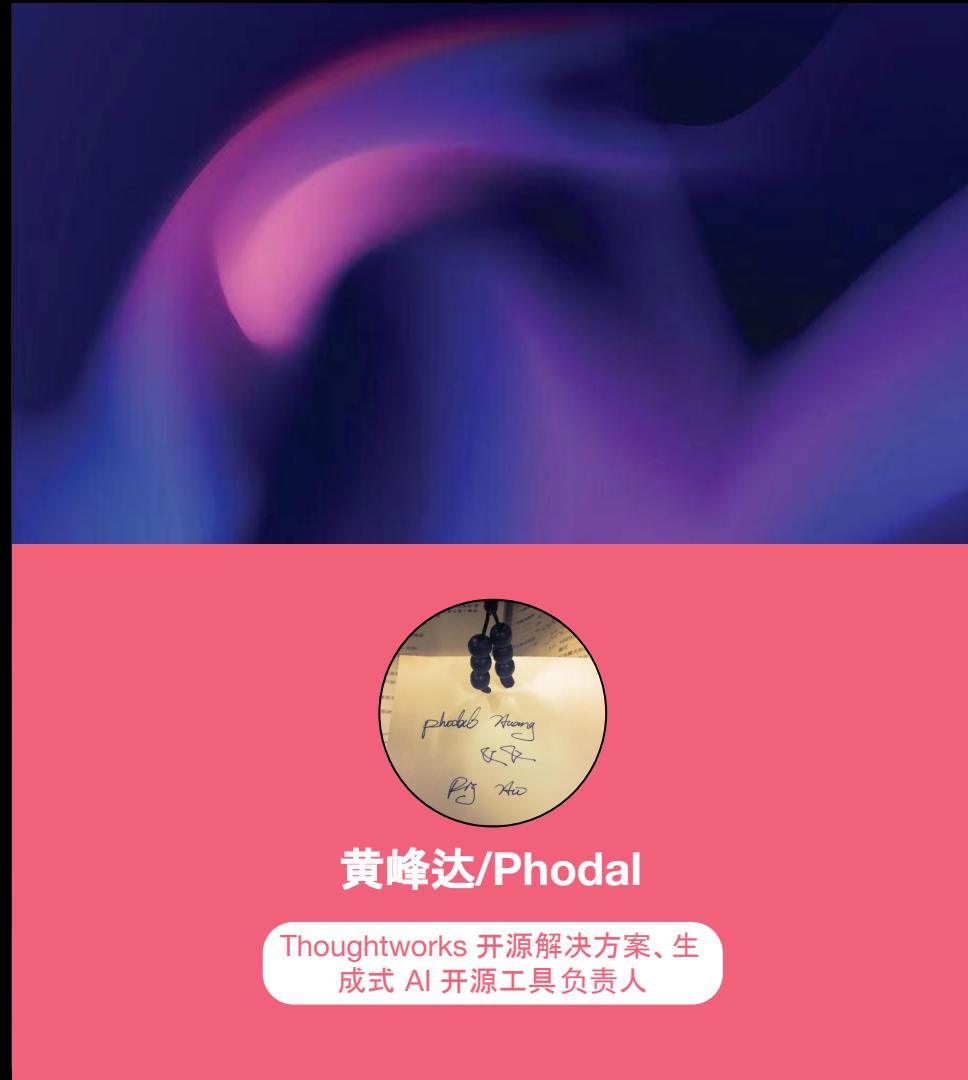


## 主题分享 #2

# AI 赋能研发开源 方案UnitMesh

/thoughtworks

Strategy. Design. Engineering.

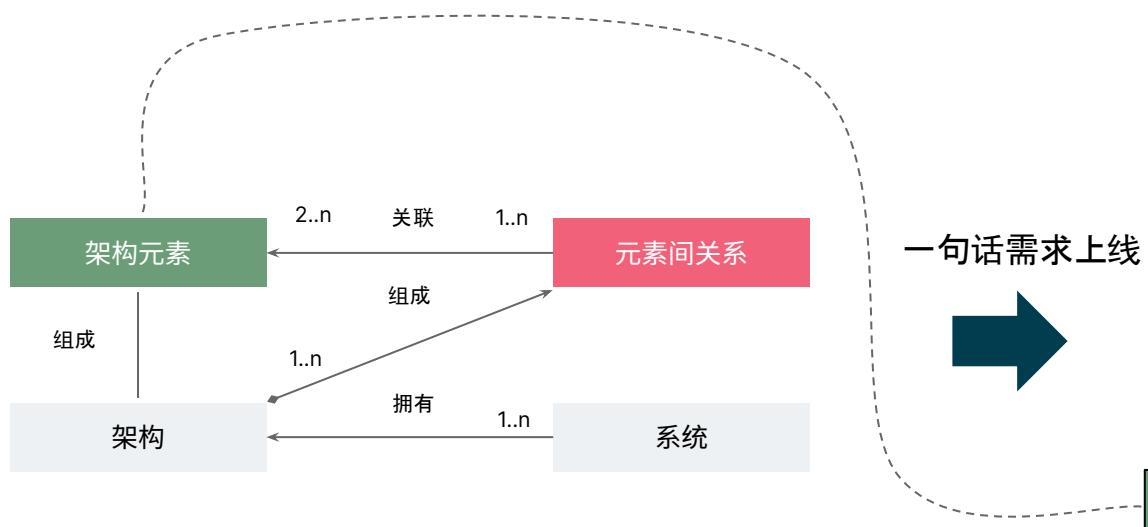


黄峰达/Phodal

Thoughtworks 开源解决方案、生  
成式 AI 开源工具负责人

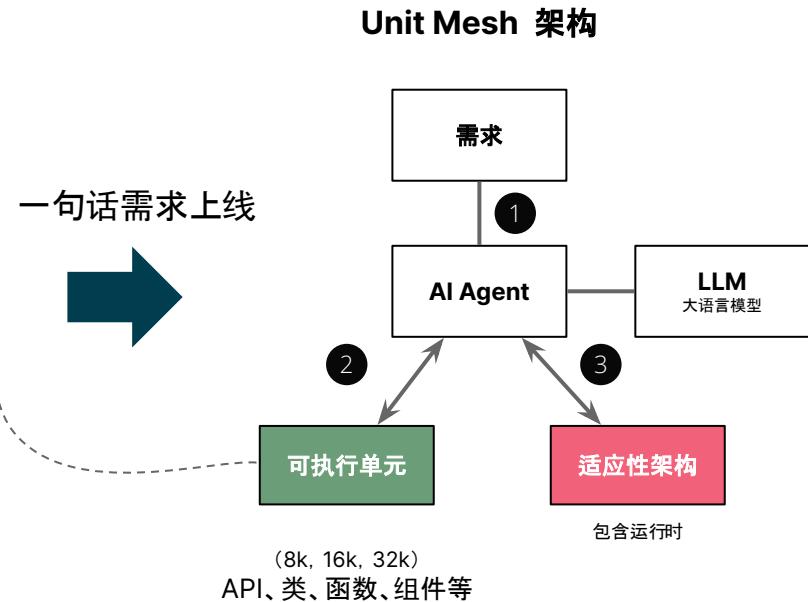
# 软件架构模式转变: Unit Mesh 架构设想

生成式 AI 驱动下的软件架构的一种可能性 (2023.03)



架构是什么？

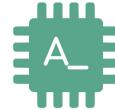
出自《软件系统架构: 使用视点和视角与利益相关者合作》(部分)



# **DEMO**

# AutoCRUD: 基于已有架构体系的编码 AI Agent

思想：以人为本，AI 生成代码，人来检验 AI 生成的代码



1. create\_story\_detail.vm
2. create\_dto\_and\_entity.vm
3. lookup\_or\_create\_endpoint.vm
4. create\_service\_and\_repository.vm
  - a. create\_controller.vm
  - b. update\_controller\_method.vm

后端 CRUD 数据流  
(AutoDev 中的 AutoCrud 提示词模板)

你是一个资深的后端 CRUD 工程师，请根据下面的用户故事和 Controller 列表。要求：

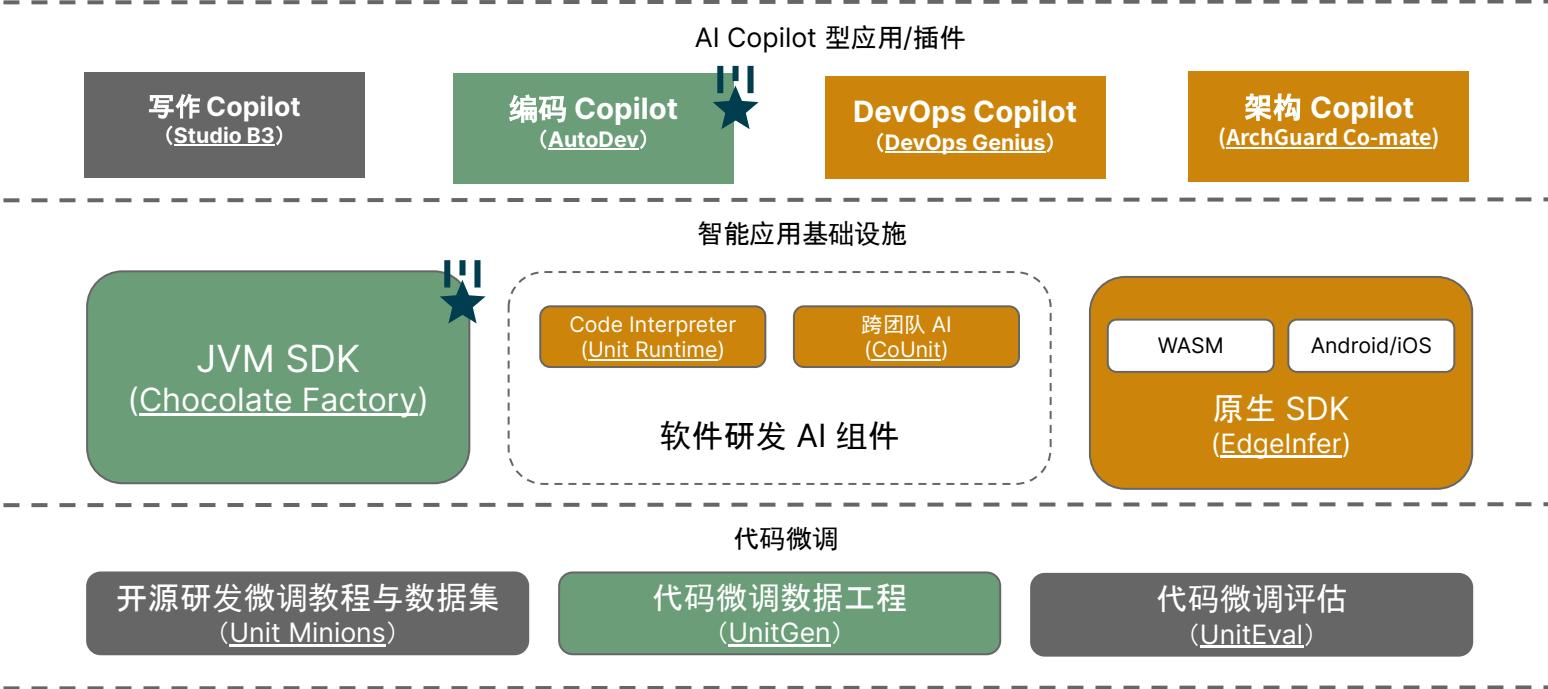
- 返回最合适的 Controller 名字
- 如果不存在合适的 Controller 名字，请返回适合的 Controller 名字。

Controller 列表：

```
###  
{controllers}  
###  
  
###  
{storyDetail}  
###
```

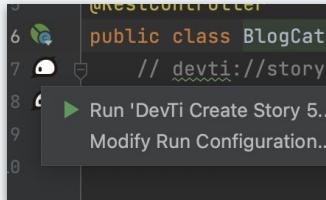
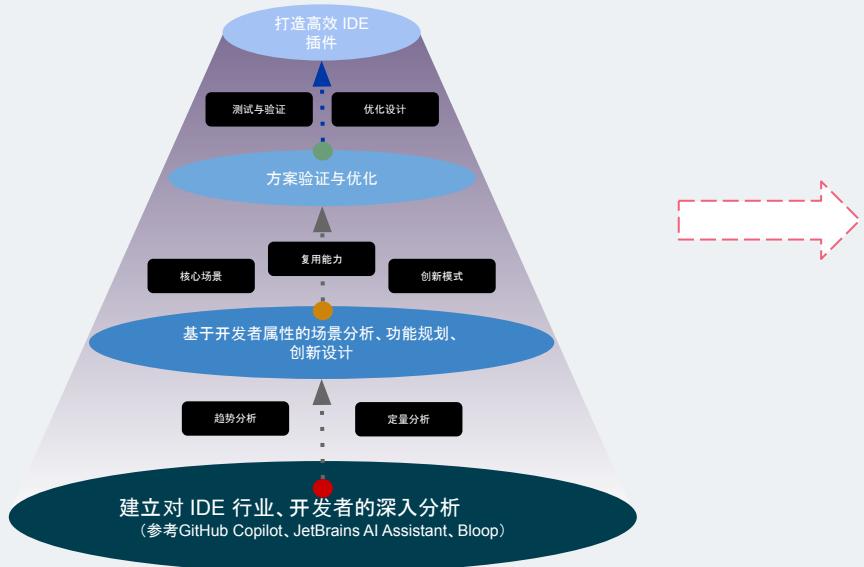
# Thoughtworks AIGC 赋能研发开源方案

Unit Mesh -> GitHub: <https://github.com/unit-mesh>



# AutoDev - AI 辅助开发

GitHub: <https://github.com/unit-mesh/auto-dev>



工具链集成  
(如:DevOps 看板、API 市场)

A screenshot of an IDE showing an AI assistant's suggestions for a 'commitParser()' method. The suggestions include 'Test this', 'Code complete', 'Generate Living Doc', and 'Translate to Kotlin'. The code snippet is annotated with 'AutoDev AI Assistant' and 'Chat with 'createCommitParser' function'.

```
commitParser(): CommitParser {
    ns     AutoDev AI Assistant
    sg Chat with 'createCommitParser' function
    ns Test this
    Code complete
    Generate Living Doc
    ns Translate to Kotlin
```

自动化生成  
(代码、测试、文档等)

A screenshot of an IDE showing a context menu for a 'Task :compileJava' entry. The menu includes options like 'Fix This (AutoDev)', 'Copy', 'Compare with Clipboard', and 'Search with Google'. The code snippet is annotated with 'Task :compileJava' and 'Fix This (AutoDev)'.

```
Task :compileJava
Users/phodal/IdeaProjects/untitled
Entity
^ Fix This (AutoDev)
symbol Copy ⌘C
Users/
Table Search with Google
```

围绕开发者日常  
(错误修复建议、提交信息生成、  
API 测试数据生成等)

A screenshot of an IDE showing various AI-powered developer assistance features. At the top, a bar says 'AutoDev Explain selected code Code'. Below it, a 'User' section says 'Explain selected Kotlin code:' followed by some sample code. A 'Kotlin' section shows code annotations for 'SimpleToolWindowPanel', 'NullableComponent', 'progressBar', 'myTitle', and 'myList'. A 'Assistant' section provides detailed explanations for each of these components. At the bottom, there's a 'Want new feature?' button and a '聊天' (Chat) section with a note '(带上下文)'.

Kotlin

```
SimpleToolWindowPanel(true, true),
NullableComponent {
private var progressBar: JProgressBar =
private val myTitle = JLabel
("Conversation")
private val myList = JPanel
```

Assistant

The provided Kotlin code defines a class called ChatCodingPanel. This class extends SimpleToolWindowPanel and implements the NullableComponent interface. It has several properties and components:

- chatCodingService: This is a dependency injected into the class constructor. It is of type ChatCodingService.
- disposable: This is another dependency injected into the class constructor. It is of type Disposable.
- progressBar: This is a private property of type JProgressBar. It is used to display progress in the panel.
- myTitle: This is a private property of type JLabel. It represents the title of the panel and is initialized with the text "Conversation".
- myList: This is a private property of type JPanel. It is used to hold a list of items in a vertical layout.
- inputSection: This is a private property of type AutoDevInputSection. It represents an input section in the panel.
- focusMouseListener: This is a private property of type MouseAdapter. It is used to handle mouse events related to focus.
- panelContent: This is a private property of type DialogPanel. It represents the content of the panel.
- myScrollPane: This is a private property of type JBScrollPane. It is used to provide scrolling functionality to the panel.

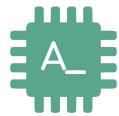
The code snippet provided is a partial implementation of the ChatCodingPanel class. It defines the properties and components that

'Enter' to start, 'Shift+Enter' for a new line

聊天  
(带上下文)

# AutoDev 能力全景

支持自有大模型、自定义 AI 行为等



自定义模型

自定义 Prompts

团队 Prompts

自定义规范

自定义文档

Prompt 覆写

# **DEMO**

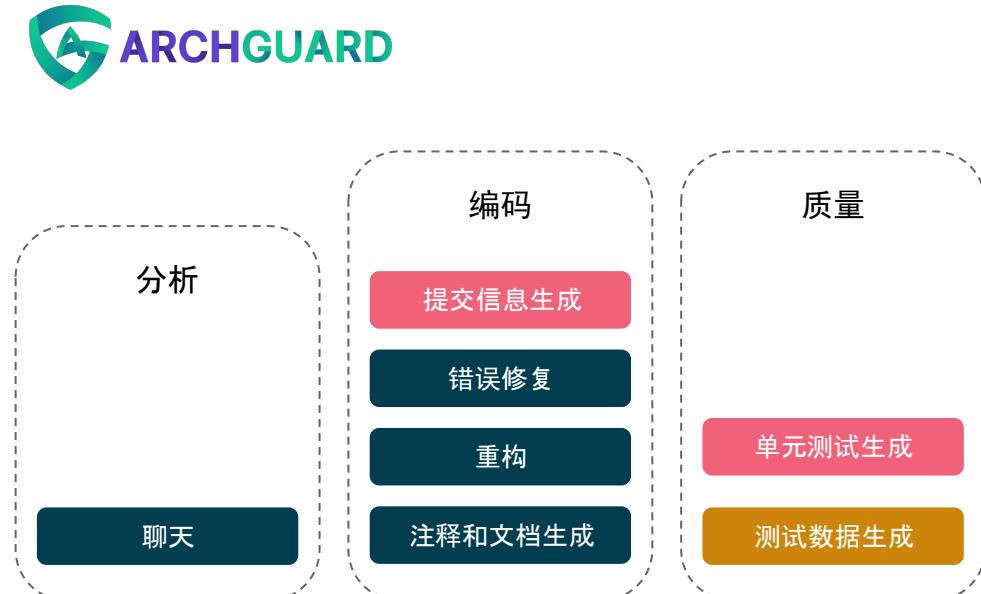
# AutoDev 的上下文工程: 精准单元测试生成示例

结合静态代码分析、依赖分析、通用测试规范，生成测试上下文，以生成精准的测试用例



# AutoDev + ArchGuard 质量提升(Kotlin 语言)

一周五天：90+ 测试类，200+ 测试用例，测试覆盖率提升 11.55%，行数累增 6610，非空行数 4663

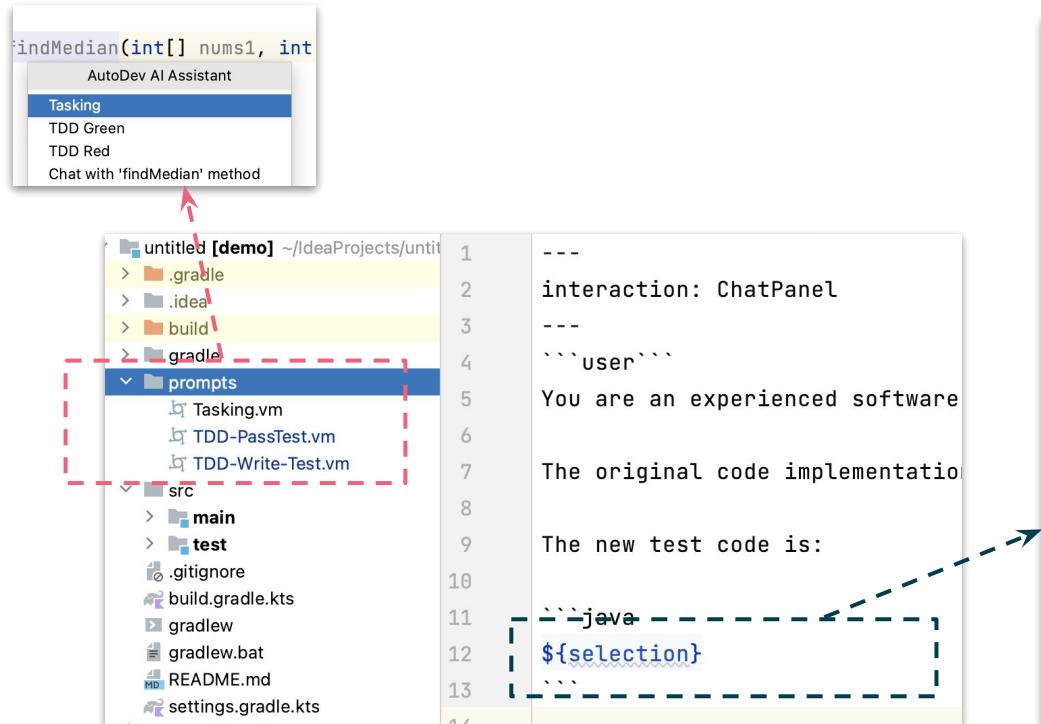


功能	次数
提交信息生成	167 次
编写单元测试	134 次
生成测试数据	32 次
重构	22 次
编写文档	36 次
其它	31 次

GitHub: <https://github.com/archguard>

# 团队 AI: 自定义团队的 AI Action

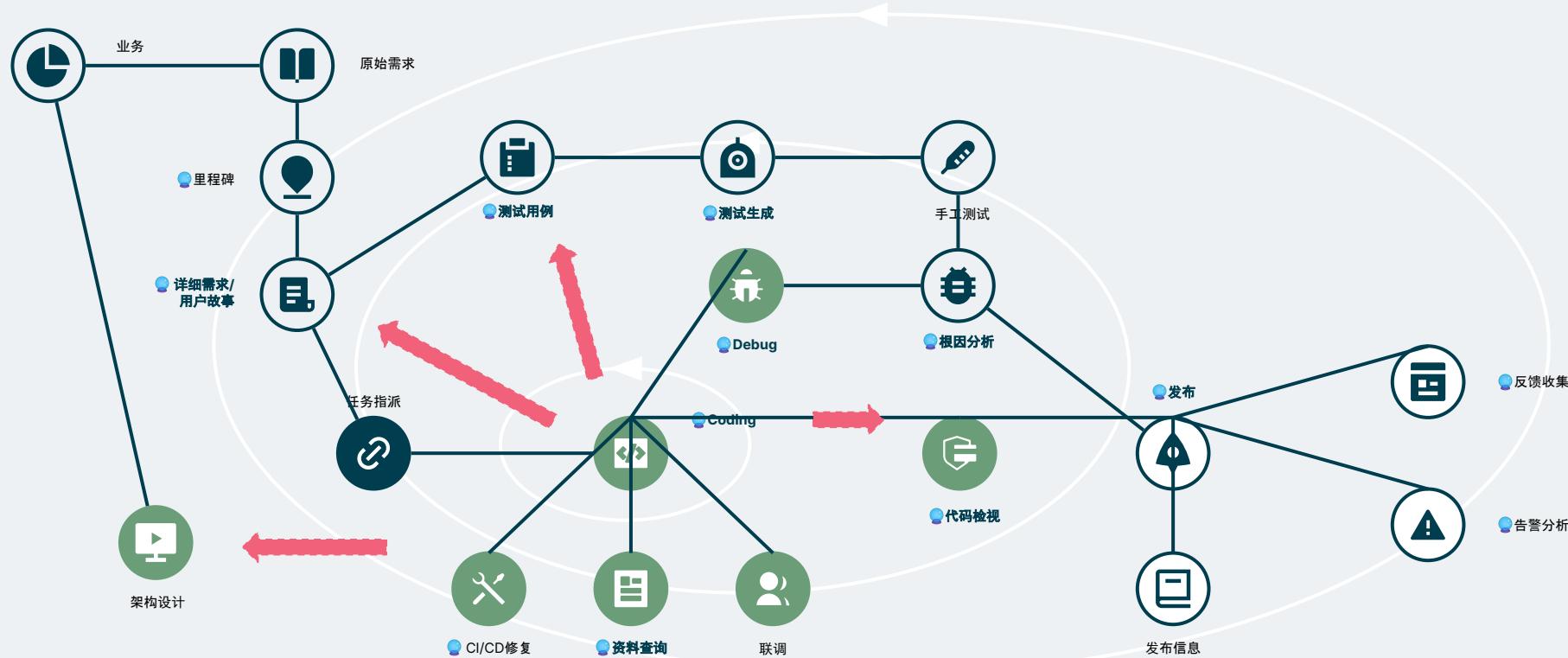
Prompt 即代码，在团队中共享适合于项目的 AI 辅助实践，提升整体开发效率



- selection: 当前选定的文本。
- commentSymbol: 获取当前语言的注释符号, 例如:  
//, #, --, /\* \*/
- beforeCursor: 当前光标之前的文本。
- afterCursor: 当前光标之后的文本。
- language: 当前文件的语言, 例如: kotlin, java, python, javascript。
- fileName: 当前文件的文件名。
- filePath: 当前文件的文件路径。
- methodName: 当前方法的方法名。
- frameworkContext: 获取当前文件的框架上下文, 例如: spring, junit, mockito 等。

# Development 的日常协调工作

需求写不好，开发再快也没用



AI 辅助需求编写是一个复杂的知识梳理与工程化。除了模型，还需要考虑：既有实现逻辑、文档中的业务知识、可行性分析等。

需求编写也需要一个全方位辅助的 AI 文本编辑器。

—— Studio B3



# **DEMO**

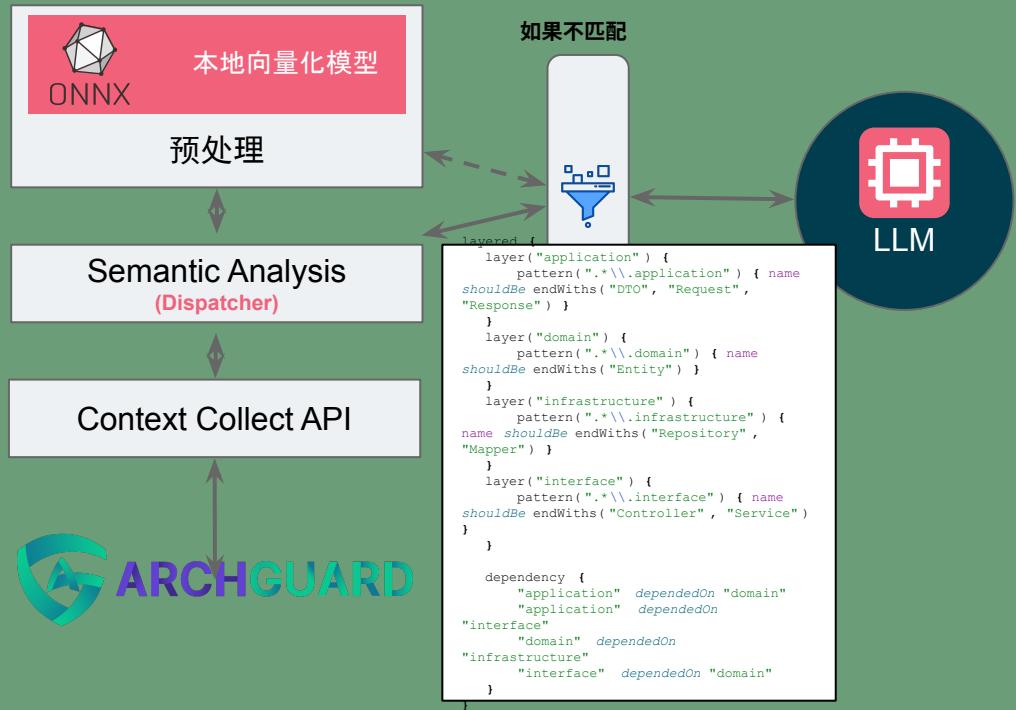
按真正的 LLM as Copilot 方式，深入角色日常活动设计 AI 工具，才能大提升角色的效率。并让普通的模型具备更高的可落地性。



不追求一次生成，提升 AI 使用频次，以更好的实现 AI 增强人类。

# 生成式 AI 辅助架构 ArchGuard Co-mate

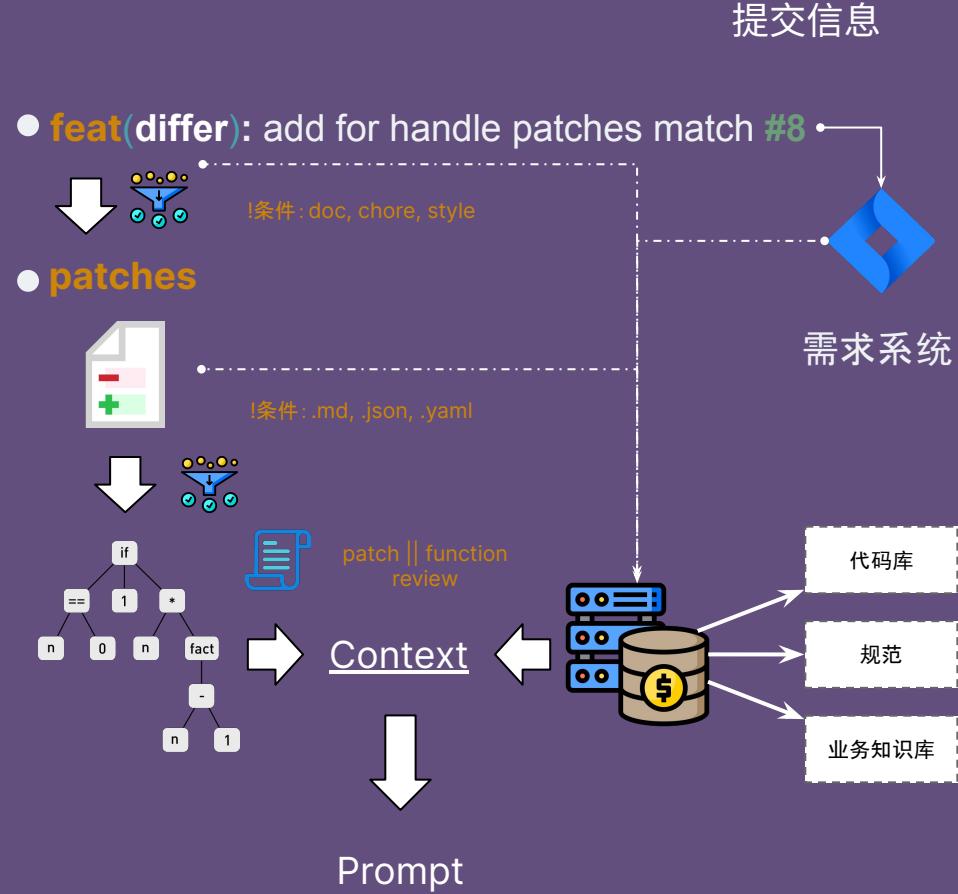
架构是一个复杂的问题，人类没有建立统一的标准。  
生成式 AI 的辅助强弱，取决于架构的数字化水平，与开发者对于架构的理解。



ArchGuard 是一个由 Thoughtworks 开源的架构治理平台。它可以帮助架构师、开发人员进行架构自助，自定义架构的洞察、分析系统间的远程服务依赖情况、数据库依赖、API 依赖等。并根据一些架构治理模型，对现有系统提出改进建议。

# **DEMO**

AI 在研发领域落地还需要流程设计，它还意味着：重塑现有的流程与规范。



# 代码检视的多 AI 工具协作设计

重新审视代码检视实践，哪些知识继续由人来学习，哪些动作可以 AI 完成？



IDE 插件



预提交检查

日常场景：在提交代码前，做一些预检查。



DevOps 平台



自动修复 (PR)

日常场景：AI 修复 sonar、AI 安全等工具发现的问题。



IDE 插件



业务变更总结

日常场景：总结一下团队成员最近的代码修改。

The screenshot shows a code editor with Python-like syntax. A tooltip at the bottom right says "ML-SUGGESTED EDIT". The code includes comments like "Finds the opid name using the example datetime." and "If the opid is not found, it returns None". Below the code, there's a "Close" button and an "Apply" button.

DevOps 平台

IDE 插件

ChatBot/IM

数据 API

LLM SDK

需求信息

代码

需求、代码信息

生成式 AI



语法错误

代码风格违规

潜在的 Bugs

自然语言处理  
(NLP)

注释和文档

语义化分析

学习

历史数据

自适应推荐

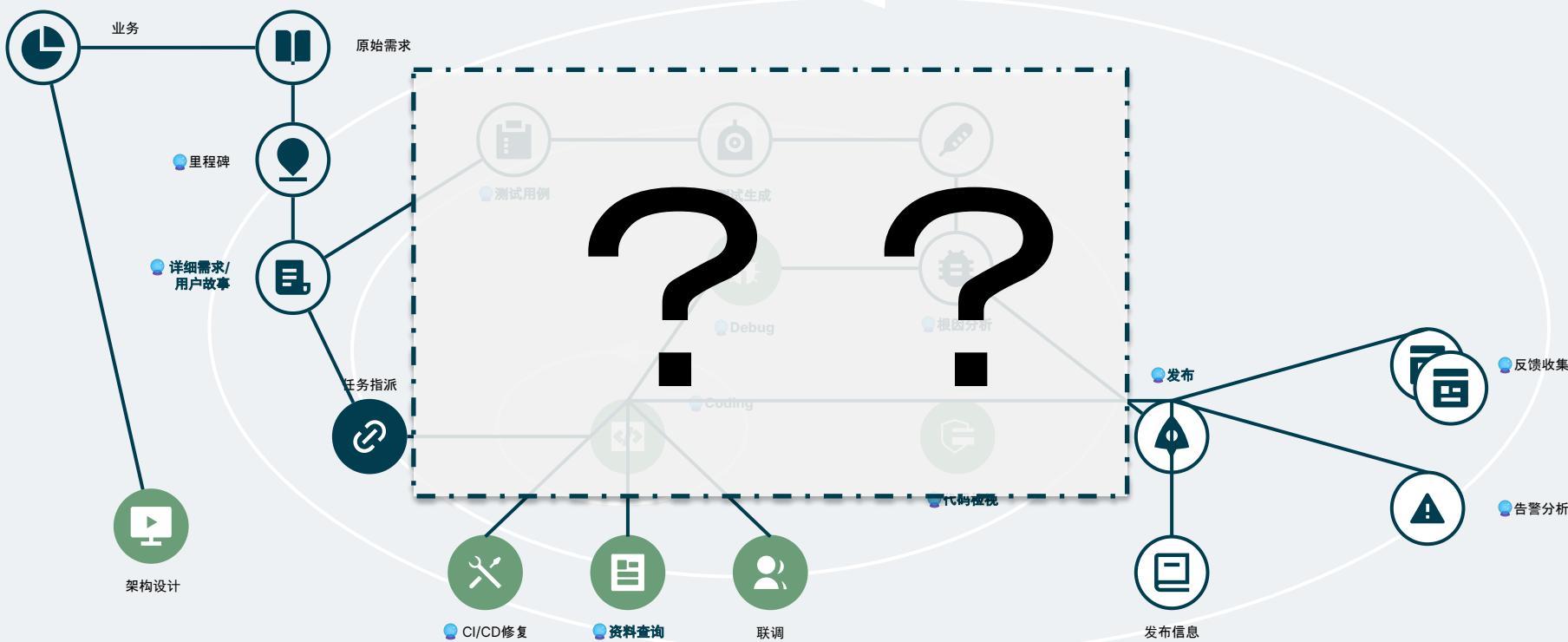
自动纠正错误代码



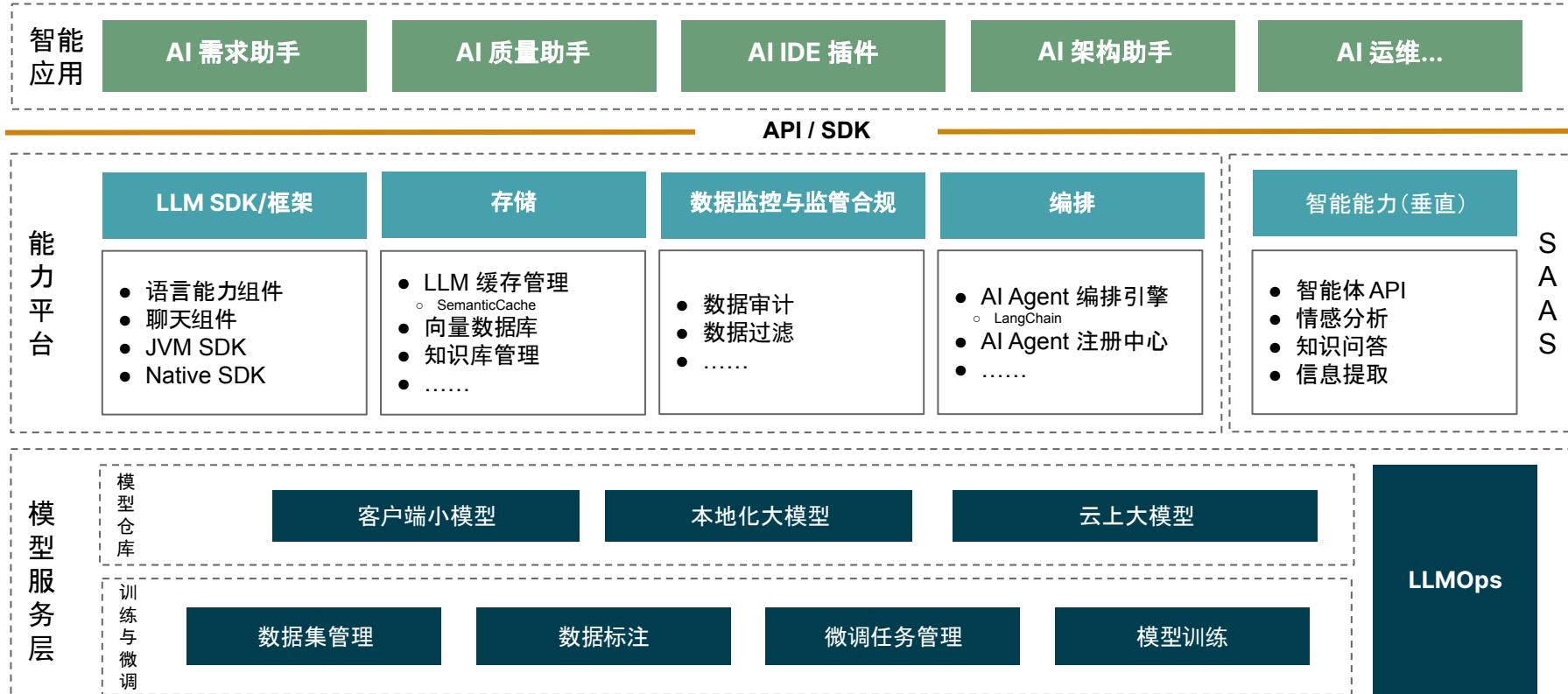
存储

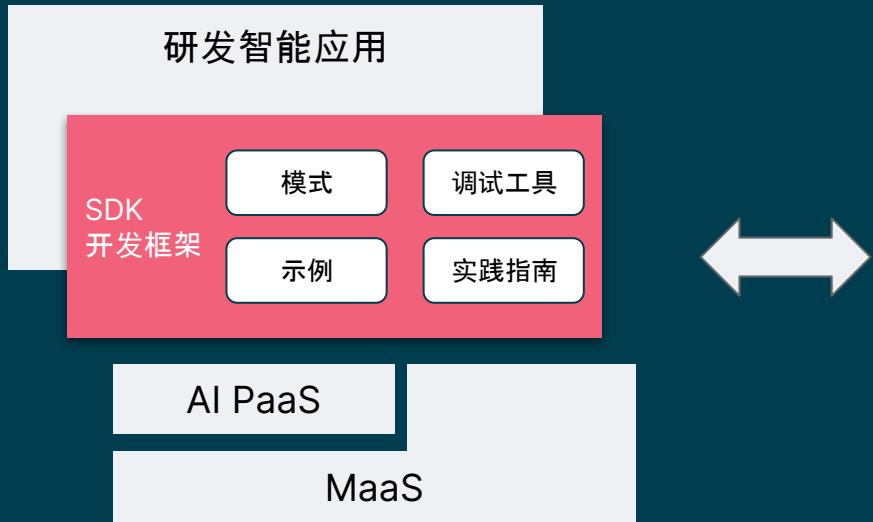
# 数智化时代的工具链

我们应该探索更多的从单点走向跨角色辅助 => LLM as Integrator。



# LLM 应用参考架构全景





结合现有基础设施的智能应用开发框架，是 Copilot 型应用 AI 工程化落地的基本要素。

# AI 应用开发框架

生成式 AI 应用不断多元化且日益丰富；专用任务模型体积缩小、端侧 AI 能力提升，使其能够在桌面 CPU 和移动设备上等端侧运行。

生成式 AI 应用



JVM SDK  
(JVM)

生成式 AI 应用:IM/协同办公  
搜索增强:本地语义化搜索  
端侧辅助:本地自动补全  
指令解析:智能中枢



Mobile SDK  
(嵌入式设备)

搜索增强:本地语义化搜索  
端侧辅助:本地自动补全

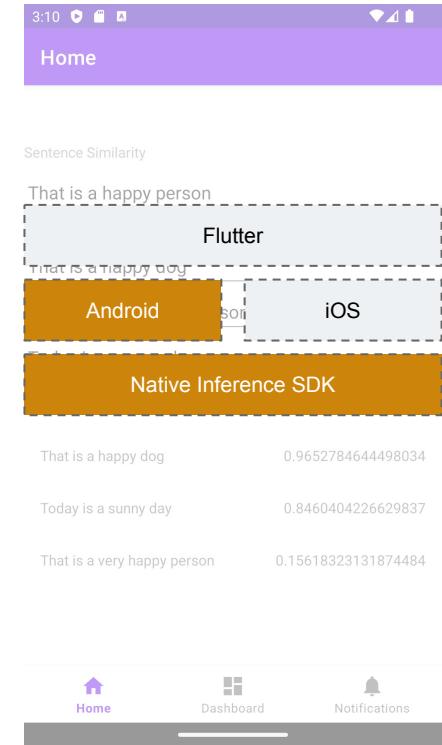


Web/Browser SDK  
(WASM)

智能办公场景



Other OS



# AI 应用开发框架: Chocolate Factory

开源 LLM 开发框架，旨在帮助您轻松打造强大的研发 Copilot 型应用。



```
dependencies {  
    implementation("cc.unitmesh:cocoa-core:0.3.4")  
    implementation("cc.unitmesh:code-splitter:0.3.4")  
    implementation("cc.unitmesh:store-elasticsearch:0.3.4")  
    implementation("cc.unitmesh:sentence-transformers:0.3.4")  
}
```

GitHub: <https://github.com/unit-mesh/chocolate-factory>

应用案例：

- 辅助编程 IDE 插件 AutoDev: <https://github.com/unit-mesh/auto-dev>
- 移动端结合 native SDK 示例: <https://github.com/unit-mesh/android-semantic-search-kit>
- DevOps 工具链: <https://github.com/unit-mesh/devops-genius/>

LLM 基础（类似 LangChain）特性：

- LLM 基础设施封装，如向量数据库、LLM API 封装。
- 检索增强生成（RAG）知识与模式集成。

应用开发、特性：

- 模块化开发：基于 JVM 语言体系，无缝与现有系统集成，按需选取。
- SDLC 模块：版本管理、静态代码分析、前端 UI 设计 DSL 等。
- Prompt 持续集成与测试、RAG PoC 场景下的 Script 优化。

# AI 应用开发框架: 文档沉淀知识示例

AI 应用开发知识体系建设: <https://framework.unitmesh.cc/patterns>

**question:** Semantic Workflow 是如何实现的?

**englishQuery:** How is Semantic Workflow implemented?

**originLanguageQuery:** Semantic Workflow 是如何实现的?

**hypotheticalCode:**

```
public class SemanticWorkflow {  
    private WorkflowEngine engine;  
  
    public SemanticWorkflow() {  
        engine = new WorkflowEngine();  
    }  
  
    public void executeWorkflow(Workflow workflow) {  
        // Perform semantic analysis on the workflow  
        SemanticAnalyzer analyzer = new  
        SemanticAnalyzer();  
        Workflow analyzedWorkflow =  
        analyzer.analyze(workflow);  
  
        // Execute the analyzed workflow using the  
        // workflow engine  
        engine.execute(analyzedWorkflow);  
    }  
}
```

## 上下文重排序 ReRank (Lost in the middle)

代码相近度评分 // 代码信息

- 0.7847863 // canonicalName:  
cc.unitmesh.cf.domains.semantic.CodeSemanticWorkflowTest
- 0.76635444 // canonicalName: cc.unitmesh.cf.domains.semantic.CodeSemanticDecl
- 0.74648994 // canonicalName: cc.unitmesh.cf.core.flow.ProblemAnalyzer
- 0.7410852 // canonicalName: cc.unitmesh.cf.domains.spec.SpecDomainDecl
- 0.72767156 // canonicalName: cc.unitmesh.cf.core.flow.DomainDeclaration
- 0.73245597 // canonicalName: cc.unitmesh.cf.core.flow.model.WorkflowResult
- 0.7434818 // canonicalName:  
cc.unitmesh.cf.domains.interpreter.CodeInterpreterWorkflow.execute
- 0.757218 // canonicalName: cc.unitmesh.cf.core.flow.Workflow
- 0.7722022 // canonicalName:  
cc.unitmesh.cf.domains.semantic.flow.SemanticProblemAnalyzer
- 0.807935 // canonicalName:  
cc.unitmesh.cf.domains.semantic.CodeSemanticWorkflow.execute

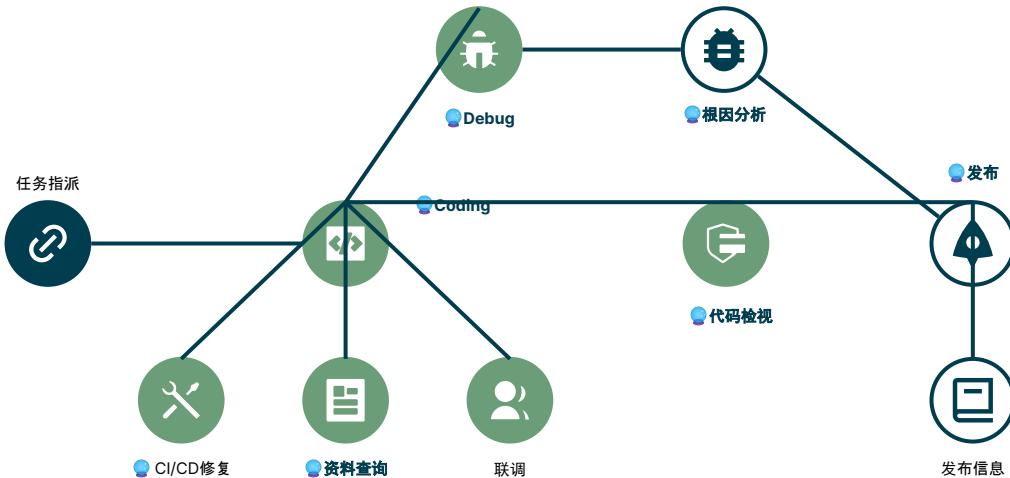
# 使用 Unit Mesh

使用 Unit Mesh 开源项目助力 AI 辅助  
研发落地

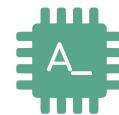


# 结合 Unit Mesh 落地辅助编程场景示例

遇到问题：<https://github.com/unit-mesh/auto-dev/issues>



1. 结合 AutoDev 自定义能力, 设计和优化流程
2. 结合开源模型, 在 AutoDev 上 定制和扩展
3. UnitGen 收集编码数据, 进行内 部代码微调



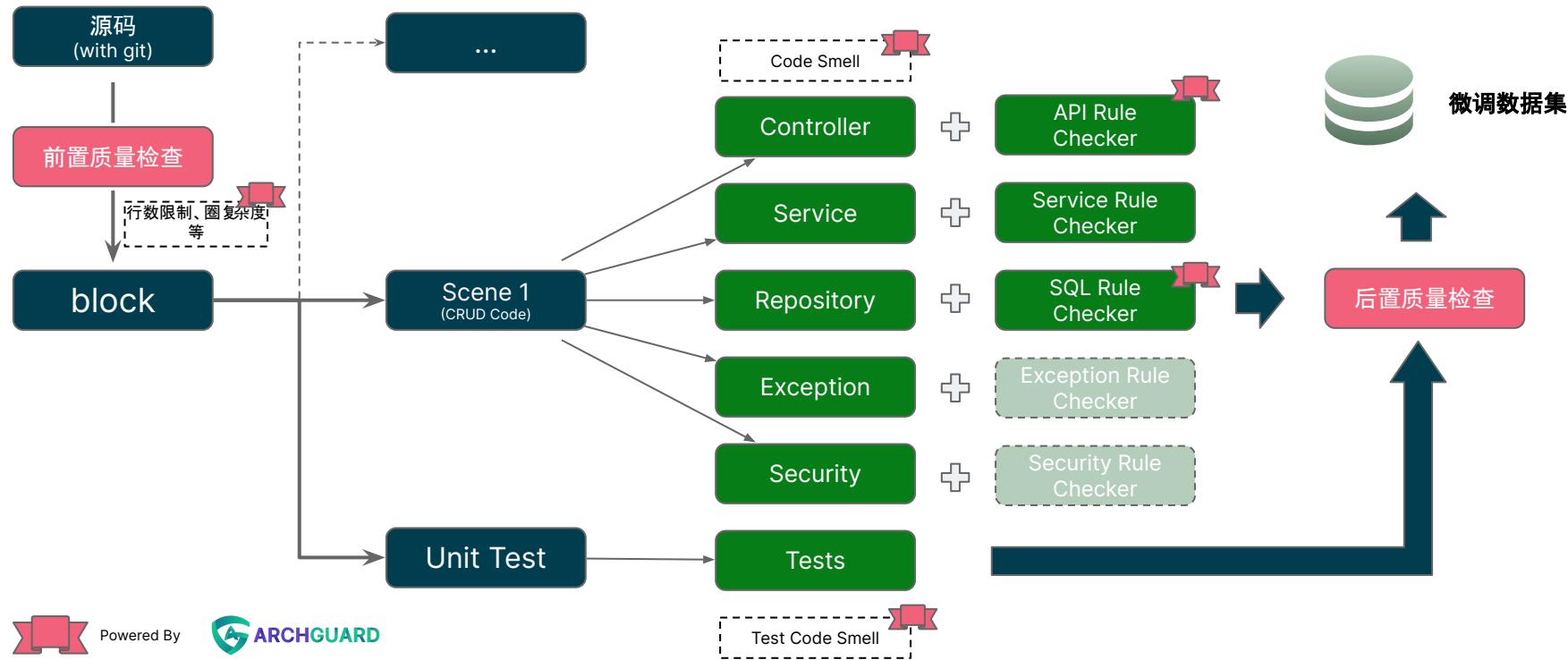
AutoDev



UnitGen

# UnitGen: 编码模型微调数据生成框架

内置静态分析 + 质量控制流水线，直接从代码库中生成高质量数据：代码补全、测试生成、文档生成等。

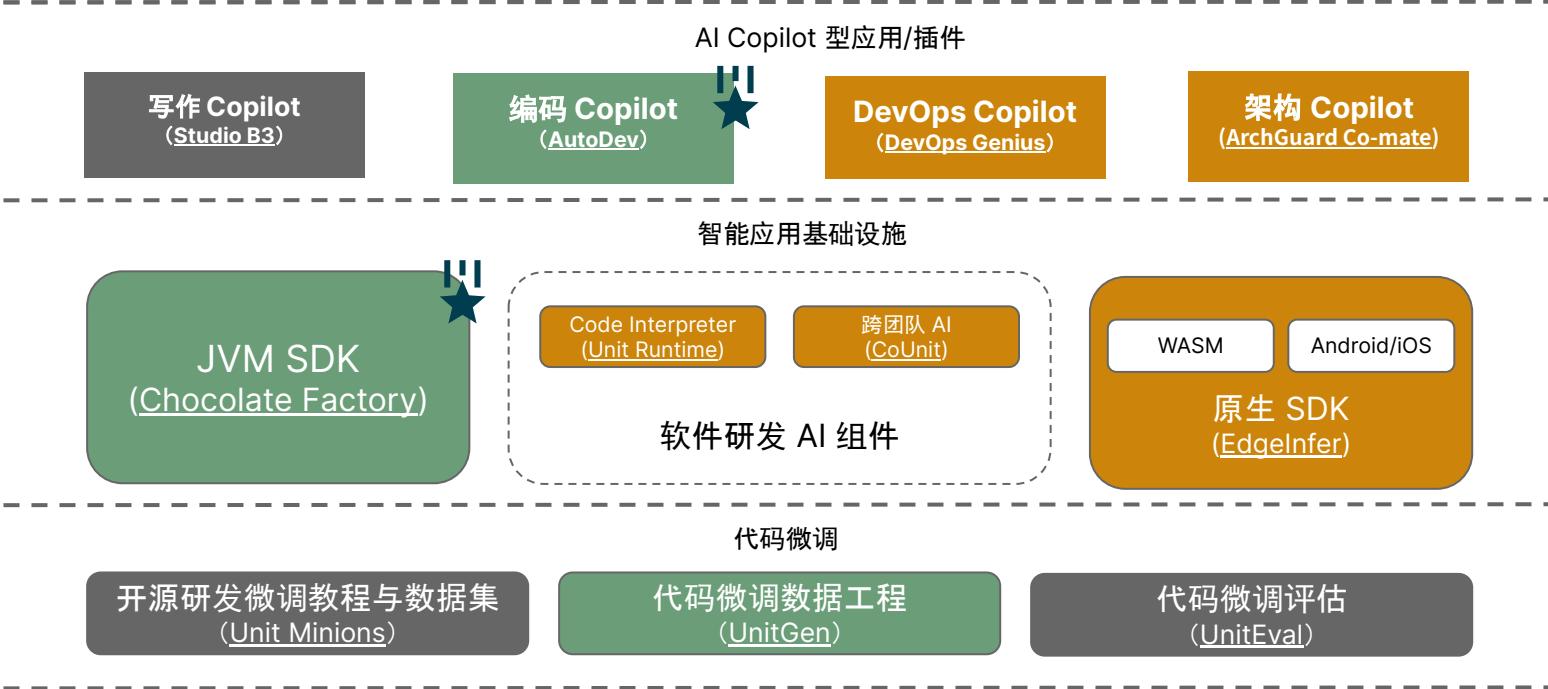


Powered By



# Thoughtworks AIGC 赋能研发开源方案: Unit Mesh

欢迎在 GitHub 上讨论: <https://github.com/unit-mesh>



# 谢谢聆听

有什么感想和问题吗?  
在B站或微信视频号留言哦



请大家扫码，备注“AI2.0”入群  
进行深入交流

# 实践Showcase #8

代码微调  
数据工程  
设计与实验

/thoughtworks

Strategy. Design. Engineering.



黄峰达/Phodal

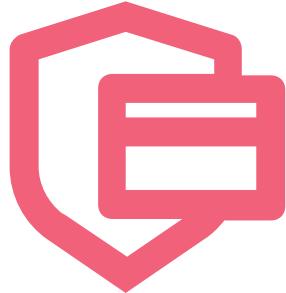
Thoughtworks 开源解决方案、生  
成式 AI 开源工具负责人

# 为什么考虑研发模型微调？

在训练好的 LLM 基础上，使用特定的数据集进一步的训练，以使模型适应特定任务或领域。

## 信息安全

敏感信息不能外泄



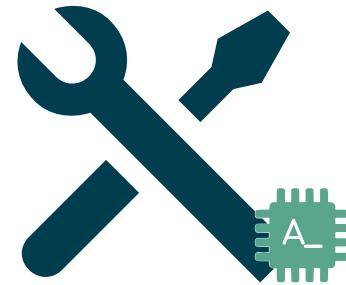
## 内部资产

开放模型基于公开的语料，缺少内部开发框架、业务知识等



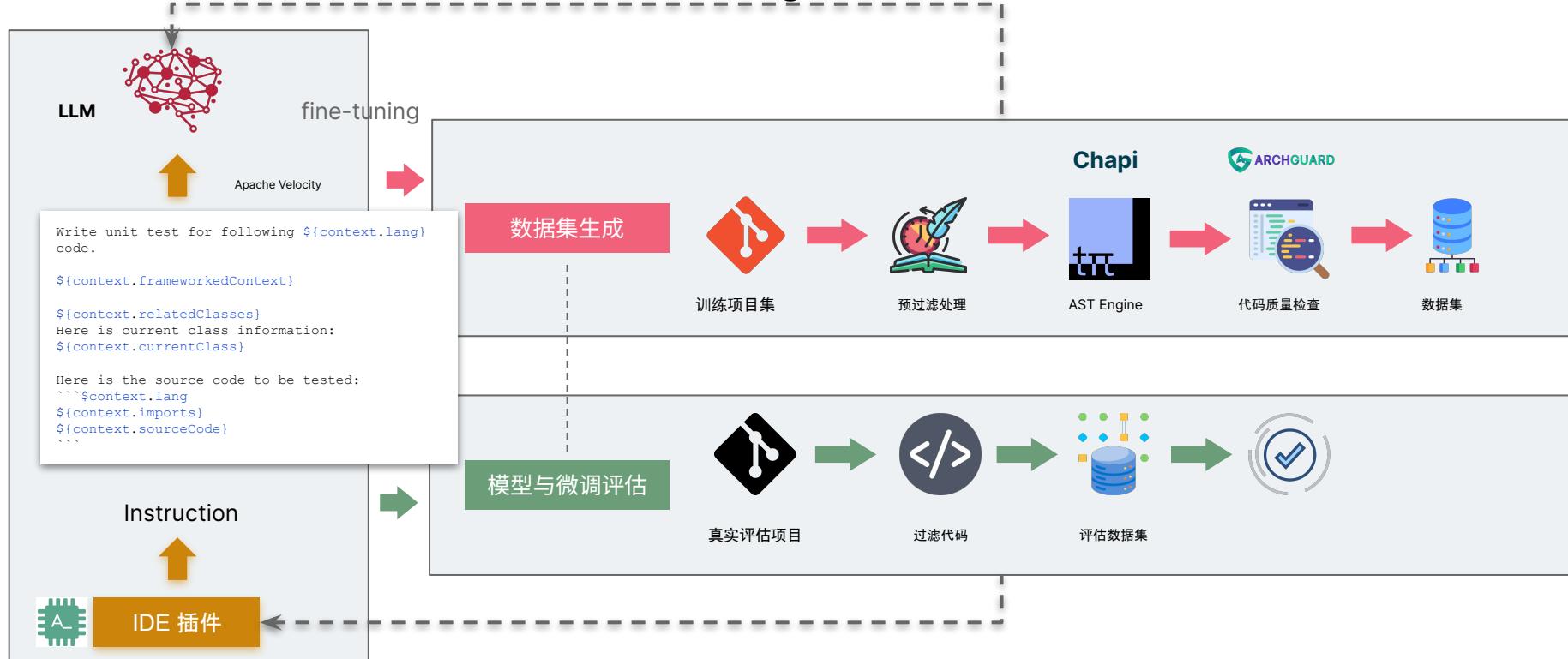
## 工具适配

适配工具的指令(prompt)和上下文理解需要



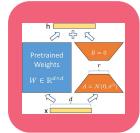
# 设计原则: 工具-微调-评估一体化设计

综合调整和评估 - (Use <-> Eval <-> Fine-tuning)



# 基本的微调准入条件

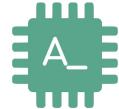
工具链探索 + 代码数据集项目构建 + 微调能力构建



能力

模型能力:微调与部署

模型评估与微调



形态

BizDevOps 流程与工具设计

流程与工具链设计



数据

研发资产数字化、DSL 化

微调数据集构建

# 能力 : 选择合适的编码模型

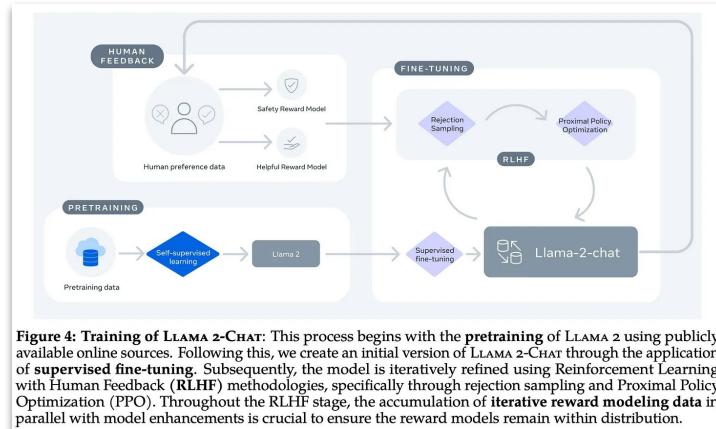
根据算力与场景，选择一大一小模型

场景驱动：

- 补全模型（高响应速度， 6~7B）
- 问答/聊天模型（高回答质量， xxxB+）
- **代码向量化模型**（代码问答， 按需）

自建（开源模型）还是买（商用解决方案）：

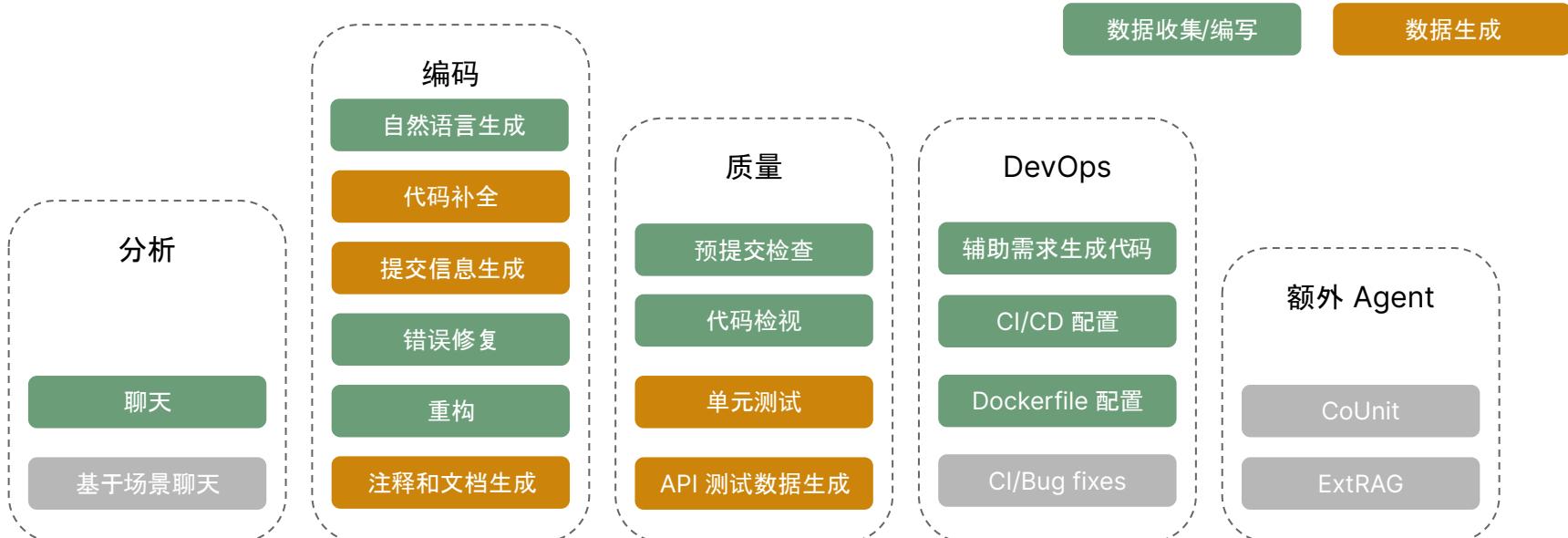
- 开源对研发能力要求较高（模型 + 数据 + 工具链）(如 LLaMa 架构，非 LLaMa 模型)
- 商用方案（定制化）



<https://cameronrwolfe.substack.com/p/llama-2-from-the-ground-up>

# 形态：围绕场景设计辅助功能

高质量的数据 => 借助高级工程师生成、编写数据集，改善数据集质量。



AutoDev 功能下的场景数据示例

# 数据生成:围绕上下文构建持续微调机制

不同场景和精准度，对数据的要求差异较大，如在测试场景下，结合项目数据集生成

```
import org.junit.jupiter.api.Test

@PhodalHuang
class TestFrameworkIdentifierTest {

    @PhodalHuang
    @Test
    fun shouldIdentifyJavaTestFrameworks() {
        // given
        val language = SupportedLang.JAVA
        val dependencies: List<CompositionDependency> = listOf(
            CompositionDependency.from(name: "junit:junit", group: "org.ju
            CompositionDependency.from(name: "org.mockito:mockito-core",

```

正确的测试框架

Junit4 还是 Junit 5 ⇒  
解析 maven, Gradle 信息

```
// response info:
class BlogPost {
    id: Long
    title: String
    content: String
    author: String
}
More lines

Java

public BlogPost createBlog(@RequestBody CreateBlogRequest request)
    CreateBlogResponse response = new CreateBlogResponse();

```

正确的函数输入和输出

静态代码分析

UnitGen 测试数据集生成上下文示例

```
@AutoConfigureMockMvc
@AutoConfigureTestDatabase
public class BlogControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private BlogService blogService;

    new *
    @Test
    public void shouldCreateBlog() throws Exception {

```

规范的测试用例

规范分析与处理

# 数据生成:AutoDev vs UnitGen 指令对比

Write unit test for following code.

You MUST use **should\_xx** style for test method name.

When testing controller, you MUST use MockMvc and test API only.

You are working on a project that uses Spring MVC, Spring WebFlux, JDBC to build RESTful APIs.

```
// class name: BookMeetingRoomRequest
// class fields: meetingRoomId
// class methods:
// super classes: [Object]

```java
@PostMapping("/{meetingRoomId}/book")
public ResponseEntity<BookMeetingRoomResponse>
bookMeetingRoom(@PathVariable String meetingRoomId, @RequestBody
BookMeetingRoomRequest request) {
    // 业务逻辑
    BookMeetingRoomResponse response = new BookMeetingRoomResponse();
    // 设置 response 的属性
    return new ResponseEntity<>(response, HttpStatus.CREATED);
}
```

```

Start test code with `@Test` syntax here:

IDE 端指令实现(AutoDev)

Write java language unit test for getSimilarityThreshold's method.

###  
- Test class should be named `CamelCase`. - - -  
- You are working on a project that uses Spring Boot, Spring Boot Web, Spring WebFlux, JPA SPRING BOOT, REACTIVE CASSANDRA, REACTIVE COUCHBASE, REACTIVE MONGO, Spring MVC  
- This project uses Spring Boot Test, Spring Test to test code.

Related code:

```
// Current file: // class SearchRequest {
// ...
//     + hasFilterExpression(): boolean
// }
//
```

Code under test:

```
```java
double getSimilarityThreshold() {
    return similarityThreshold;
}
```

```

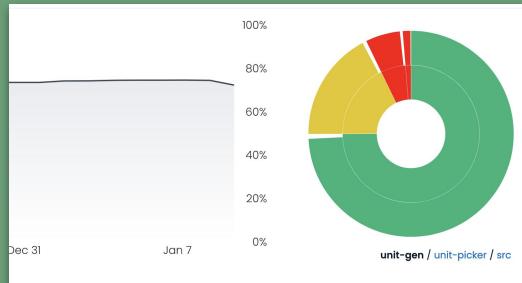
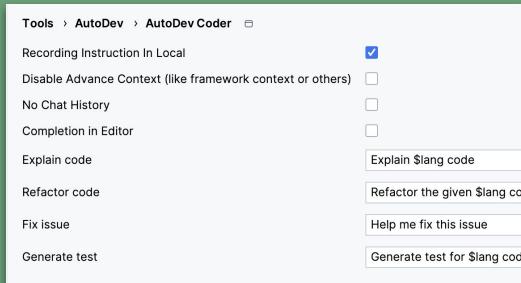
Start test code with `@Test` syntax here:

数据集指令实现(UnitGen)

# 数据收集: 构建其它场景的高质量数据

开源语料问题多、Python 语言倾向于强, 高质量研发相关数量集, 依然是稀缺

```
4
5  /**
6   * The 'JavaFrameworkIdentifier' class is a Kotlin implementation of the 'LangFrame
7   * It is responsible for identifying the test frameworks used in a Java project bas
8   *
9   * @param dependencies A list of 'CompositionDependency' objects representing the p
10  */
11 class JavaFrameworkIdentifier(private val dependencies: List<CompositionDependency>
12
13 /**
14  * A map that associates Java test framework dependencies with their correspondi
15  */
16 private val javaTestFrameworkMap: Map<String, List<String>> = mapOf(
17     "junit:junit" to listOf("JUnit"),
18     "org.testng:testng" to listOf("Testing"),
19     "org.spockframework:spock-core" to listOf("Spock"),)
```



让自己成为语料集

从 UnitGen 和 Chocolate Factory 中构建数据

半自动收集

基于公开模型 + 开源语料项目重构  
生成数据集

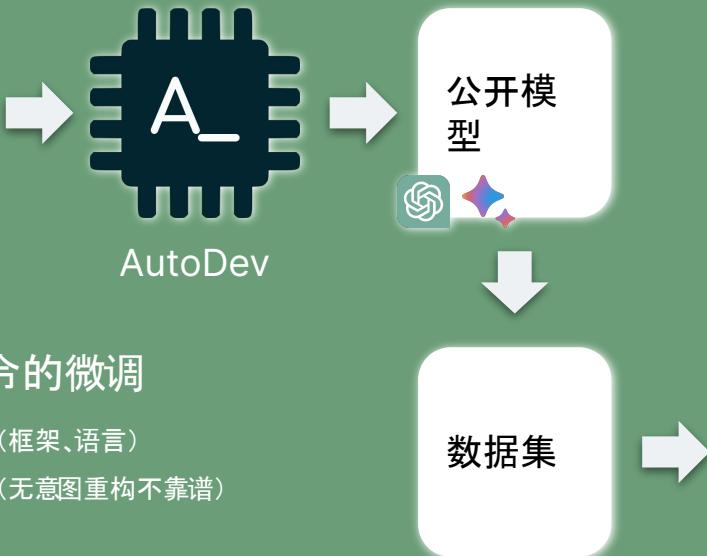
真实项目演进

选取真实项目作为功能试点, 收集  
过程数据(ArchGuard)

# 半自动收集：“肯尼亚工人 2.0”

在本地记录结果，手动筛选和过滤 ⇒ 100~200 条/天

- 代码重构
- 提交信息生成
- API 数据生成
- .....



## 结合工具指令的微调

- 特定上下文(框架、语言)
- 准确性依据(无意图重构不靠谱)
- .....

新闻截图展示了关于AI训练和使用方面的报道：

- 澎湃新闻**：AI的剥削：肯尼亚工人训练ChatGPT，看大量有害内容心理受伤
- 智源社区**：OpenAI雇佣肯尼亚工人 for chatGPT 清洗「有毒」数据
- 虎嗅网**：肯尼亚“喂养”GPT-4-虎嗅网

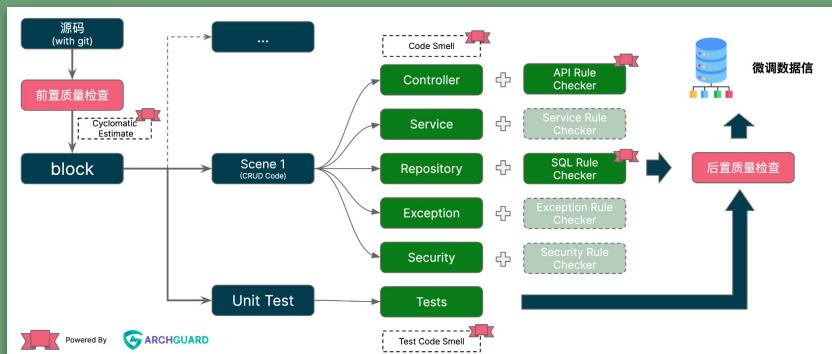
下方显示了虎嗅网文章的一部分文本，包含JSON指令和AI生成的Kotlin代码示例。

```
{"instruction": "Write", "instruction": "Refac", "instruction": "Write", "instruction": "Refac", "instruction": "Write", "instruction": "Refactor the given Kotlin code\nYou", {"instruction": "Write", "instruction": "Write a cohesive yet descriptive co", {"instruction": "Write", "instruction": "Write unit test for following code", {"instruction": "Write", "instruction": "Write a cohesive yet descriptive co", {"instruction": "Write", "instruction": "Write a cohesive yet descriptive co", {"instruction": "Write", "instruction": "Write unit test for following code", {"instruction": "Write", "instruction": "Write a cohesive yet descriptive co", {"instruction": "Write", "instruction": "Write unit test for following code", {"instruction": "Write", "instruction": "Write a cohesive yet descriptive commit message for a given diff. \n\nexamples", {"instruction": "Write", "instruction": "Write a cohesive yet descriptive commit message for a given diff. \n\nexamples":}
```

# **DEMO**

# 基于真实项目演进：收集过程数据

重构的数据蒸馏能 work 吗？



UnitGen 质量处理流程

**RefactoringMiner**



质量不过关的代码



公开模型



新数据

# 自动的数据集并不总是可靠的

当你需要 byExperience 时, 模型就无法理解

```
public double calculateInsurance (double income) {
    if (income <= 10000) {
        return income * 0.365;
    } else if (income <= 30000)
    {
        return (income - 10000)
* 0.2 + 35600;
    } else if (income <= 60000)
    {
        return (income - 30000)
* 0.1 + 76500;
    } else {
        return (income - 60000)
* 0.02 + 105600;
    }
}
```

JetBrains IDE 示例



```
java Copy code
public double calculateInsurance(double income) {
    final double BASE_RATE = 0.365;
    final double FIRST_RANGE_RATE = 0.2;
    final double SECOND_RANGE_RATE = 0.1;
    final double THIRD_RANGE_RATE = 0.02;

    final double FIRST_RANGE_THRESHOLD = 10000;
    final double SECOND_RANGE_THRESHOLD = 30000;
    final double THIRD_RANGE_THRESHOLD = 60000;

    double insurance;

    if (income <= FIRST_RANGE_THRESHOLD) {
        insurance = income * BASE_RATE;
    } else if (income <= SECOND_RANGE_THRESHOLD) {
        insurance = (income - FIRST_RANGE_THRESHOLD) * FIRST_RANGE_RATE;
    } else if (income <= THIRD_RANGE_THRESHOLD) {
        insurance = (income - SECOND_RANGE_THRESHOLD) * SECOND_RANGE_RATE;
    } else {
        insurance = (income - THIRD_RANGE_THRESHOLD) * THIRD_RANGE_RATE;
    }

    return insurance;
}
```

重构



```
java Copy code
public enum IncomeRange {
    RANGE_1(10000, 0.365, 0),
    RANGE_2(30000, 0.2, 35600),
    RANGE_3(60000, 0.1, 76500),
    RANGE_4(Double.MAX_VALUE, 0.02, 105600);

    private final double threshold;
    private final double rate;
    private final double base;

    IncomeRange(double threshold, double rate, double base) {
        this.threshold = threshold;
        this.rate = rate;
        this.base = base;
    }

    public double calculateInsurance(double income) {
        if (income <= threshold) {
            return income * rate + base;
        } else {
            return (income - threshold) * rate + base;
        }
    }
}

public class InsuranceCalculator {
    public double calculateInsurance(double income) {
        for (IncomeRange range : IncomeRange.values()) {
            if (income <= range.threshold) {
                return range.calculateInsurance(income);
            }
        }
        return 0;
    }
}
```

重构。结合 Enum

# 使用策略模式重构

也许应该换个思路，或许我们不需要重构？？？

```
java Copy code
class LowIncomeCalculator implements InsuranceCalculator {
    @Override
    public double calculate(double income) {
        return income * 0.365;
    }
}

class MediumIncomeCalculator implements InsuranceCalculator {
    @Override
    public double calculate(double income) {
        return (income - 10000) * 0.2 + 35600;
    }
}

class HighIncomeCalculator implements InsuranceCalculator {
    @Override
    public double calculate(double income) {
        return (income - 30000) * 0.1 + 76500;
    }
}

class VeryHighIncomeCalculator implements InsuranceCalculator {
    @Override
    public double calculate(double income) {
        return (income - 60000) * 0.02 + 105600;
    }
}
```



ChatGPT

这段代码看起来没有明显的错误，但有一些潜在的改进和注意事项：

1. 硬编码的数值：在方法中硬编码的数值（如10000和35600）可能会导致代码的可读性和维护性下降。建议将这些数值定义为类的常量或者通过参数传递进来，以便在将来容易进行修改。

```
java Copy code
public class MediumIncomeCalculator implements InsuranceCalculator {
    private static final double BASE_INCOME = 10000;
    private static final double BASE_CALCULATION = 35600;

    @Override
    public double calculate(double income) {
        return (income - BASE_INCOME) * 0.2 + BASE_CALCULATION;
    }
}
```

2. 注释缺失：没有注释来解释该方法的计算逻辑。添加一些注释可以帮助其他开发者理解代码的目的和计算方式。

```
java Copy code
/**
 * Calculates insurance based on medium income.
 * Formula: (income - BASE_INCOME) * 0.2 + BASE_CALCULATION
 */
public class MediumIncomeCalculator implements InsuranceCalculator {
    // ... (constants and other code)
}
```

3. 异常处理：如果在实际应用中，输入的收入可能会出现负值或其他不合法的情况，你

重构。结合策略模式

我们比以往任何时候  
都更加需要知识 ——  
开发工具的人需要复  
习什么好的代码、如  
何重构等等。

User  
重构为注册表模式

Java

```
public double calculateInsurance(double income) {
    if (income <= 10000) {
        return income * 0.365;
    } else if (income <= 30000) {
        return (income - 10000) * 0.2 + 35600;
    } else if (income <= 60000) {
    }
}
```

Assistant

Java

```
public double calculateInsurance(double income) {
    InsuranceCalculator calculator = new InsuranceCalculator();
    return calculator.calculate(income);
}

public interface InsuranceCalculator {
    double calculate(double income);
}
```

refactor(badsmell): extract if-else logic into separate classes

Extracted the if-else logic in the 'calculateInsurance' method into separate classes to improve readability and maintainability. Each bracket calculation is now handled by a separate class implementing the 'InsuranceCalculator' interface.

Assistant

这次代码重构将calculateInsurance方法中的if-else逻辑提取到了单独的类中，以提高可读性和可维护性。每个类都实现了InsuranceCalculator接口，并根据不同的收入档位计算保险费用。这种改变对核心功能和代码的整体结构产生了重大影响，使代码更加模块化和可扩展。潜在的问题或风险在于，如果未来需要添加更多的收入档位，可能需要修改现有的类或添加新的类，这可能会导致代码的复杂性增加。为了与现有代码库保持兼容性和一致性，重要的是确保新代码与现有代码遵循相同的标准和实践，以确保代码的统一性和长期可维护性。

人工校验的成本在不断提高，如何降低验证成本是下一个挑战。

# 谢谢聆听

有什么感想和问题吗?  
在B站或微信视频号留言哦



请大家扫码，备注“AI2.0”入群  
进行深入交流