

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分



# Agents for Coding and Software Development

From code completion to software engineering

**Carnegie Mellon University**  
Language Technologies Institute

**Graham Neubig**  
Language Technologies Institute

1

面向编码与软件开发的智能体

从代码补全走向软件工程

Graham Neubig

语言技术研究所 (Language Technologies Institute)

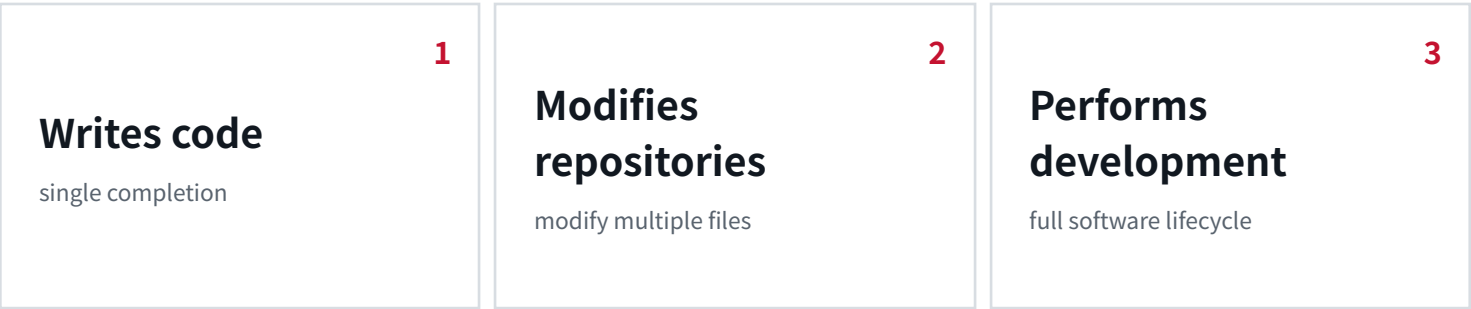
面向编码与软件开发的智能体

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Coding agents



2

编码智能体

- 1：编写代码——单次补全
- 2：修改代码仓库——修改多个文件
- 3：执行开发——覆盖完整的软件生命周期

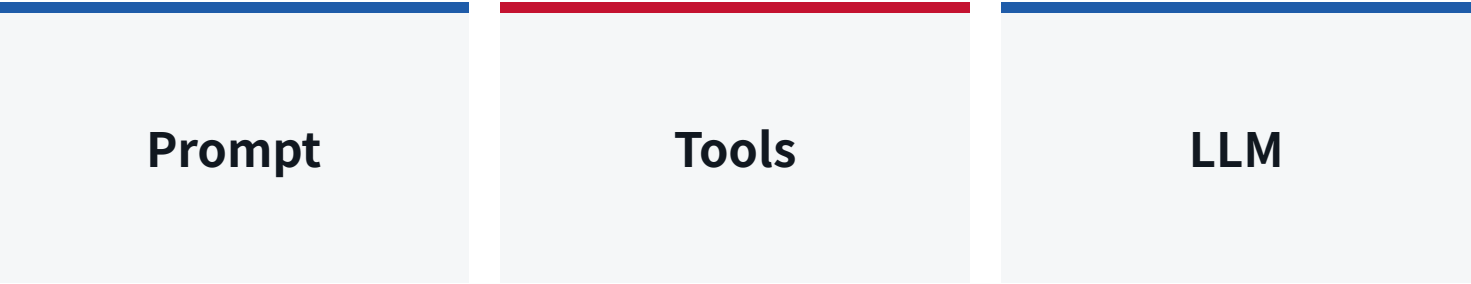
编码智能体

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Three ingredients



3

三项要素

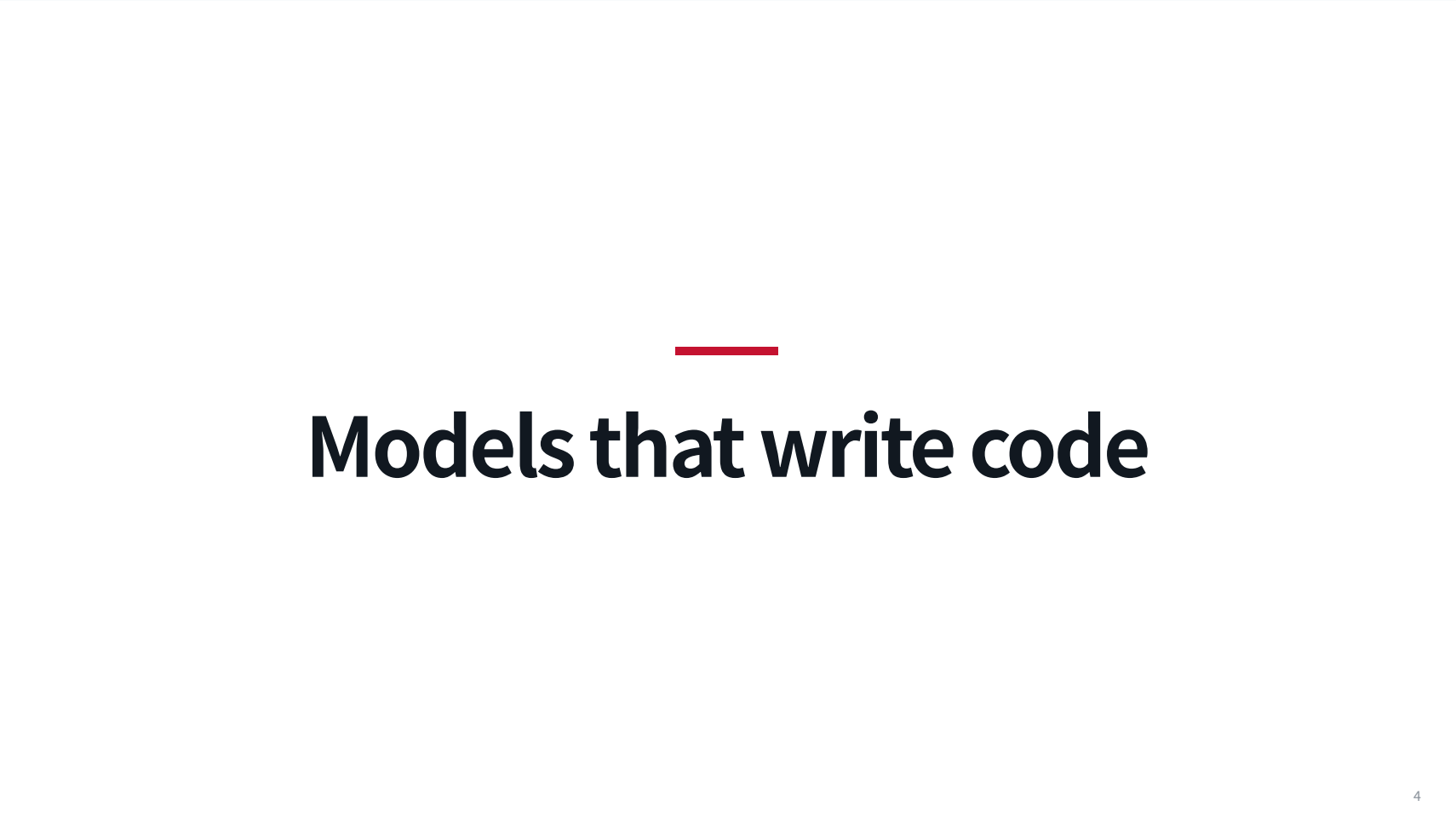
提示词 · 工具 · 大语言模型（LLM）

三项要素

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分



能够编写代码的模型

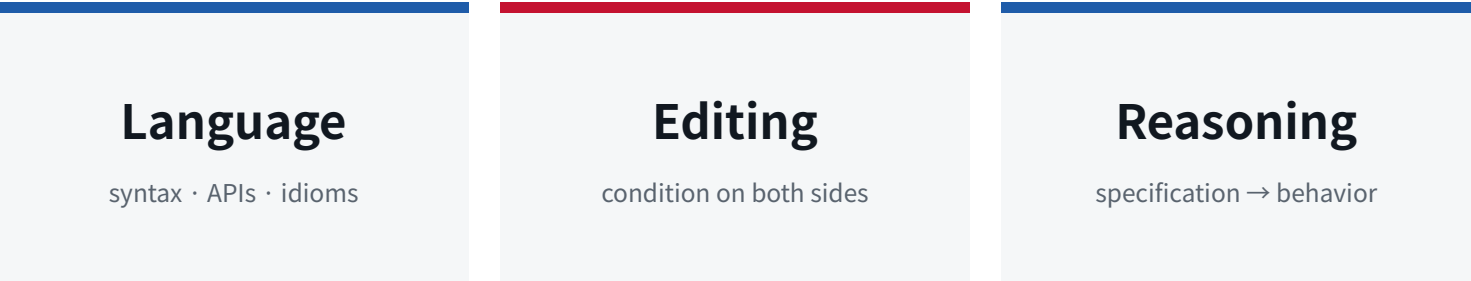
能够编写代码的模型

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Writing code from a prompt



5

根据提示词编写代码

语言：语法 · API · 惯用写法

编辑：同时以代码片段两侧为条件

推理：从规格说明推导出行为

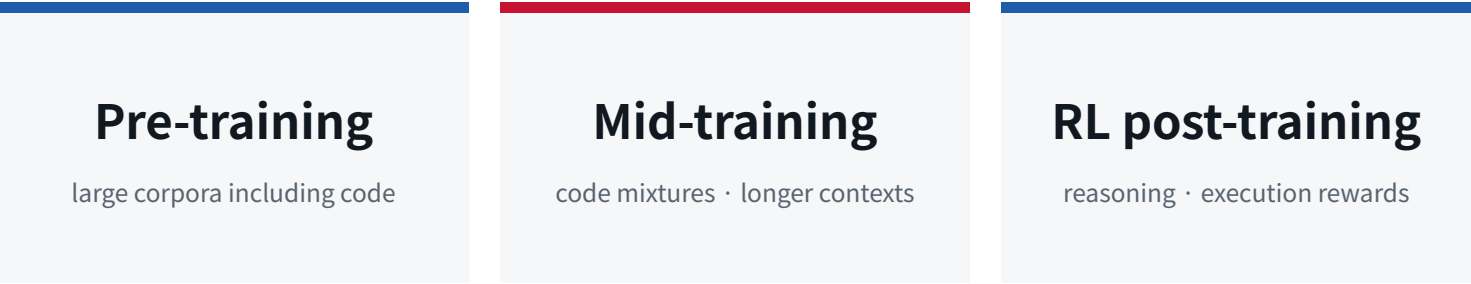
根据提示词编写代码

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

# Training a coding model



6

## 训练编码模型

预训练：包含代码的大型语料

中期训练：代码混合数据 · 更长上下文

强化学习后训练：推理 · 执行奖励

## 训练编码模型

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Pre-training: learning from code and text

- **Mix sources:** web text, technical prose, mathematics, source code, and documentation.
- **Preserve structure:** indentation, file boundaries, APIs, and text-code relationships.
- **Evaluate during training:** held-out loss by domain + functional coding tasks.

$$\mathcal{L}_{\text{LM}} = -\mathbb{E}_{x \sim D_{\text{mix}}} \sum_t \log p_{\theta}(x_t \mid x_{<t})$$

Groeneveld et al., 2024 · OLMo, § 2 ↗

预训练：从代码与文本中学习

混合来源：网页文本、技术文章、数学、源代码和文档。

保留结构：缩进、文件边界、API，以及文本与代码之间的关系。

训练期间评估：按领域划分的留出数据损失 + 功能性编码任务。

$$\text{LLM} = -\mathbb{E}_{x \sim D_{\text{mix}}} \sum_t \log p_{\theta}(x_t \mid x_{<t})$$

Groeneveld 等, 2024 · OLMo, 第 2 节 ↗

预训练：从代码与文本中学习

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

# Pre-training: choosing and cleaning data

- **Filter by language and source:** remove generated dumps, broken encodings, and low-value repetition.
- **Deduplicate:** forks, vendored libraries, near-identical files, and copied solutions.
- **Control provenance:** record licenses, source dates, opt-outs, and sensitive-data filtering.
- **Prevent leakage:** split by repository or problem family; decontaminate evaluation tasks.

StarCoder · data selection, filtering, and decontamination ↗

8

## 预训练：选择并清理数据

按语言和来源过滤：移除自动生成的数据转储、损坏的编码和低价值重复内容。

去重：代码分支、供应商库、近乎相同的文件，以及复制的解答。

控制来源记录：记录许可证、来源日期、退出请求，以及敏感数据过滤情况。

防止数据泄漏：按代码仓库或问题类别划分；对评估任务去污染。

StarCoder · 数据选择、过滤与去污染 ↗

## 预训练：选择并清理数据

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Pre-training: splitting code into tokens

- **Learn code-aware subwords:** byte-level BPE on code + prose; identifiers can span tokens.
- **Preserve whitespace:** spaces, tabs, and newlines carry syntax and literal content.
- **Compress frequent runs:** merge whitespace into tokens instead of spending one token per space.

Source

if·ready:↵····return·value↵

Tokens

if | ·ready | : | ↵··· | ·return | ·value | ↵

Actual StarCoder2 split · “·” = space, “↵” = newline, “|” = token boundary

预训练：将代码切分为词元

预训练：将代码切分为词元

学习代码感知型子词：对代码 + 文章使用字节级 BPE；标识符可能跨越多个词元。

保留空白：空格、制表符和换行符都承载语法和字面内容。

压缩常见连续片段：将空白合并为词元，而不是每个空格都占用一个词元。

源代码：if·ready:↵····return·value↵

词元：if | ·ready | : | ↵··· | ·return | ·value | ↵

实际 StarCoder2 切分 · “·” 表示空格， “↵” 表示换行， “|” 表示词元边界

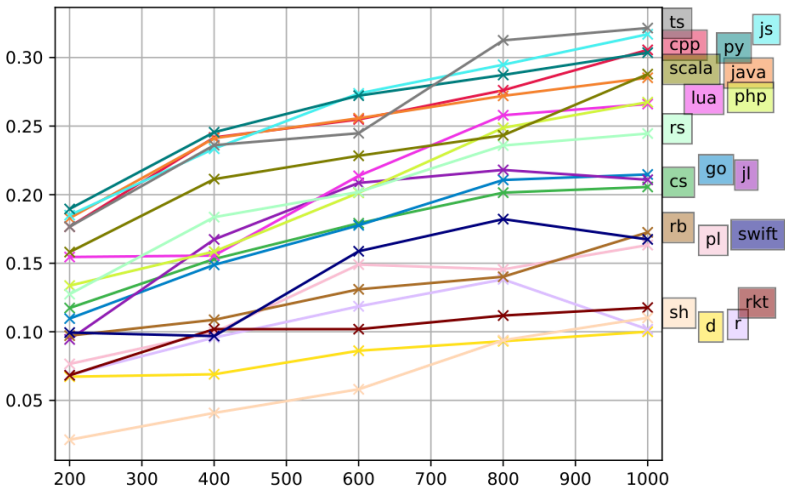
StarCoder，第 5.3 节 ↗ · Chen 等，2021 · 空白连续片段词元 ↗ · StarCoder2 词元分析器 ↗

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Pre-training on a large code corpus



**StarCoderBase**  
15.5B parameters  
1T tokens · 80+ languages  
Source, issues, commits, notebooks  
**Most languages improve with more tokens.**

Horizontal: training tokens (billions) · Vertical: pass@1 · Each curve: one language

Li et al., 2023 · StarCoder ↗

10

在大型代码语料上进行预训练

StarCoderBase: 155 亿个参数; 1T 个词元 · 80 多种语言; 数据包括源代码、问题、提交记录、笔记本。

大多数语言会随着词元数量增加而进步。

图表: 横轴为训练词元数 (十亿) ; 纵轴为 pass@1; 每条曲线代表一种语言。横轴刻度: 200、300、400、500、600、700、800、900、1000; 纵轴刻度: 0.05、0.10、0.15、0.20、0.25、0.30。

图中语言标签: ts、cpp、py、js、scala、java、lua、php、rs、cs、go、jl、rb、pl、swift、sh、d、r、rkt。

Li 等, 2023 · StarCoder ↗

在大型代码语料上进行预训练

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Mid-training: adding more code

- **Continue language-model training:** increase code exposure while retaining text and math.
- **Choose the mixture empirically:** compare code gains with general-capability retention.
- **Code Llama:** Llama 2 → 500B additional tokens for 7B / 13B / 34B models.
- **Then extend context:** train on related files and longer sequences with suitable positions.

[Rozière et al., 2023 · Code Llama, § 2 ↗](#)

中期训练：增加更多代码

继续进行语言模型训练：在保留文本和数学数据的同时，增加代码曝光量。

通过实证选择数据混合比例：比较代码能力提升与通用能力保留情况。

Code Llama：Llama 2 → 为 7B / 13B / 34B 模型额外训练 500B 个词元。

随后扩展上下文：使用合适的位置编码，在相关文件和更长序列上训练。

Rozière 等, 2023 · Code Llama, 第 2 节 ↗

中期训练：增加更多代码

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Mixing code, text, and math

| Code / text / math | Common code | MATH | MMLU |
|--------------------|-------------|------|------|
| 100 / 0 / 0        | 49.8        | 10.3 | 23.8 |
| 70 / 20 / 10       | 48.3        | 33.2 | 62.9 |

Hui et al., 2024 · Qwen2.5-Coder, Table 3 (selected columns) ↗

混合代码、文本与数学

代码 / 文本 / 数学 | 常见代码 | MATH | MMLU

100 / 0 / 0 | 49.8 | 10.3 | 23.8

70 / 20 / 10 | 48.3 | 33.2 | 62.9

Hui 等, 2024 · Qwen2.5-Coder, 表 3 (选列) ↗

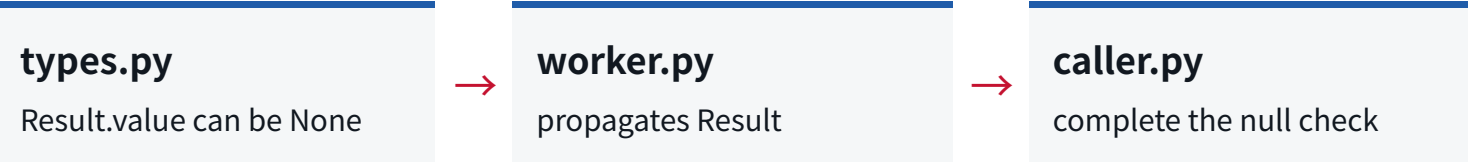
混合代码、文本与数学

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Mid-training: handling longer inputs



- **Coherent sequences:** related files, paths, and cross-file dependencies.
- **Positions + length:** Qwen2.5-Coder trains at 8K → 32K; YaRN enables 128K.
- **Validate:** cross-file completion, infilling, and short-context retention.

Qwen2.5-Coder · § 3.2.2; see also Lecture 3 ↗

中期训练：处理更长的输入

中期训练：处理更长的输入

types.py: Result.value 可能为 None → worker.py: 传递 Result → caller.py: 补全空值检查

连贯序列：相关文件、路径和跨文件依赖。

位置编码 + 长度：Qwen2.5-Coder 的训练长度为 8K → 32K；YaRN 支持 128K。

验证：跨文件补全、填空，以及短上下文能力保留。

Qwen2.5-Coder，第 3.2.2 节；另见第 3 讲 ↗

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Infilling: using code before and after a gap

Fill the return-type annotation

```
def is_positive(x: int) -> ____:
    return x > 0
```

The body is to the right of the hole.

Use both sides to propose the span

```
def is_positive(x: int) -> bool:
    return x > 0
```

The suffix supplies evidence for bool .

Fried et al. · InCoder, § 2 and § 4 · illustrative example ↗

填空：利用缺口两侧的代码

填空：利用缺口两侧的代码

补全返回类型注解：def is\_positive(x: int) -> \_\_\_\_: / return x > 0

两侧都用于提出填入片段：def is\_positive(x: int) -> bool: / return x > 0

缺口右侧就是函数体。

后缀为 bool 提供了证据。

Fried 等 · InCoder, 第 2 节与第 4 节 · 示例仅用于说明 ↗

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

# Infilling: marking and predicting gaps

|           |                                         |
|-----------|-----------------------------------------|
| Original  | prefix span suffix                      |
| Training  | prefix [M0] suffix [M0] span [END]      |
| Inference | prefix [M0] suffix [M0] → generate span |

- **One or more holes:** distinct sentinels associate each span with its location.
- **Zero-shot edits:** complete code, infer types, generate comments, rename variables.

Fried et al. · InCoder (2022 preprint; ICLR 2023), § 2 and § 4 ↗

填空：标记并预测缺口

原文：prefix span suffix

训练：prefix [M0] suffix [M0] span [END]

推理：prefix [M0] suffix [M0] → 生成 span

一个或多个缺口：不同的哨兵标记会把每个片段与其位置关联起来。

零样本编辑：补全代码、推断类型、生成注释、重命名变量。

Fried 等 · InCoder (2022 年预印本; ICLR 2023) , 第 2 节与第 4 节 ↗

填空：标记并预测缺口

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Infilling: the value of right context

Completed functions passing tests (%)

| Inference method              | Suffix used for | Single-line | Multi-line |
|-------------------------------|-----------------|-------------|------------|
| Left-to-right · 1 candidate   | Not used        | 48.2%       | 24.9%      |
| Left-to-right · 10 candidates | Reranking only  | 54.9%       | 28.2%      |
| Causal-masked infilling       | Generation      | 69.0%       | 38.6%      |

Fried et al. · InCoder, § 4.1 · HumanEval-derived infilling tasks ↗

填空：右侧上下文的价值

填空：右侧上下文的价值

通过测试的已完成函数 (%)

推理方法 | 使用后缀的方式 | 单行 | 多行

从左到右 · 1 个候选 | 不使用 | 48.2% | 24.9%

从左到右 · 10 个候选 | 仅用于重排 | 54.9% | 28.2%

因果掩码填空 | 生成时使用 | 69.0% | 38.6%

Fried 等 · InCoder, 第 4.1 节 · 基于 HumanEval 的填空任务 ↗

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

# Infilling: preserving code completion

Pass@1 · matched 52B-token training budget

| Training objective              | HumanEval | MBPP  |
|---------------------------------|-----------|-------|
| Left-to-right language modeling | 6.0%      | 8.9%  |
| Causal masking                  | 8.0%      | 10.9% |

Fried et al. · InCoder, § 5 · objective ablation ↗    Related scaling study: Bavarian et al., 2022 ↗

填空：保留代码补全能力

pass@1 · 匹配 52B 个词元的训练预算

训练目标 | HumanEval | MBPP

从左到右的语言建模 | 6.0% | 8.9%

因果掩码 | 8.0% | 10.9%

Fried 等 · InCoder, 第 5 节 · 目标函数消融 ↗ · 相关扩展研究: Bavarian 等, 2022 ↗

填空：保留代码补全能力

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

# Beyond infilling: more ways to learn from code

## Commit diffs

before + message → after / diff

[Muennighoff et al., 2023 · OctoPack ↗](#)

## Diagnostics

broken code + error → repair

[Yasunaga & Liang, 2020 · DrRepair ↗](#)

## Tests

program + test outcomes → reward

[Le et al., 2022 · CodeRL ↗](#)

## Execution traces

program → executed lines + states

[Liu et al., 2023 · CodeExecutor ↗](#)

18

超越填空：从代码中学习的更多方式

提交差异：修改前 + 消息 → 修改后 / 差异 (diff)

诊断信息：有缺陷的代码 + 错误 → 修复

测试：程序 + 测试结果 → 奖励

执行轨迹：程序 → 已执行的代码行 + 状态

Muennighoff 等, 2023 · OctoPack ↗ · Yasunaga 与 Liang, 2020 · DrRepair ↗ · Le 等, 2022 · CodeRL ↗ · Liu 等, 2023 · CodeExecutor ↗

超越填空：从代码中学习的更多方式

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

评估生成的代码

Evaluating Generated Code

19

评估生成的代码

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

# Comparing with a reference solution

- **Exact match:** does generated code reproduce the reference?
- **BLEU-4:** measure token n-gram overlap, with a brevity penalty.

| Integer x: is it positive? | Code                                | Behavior                |
|----------------------------|-------------------------------------|-------------------------|
| Reference                  | <code>return x &gt; 0</code>        | Specification           |
| Small textual change       | <code>return x &gt;= 0</code>       | Wrong at x = 0          |
| Different expression       | <code>return not (x &lt;= 0)</code> | Equivalent for integers |

[Yin & Neubig, ACL 2017 · § 5.1 ↗](#) [Ren et al., 2020 · CodeBLEU ↗](#)

## 与参考解答进行比较

与参考解答进行比较

精确匹配：生成的代码是否复现参考解答？

BLEU-4：衡量词元 n 元组的重合度，并加入简短度惩罚。

整数 x：它是正数吗？

代码 | 行为

参考解答 | `return x > 0` | 规格说明

小幅文本修改 | `return x >= 0` | 在 x = 0 时错误

不同表达式 | `return not (x <= 0)` | 对整数而言等价

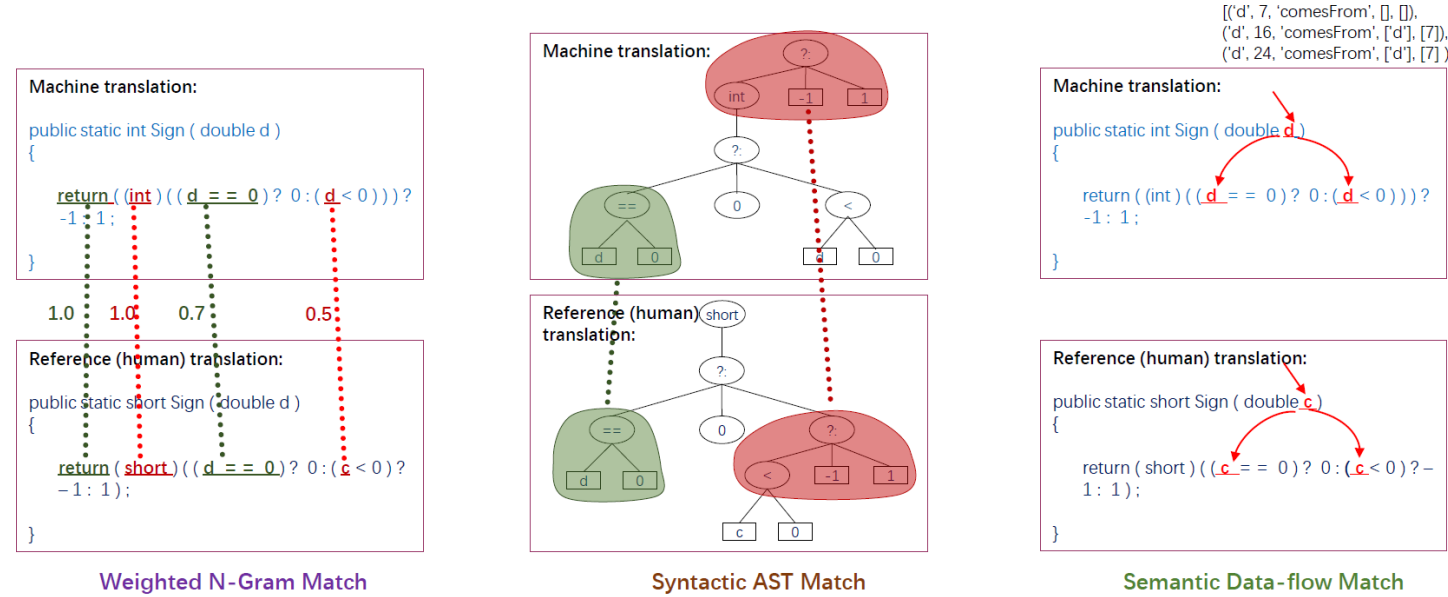
Yin 与 Neubig, ACL 2017, 第 5.1 节 ↗ · Ren 等, 2020 · CodeBLEU ↗

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Comparing code structure



比较代码结构

机器翻译示例：public static int Sign(double d) { return ((int)((d == 0) ? 0 : (d < 0)) ? -1 : 1); }; 人工参考翻译：public static short Sign(double d) { return (short)((d == 0) ? 0 : (d < 0) ? -1 : 1); }

加权 N 元组匹配

语法抽象语法树 (AST) 匹配

语义数据流匹配

CodeBLEU =  $\alpha$  · N 元组匹配 (BLEU) +  $\beta$  · 加权 N 元组匹配 +  $\gamma$  · 语法 AST 匹配 +  $\delta$  · 语义数据流匹配

中间图示以语法树标出机器翻译与人工参考之间的节点差异；右侧图示比较变量 d 与 c 的数据流关系。

Ren 等, 2020 · CodeBLEU · 原始概览图 ↗

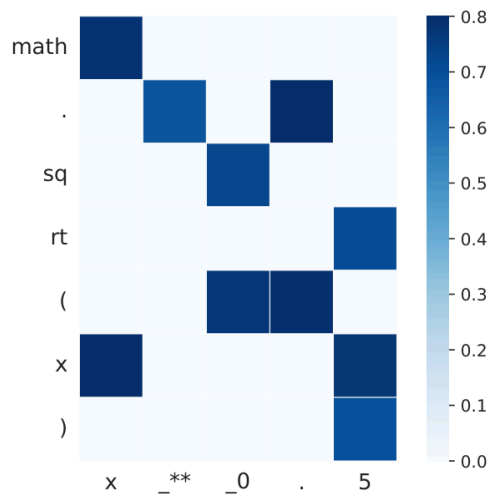
比较代码结构

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Comparing learned code representations



CodeBERTScore

- Encode prompt + each code snippet
- Compare contextual token vectors
- Best matches → precision / recall → F-score

Zhou et al., EMNLP 2023 · CodeBERTScore · original similarity heatmap ↗

22

比较学习得到的代码表示

CodeBERTScore：对提示词和每段代码片段分别编码；比较上下文词元向量；最佳匹配 → 精确率 / 召回率 → F 分数。

相似度热图：纵向词元 math、.、sq、rt、(、x、)；横向词元 x、\*\*、\_0、.、5；颜色刻度为 0.0 至 0.8。

Zhou 等, EMNLP 2023 · CodeBERTScore · 原始相似度热图 ↗

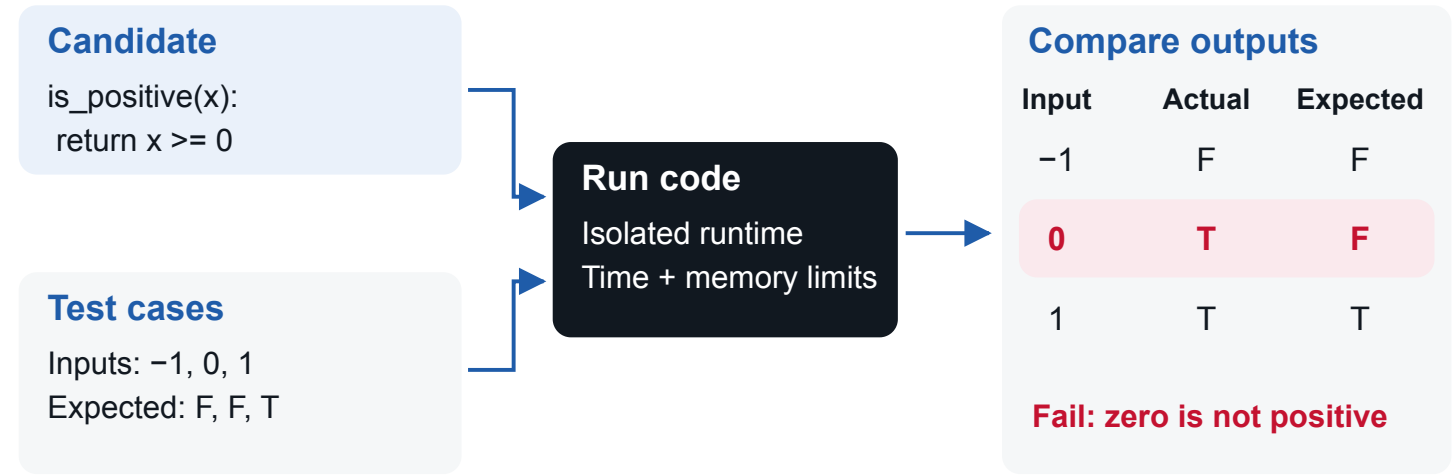
比较学习得到的代码表示

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Checking behavior by running code



Chen et al., 2021 · functional evaluation · classroom example ↗

23

通过运行代码检查行为

候选代码：is\_positive(x): return x >= 0

测试用例：输入 -1、0、1；预期结果：F、F、T。

运行代码：隔离运行环境；时间与内存限制。

比较输出：输入 | 实际 | 预期

-1 | F | F

0 | T | F

1 | T | T

失败：零不是正数。

Chen 等, 2021 · 功能评估 · 课堂示例 ↗

通过运行代码检查行为

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Benefits and limits of execution checks

| Benefit                                   | Disadvantage / limit                                      |
|-------------------------------------------|-----------------------------------------------------------|
| Accepts different correct implementations | Finite tests can miss incorrect behavior                  |
| Detects semantic errors through execution | Needs trusted tests, dependencies, and isolated execution |
| Matches observed behavior                 | Requires time and a stable execution environment          |

Chen et al., 2021 · functional evaluation ↗    Liu et al., 2023 · EvalPlus / test adequacy ↗

执行检查的好处与局限

好处 | 缺点 / 局限

接受不同的正确实现 | 有限的测试可能漏掉错误行为

通过执行检测语义错误 | 需要可信测试、依赖项和隔离执行

与观察到的行为相匹配 | 需要时间和稳定的执行环境

Chen 等, 2021 · 功能评估 ↗ · Liu 等, 2023 · EvalPlus / 测试充分性 ↗

执行检查的好处与局限

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

# Evaluating functions and full programs

## HumanEval

164 Python functions  
docstring → completion

hidden unit tests

## CodeContests

competitive problems · includes Codeforces  
statement → full program

compile + test

[HumanEval ↗](#) [CodeContests / AlphaCode ↗](#)

25

### 评估函数与完整程序

HumanEval: 164 个 Python 函数; 从文档字符串补全代码; 隐藏的单元测试。

CodeContests: 竞赛题目 · 包含 Codeforces; 从题面生成完整程序; 编译 + 测试。

[HumanEval ↗](#) · [CodeContests / AlphaCode ↗](#)

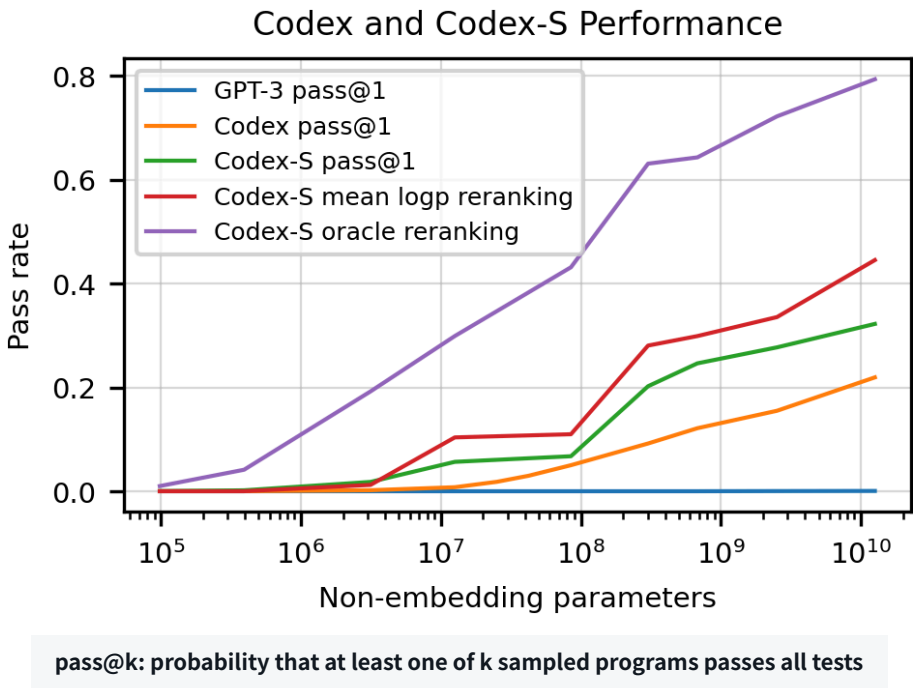
### 评估函数与完整程序

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Success across multiple attempts



Chen et al., 2021 · HumanEval ↗

26

多次尝试中的成功率

Codex 与 Codex-S 的表现

图表：纵轴为通过率；横轴为非嵌入参数，刻度 10<sup>5</sup>、10<sup>6</sup>、10<sup>7</sup>、10<sup>8</sup>、10<sup>9</sup>、10<sup>10</sup>。  
曲线：GPT-3 pass@1、Codex pass@1、Codex-S pass@1、Codex-S 平均 logp 重排、Codex-S oracle 重排。

pass@k：在采样的 k 个程序中，至少有一个程序通过全部测试的概率。

Chen 等, 2021 · HumanEval ↗

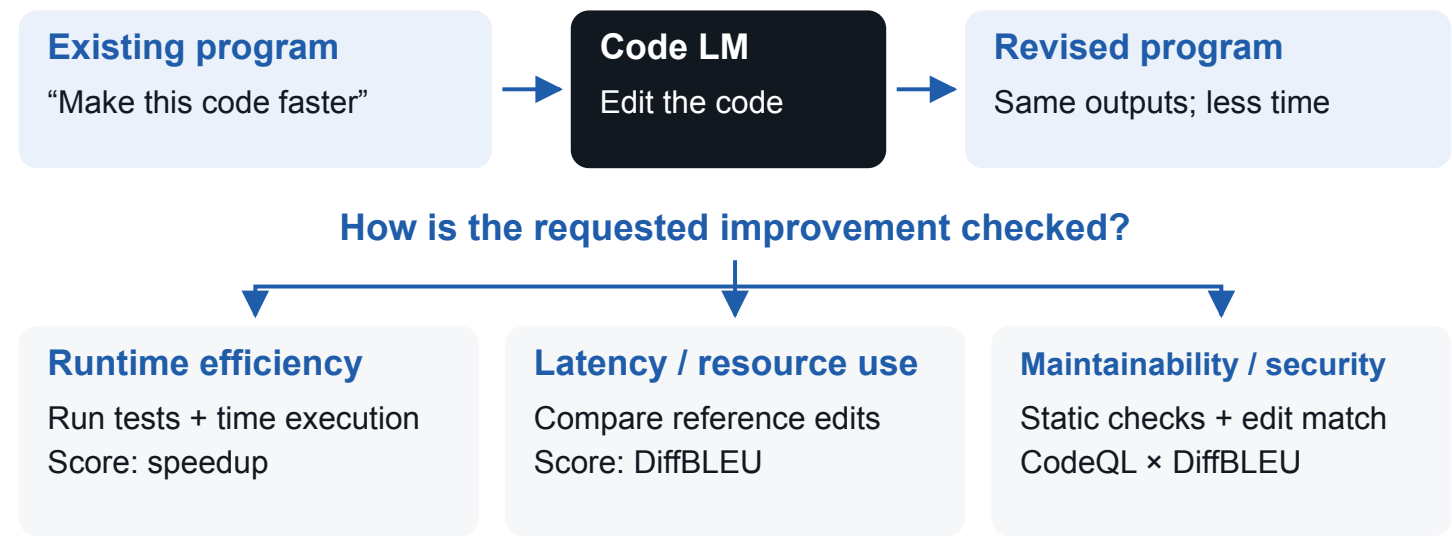
多次尝试中的成功率

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Non-functional requirements



Singhal et al., 2024 · NoFunEval · 397 code-editing tasks ↗

27

非功能性需求

现有程序：“让这段代码更快” → Code LM：编辑代码 → 修改后的程序：输出相同；耗时更少。

如何检查所要求的改进？

运行效率：运行测试 + 计时执行；评分：加速比。

延迟 / 资源使用：比较参考修改；评分：DiffBLEU。

可维护性 / 安全性：静态检查 + 编辑匹配；CodeQL × DiffBLEU。

Singhal 等, 2024 · NoFunEval · 397 个代码编辑任务 ↗

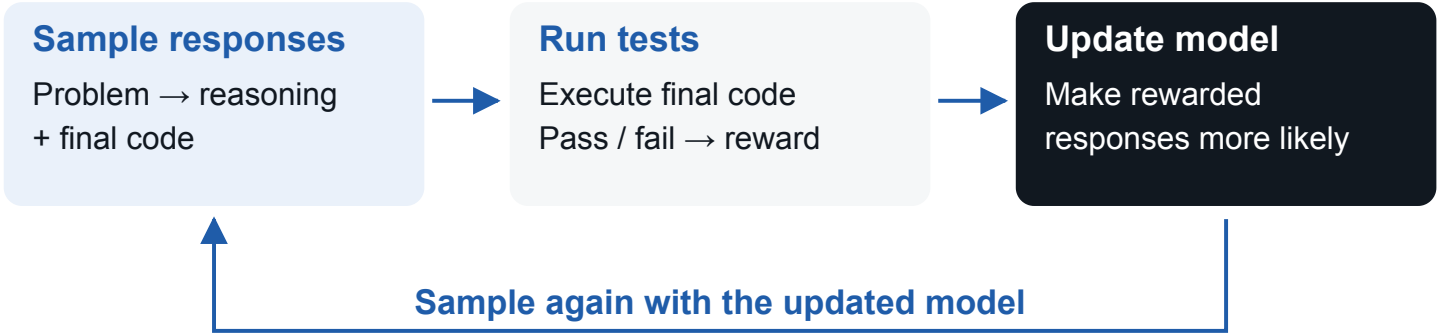
非功能性需求

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Learning from test results



[Luo et al., 2025 · DeepCoder · execution-reward RL ↗](#)

28

从测试结果中学习

采样回答：问题 → 推理 + 最终代码 → 运行测试：执行最终代码，测试通过 / 失败 → 奖励 → 更新模型：让得到奖励的回答更可能出现。

使用更新后的模型再次采样。

Luo 等, 2025 · DeepCoder · 基于执行奖励的 RL ↗

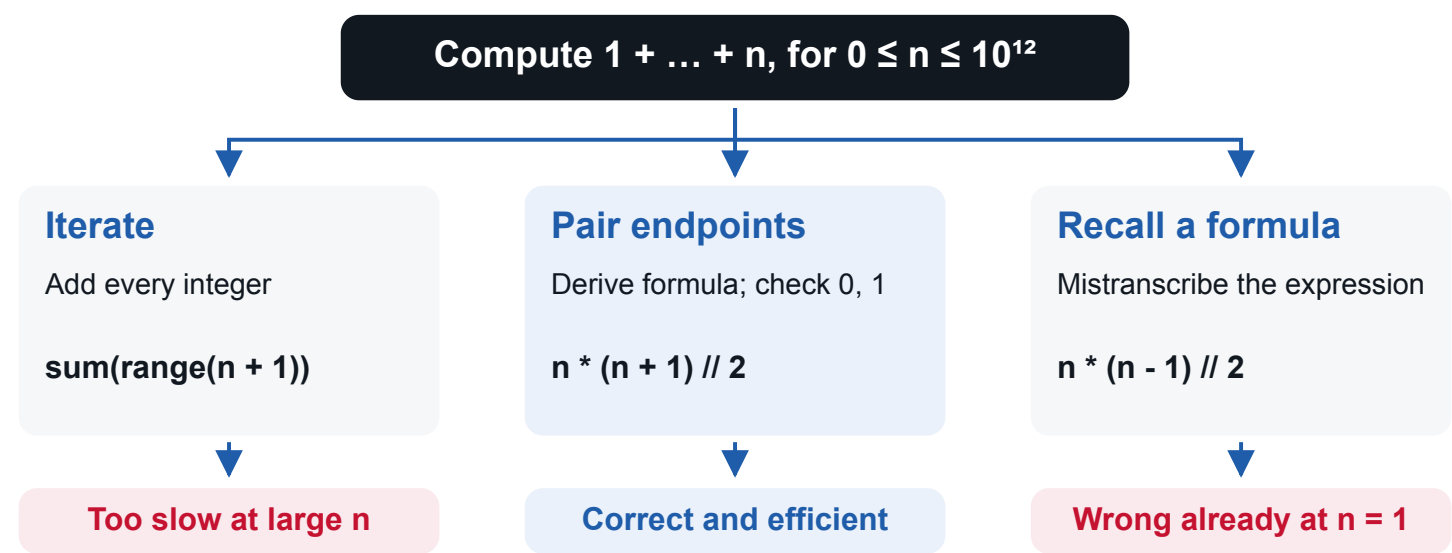
从测试结果中学习

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Reasoning toward a solution



29

推理以得出解法

计算  $1 + \dots + n$ ，其中  $0 \leq n \leq 10^{12}$ 。

逐项迭代：累加每个整数；`sum(range(n + 1))`；当  $n$  很大时太慢。

配对端点：推导公式；检查  $n = 0, 1$ ； $n * (n + 1) // 2$ ；正确且高效。

回忆公式：把表达式抄错； $n * (n - 1) // 2$ ；在  $n = 1$  时就错了。

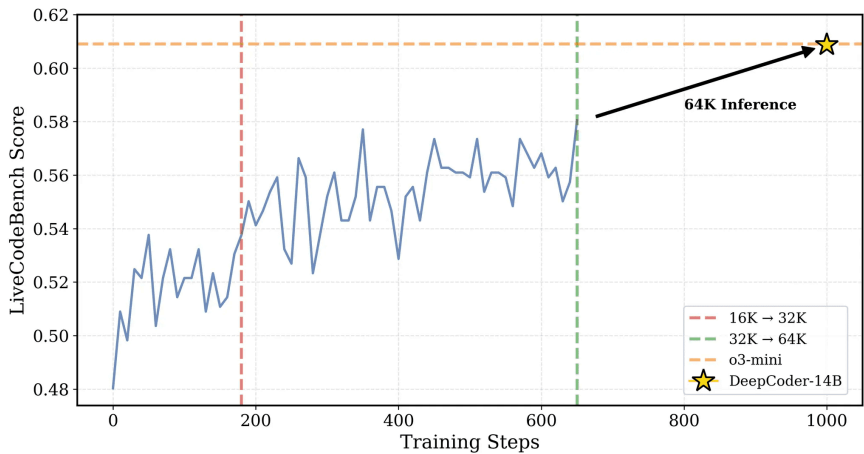
推理以得出解法

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Coding accuracy after reinforcement learning



16K → 32K: response-token limit during RL (K = 1,000 tokens).  
★ / 64K: more tokens at evaluation, no further RL. Orange dashed line: o3-mini.

[Luo et al., 2025 · DeepCoder · LiveCodeBench coding tasks ↗](#)

30

强化学习后的编码准确率

LiveCodeBench 得分；横轴：训练步数，纵轴：LiveCodeBench 得分。图例：16K → 32K；32K → 64K；o3-mini；DeepCoder-14B。

16K → 32K：RL 期间的回答词元上限（K = 1,000 个词元）。

★ / 64K：评估时使用更多词元，不再继续 RL。橙色虚线：o3-mini。

Luo 等, 2025 · DeepCoder · LiveCodeBench 编码任务 ↗

强化学习后的编码准确率

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

智能体式编码



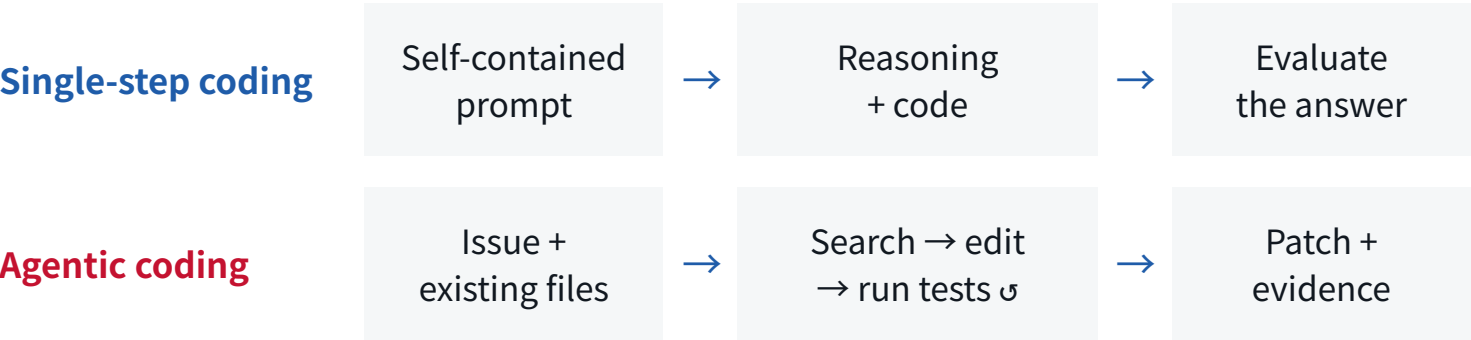
智能体式编码

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

From single-step to agentic coding



32

从单步编码走向智能体式编码

单步编码：自包含提示词 → 推理 + 代码 → 评估答案。

智能体式编码：问题 + 现有文件 → 搜索 → 编辑 → 运行测试 ∪ → 补丁 + 证据。

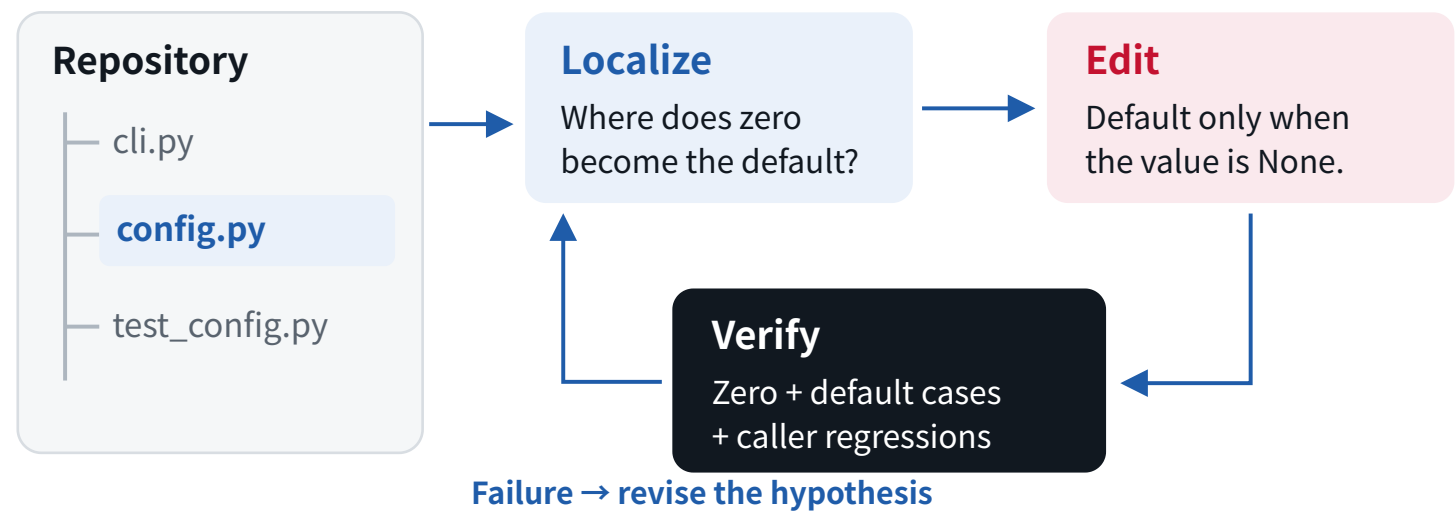
从单步编码走向智能体式编码

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

The localize–edit–verify loop



Interaction-loop concepts: SWE-agent · original teaching diagram ↗

33

定位—编辑—验证循环

代码仓库：cli.py、config.py、test\_config.py。

定位：零值在哪里变成默认值？

编辑：仅当值为 None 时使用默认值。

验证：零值 + 默认值场景 + 调用方回归。

失败 → 修正假设。

交互循环概念：SWE-agent · 原始教学示意图 ↗

定位—编辑—验证循环

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Localize: find the cause

Localize the implementation

```
$ rg 'retries' buggy_config.py
return config.get("retries") or 3
```

Zero is falsy: inspect the defaulting rule.

Reproduce the behavior

```
$ RETRY_MODULE=buggy_config python3 \
-m unittest test_config.RetryTests.test_zero
AssertionError: 3 != 0
FAILED (failures=1)
```

Now repair the condition, not the test.

定位：找出原因

定位实现：\$ rg 'retries' buggy\_config.py; return config.get("retries") or 3

重现行为：\$ RETRY\_MODULE=buggy\_config python3 -m unittest test\_config.RetryTests.test\_zero; AssertionError: 3 != 0; FAILED (failures=1)

零值是 falsy：检查默认值规则。

现在修复条件，而不是测试。

定位：找出原因

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

# Edit: change the defaulting rule

## Before

```
return config.get("retries") or 3
```

All falsy values select the default.

## After

```
value = config.get("retries")
return 3 if value is None else value
```

An explicit zero now survives.

35

编辑：修改默认值规则

修改前：return config.get("retries") or 3

所有 falsy 值都会选用默认值。

修改后：value = config.get("retries"); return 3 if value is None else value

显式设置的零值现在会被保留。

编辑：修改默认值规则

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Verify: check the fix and regressions

Run the existing tests

```
$ python3 -m unittest -v test_config

missing    → 3    PASS
None       → 3    PASS
zero       → 0    PASS
positive   → 2    PASS

Ran 4 tests: OK
```

Use failures to guide the next step

- Zero still becomes three?**  
Localize where the value changes.
- A default case breaks?**  
Edit the rule, then verify again.

36

验证：检查修复与回归

运行现有测试：\$ python3 -m unittest -v test\_config

missing → 3：通过；None → 3：通过；zero → 0：通过；positive → 2：通过；运行了 4 项测试：全部通过。

使用失败信息指导下一步：零值仍变成 3？定位值在哪里发生变化。

默认值场景出错？编辑规则，然后重新验证。

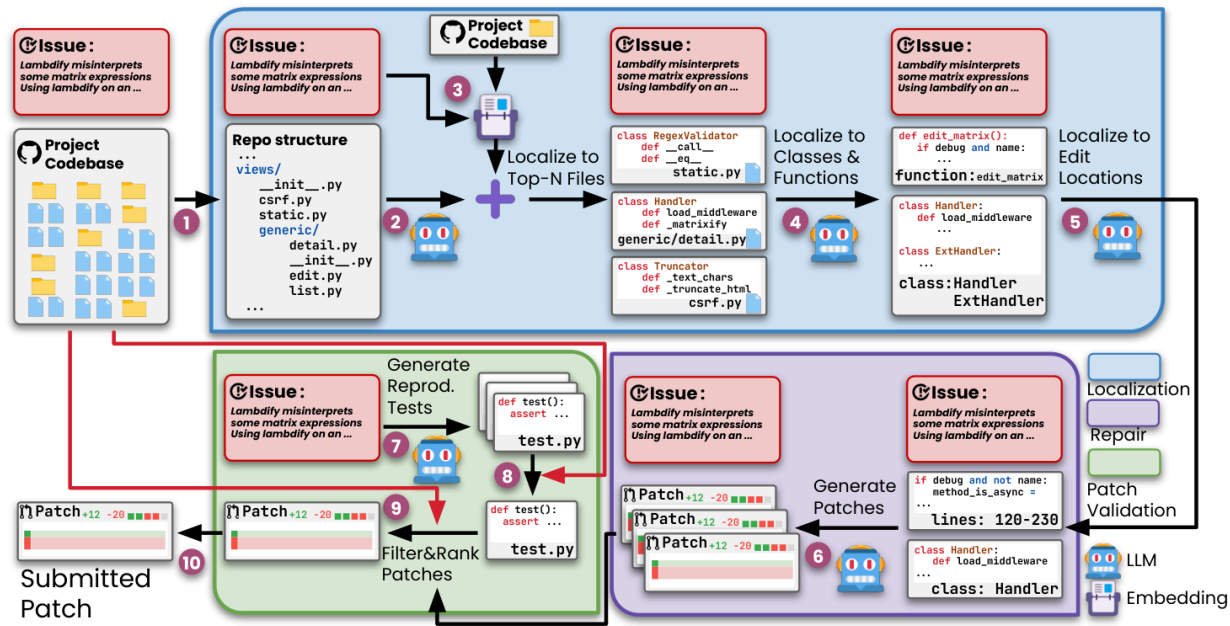
验证：检查修复与回归

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

An alternative: a fixed workflow



Xia et al. · Agentless, original overview figure ↗

37

另一种方案：固定 workflow

流程图示例：Issue——Lambdify 将某些矩阵表达式误解为使用 lambdify 的形式 .....；Project Codebase。

代码仓库结构示例：views/、\_\_init\_\_.py、csrf.py、static.py、generic/detail.py、edit.py、list.py 等。

1 输入问题与代码仓库；2 将问题与代码库交给流程；3 定位排名靠前的 N 个文件；示例文件包括 static.py、generic/detail.py、csrf.py。

定位到类与函数：RegexValidator、Handler、Truncator；示例方法包括 \_\_call\_\_、\_\_eq\_\_、load\_middleware、truncate\_chars。

定位到编辑位置：edit\_matrix()；示例定位到第 120–230 行。

6 生成补丁；以补丁、代码片段和嵌入表示定位结果。

7 生成复现测试：test.py 中编写 def test(): assert ...；8 执行测试并筛选候选。

9 筛选并排序补丁；10 提交补丁。

图例：定位；修复；补丁验证；LLM；嵌入。

Xia 等 · Agentless · 原始概览图 ↗

译注 / 与原文翻译分开

原图中的问题描述和部分代码片段被省略号截断；译文保留了可辨认内容，没有补写省略部分。

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分



编码工具集

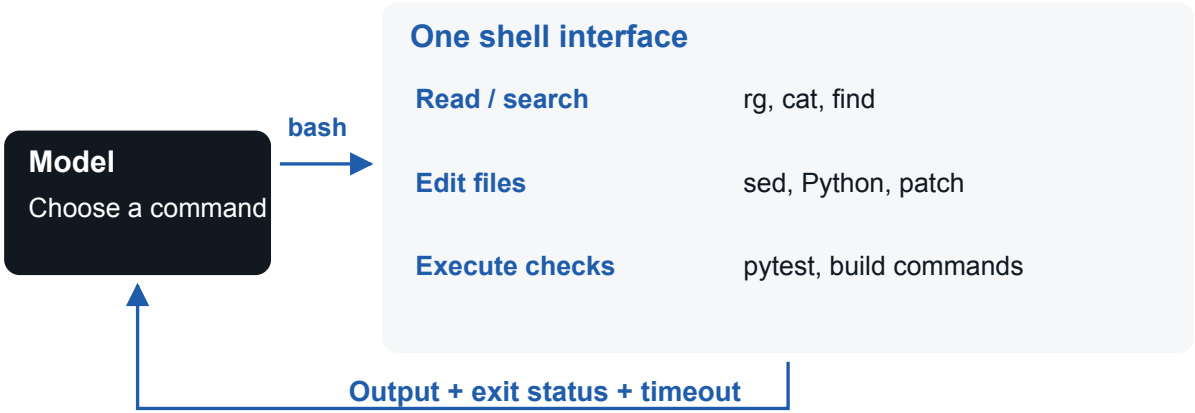
编码工具集

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Coding with a shell-only toolset



[mini-SWE-agent · shell-only interface and execution model ↗](#)

39

使用仅含 shell 的工具集进行编码

模型 → 选择命令 → bash → 一个 shell 接口。

读取 / 搜索：rg、cat、find。

编辑文件：sed、Python、patch。

执行检查：pytest、构建命令。

输出 + 退出状态 + 超时。

mini-SWE-agent · 仅 shell 接口与执行模型 ↗

使用仅含 shell 的工具集进行编码

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

# Editing with shell commands

## Replace text with sed

```
sed 's/^RETRIES = 3$/RETRIES = 5/' \
  settings.py > settings.new
mv settings.new settings.py
```

Match the line, write the result, replace the file.

## One shell, many editing methods

- sed / awk**  
Text patterns and line-based changes
- Python scripts**  
Compute replacements across files

使用 shell 命令编辑

使用 shell 命令编辑

用 sed 替换文本：sed 's/^RETRIES = 3\$/RETRIES = 5/' settings.py > settings.new；mv settings.new settings.py

匹配该行，写出结果，然后替换文件。

一个 shell，多种编辑方法：sed / awk——文本模式与按行修改。

Python 脚本——跨文件计算替换内容。

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Whole files and text replacements

Whole file · settings.py

```
RETRIES = 5
TIMEOUT = 30
```

Simple to apply; repeats unchanged code.  
Pi write · OpenCode write · OpenHands create

Search / replace · send old and new

```
settings.py
<<<<<<< SEARCH
RETRIES = 3
=====
RETRIES = 5
>>>>>>> REPLACE
```

Compact; needs an unambiguous match.  
OpenHands · Pi · OpenCode edit

完整文件 · settings.py: RETRIES = 5; TIMEOUT = 30。

容易应用；但会重复未变化的代码。

Pi write · OpenCode write · OpenHands create

搜索 / 替换 · 发送旧内容和新内容: settings.py; <<<<<<< SEARCH /  
RETRIES = 3 / ===== / RETRIES = 5 / >>>>>>> REPLACE

紧凑；但需要无歧义的匹配内容。

OpenHands · Pi · OpenCode edit

Aider · [格式](#) [↗](#) · [OpenHands](#) [↗](#) · [Pi](#) [↗](#) · [OpenCode](#) [↗](#)

完整文件与文本替换

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Diffs and patch commands

Unified diff

```
--- a/settings.py
+++ b/settings.py
@@ -1,2 +1,2 @@
-RETRIES = 3
+RETRIES = 5
  TIMEOUT = 30
```

Standard: line positions + context.  
Aider: omit hunk line numbers.

File-operation patch · Codex syntax

```
*** Begin Patch
*** Update File: settings.py
@@
-RETRIES = 3
+RETRIES = 5
  TIMEOUT = 30
*** End Patch
```

File operations + context; no line counts.  
[Codex · OpenCode / OpenHands \(preset-dependent\)](#)

差异与补丁命令

统一差异 (unified diff) : --- a/settings.py; +++ b/settings.py; @@ -1,2 +1,2 @@; -RETRIES = 3; +RETRIES = 5; TIMEOUT = 30。

标准格式: 行号 + 上下文。

Aider: 省略 hunk 行号。

文件操作补丁 · Codex 语法: \*\*\* Begin Patch; \*\*\* Update File: settings.py; @@; -RETRIES = 3; +RETRIES = 5; TIMEOUT = 30; \*\*\* End Patch。

文件操作 + 上下文; 不使用行数。

Codex · OpenCode / OpenHands (取决于预设)

Aider · [统一差异 ↗](#) · Codex · [解析器 ↗](#) · OpenCode · [选区 ↗](#) · OpenHands · [GPT-5 预设 ↗](#)

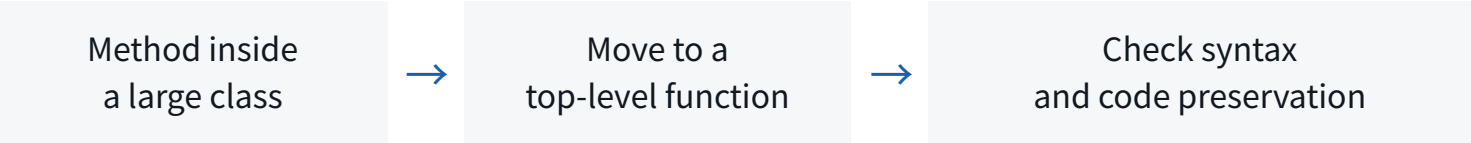
差异与补丁命令

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

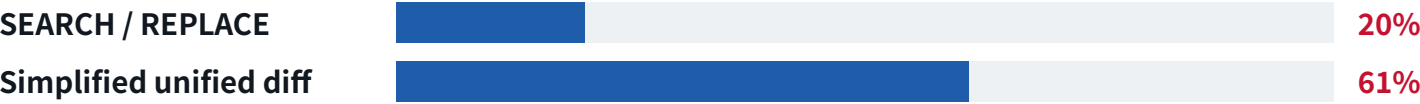
ZH / 中文译文 · 第 1/1 部分

Diff format can make a big difference



Aider refactoring benchmark: **89 Python tasks**

GPT-4 Turbo (gpt-4-1106-preview) · benchmark success rate



Aider · unified-diff experiment and refactoring benchmark ↗

差异格式可能显著影响结果

大类内部的方法 → 移到顶层函数 → 检查语法并保留代码。

Aider 重构基准：89 个 Python 任务。

GPT-4 Turbo (gpt-4-1106-preview) · 基准成功率

SEARCH / REPLACE：20%

简化统一差异：61%

Aider · 统一差异实验与重构基准 ↗

差异格式可能显著影响结果

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

# Finding relevant code

## Start with the symptom

```
rg 'retries' .
```

A flag, a loader, and a retry loop may all match.

## Follow the value

**Configuration input**  
Where is zero first read?

**Defaulting logic**  
Where does zero become three?

## 查找相关代码

## 查找相关代码

从症状开始：rg 'retries' .

标志、加载器和重试循环可能都会匹配。

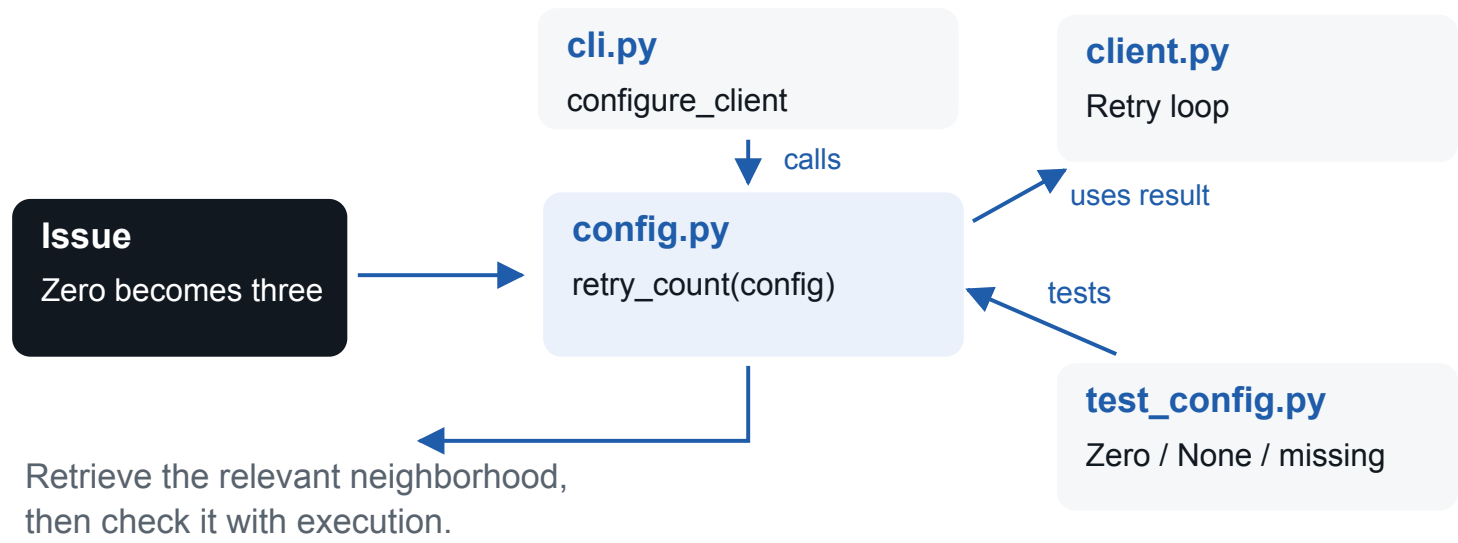
追踪值的流向：配置输入——零值最早在哪里被读取？默认值逻辑——零值在哪里变成 3？

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Navigating code with dependency graphs



Chen et al., 2025 · LocAgent ↗

45

使用依赖图导航代码

问题：零值变成了 3。

cli.py: configure\_client 调用 config.py: retry\_count(config); client.py: 重试循环使用返回结果; test\_config.py: 测试 Zero / None / missing。

先检索相关邻域，再用执行进行验证。

Chen 等, 2025 · LocAgent ↗

使用依赖图导航代码

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

编码智能体的评估与训练

# Coding agent evaluation and training

46

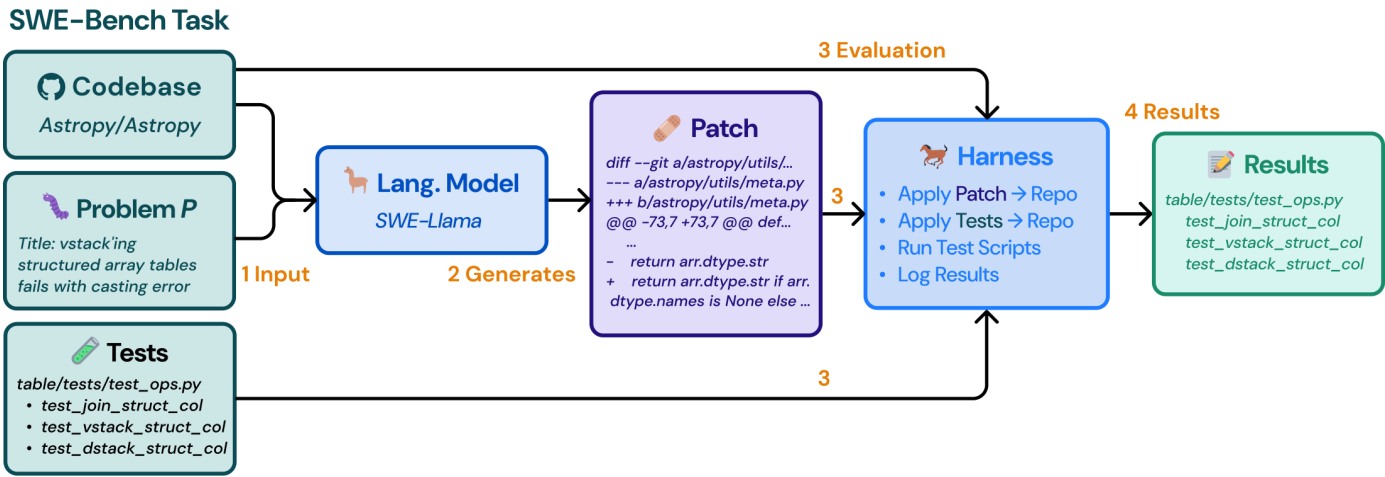
编码智能体的评估与训练

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Evaluating issue resolution



Jimenez et al. · SWE-bench evaluation pipeline, original source figure ↗

47

评估问题修复

SWE-Bench 任务：代码库 Astropy/Astropy；问题 P：标题 “vstacking structured array tables fails with casting error” ；测试：table/tests/test\_ops.py、test\_join\_struct\_col、test\_vstack\_struct\_col、test\_dstack\_struct\_col。

1 输入：代码库 + 问题 + 测试 → 2 生成：语言模型 SWE-Llama → 补丁 → 3 评估：执行框架（harness）将补丁应用到代码仓库、应用测试、运行测试脚本、记录日志 → 4 结果：table/tests/test\_ops.py、test\_join\_struct\_col、test\_vstack\_struct\_col、test\_dstack\_struct\_col。

补丁示例显示对 astropy/utils/metadata.py 的 diff，包括将 return arr.dtype.str 改为 return arr.dtype.str if arr.dtype.str is not None else ....

失败 → 通过：按要求完成修复。

通过 → 通过：保留既有行为。

Jimenez 等 · SWE-bench 评估流程 · 原始来源图 ↗

评估问题修复

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Building runnable training tasks

|                |                                            |
|----------------|--------------------------------------------|
| Runnable task  |                                            |
| Starting state | Files + dependencies + test command        |
| Issue          | Requested behavior + reproduction          |
| Checks         | Bug fails before; repair restores behavior |

Validate the environment before collecting training trajectories.

[SWE-Gym · executable training environments ↗](#)

48

构建可运行的训练任务

可运行任务：起始状态——文件 + 依赖 + 测试命令。

问题：所需行为 + 复现步骤。

检查：修复前缺陷测试失败；修复后恢复预期行为。

收集训练轨迹前，先验证环境。

SWE-Gym · 可执行训练环境 ↗

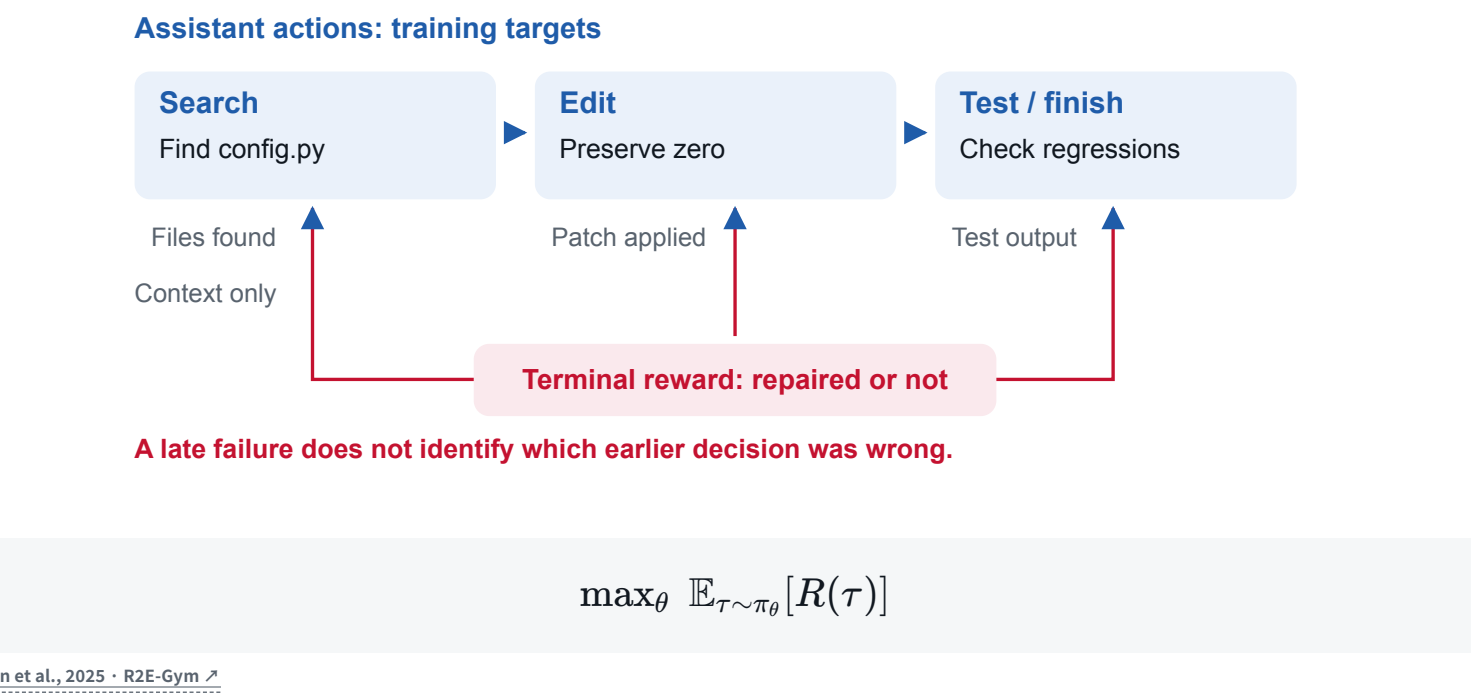
构建可运行的训练任务

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

RL for multi-step code repairs



针对多步代码修复进行强化学习

assistant 动作：训练目标

搜索：找到 config.py → 找到文件；编辑：保留零值 → 补丁已应用；测试 / 完成：检查回归 → 测试输出。

仅作上下文

终端奖励：是否修复。

较晚出现的失败无法指出较早的哪个决策错了。

$\max_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} [R(\tau)]$

Jain 等, 2025 · R2E-Gym ↗

针对多步代码修复进行强化学习

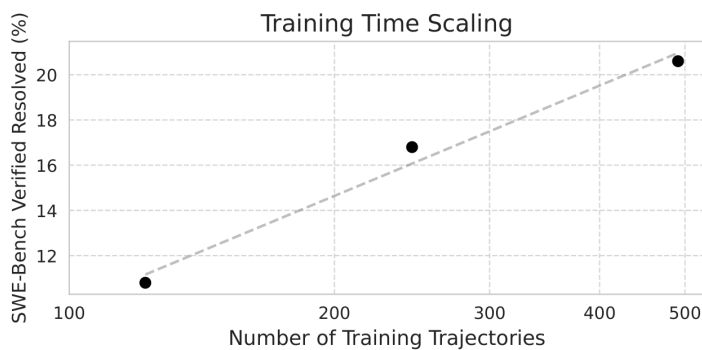
第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

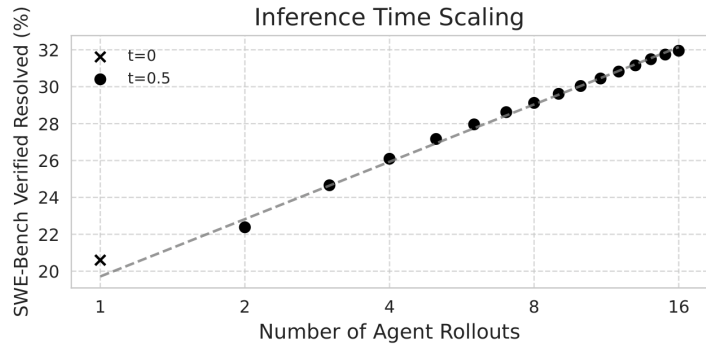
ZH / 中文译文 · 第 1/1 部分

Training data and inference compute

Training scale



Inference scale: learned-verifier selection



Pan et al., 2024 · SWE-Gym · 32B model ↗

50

训练数据与推理计算量

训练规模：训练时间扩展；横轴为训练轨迹数量，纵轴为 SWE-Bench Verified 解决率（%）。

推理规模：学习式验证器选择；推理时间扩展；横轴为智能体轨迹采样数量，纵轴为 SWE-Bench Verified 解决率（%）；图例 t=0、t=0.5。

Pan 等, 2024 · SWE-Gym · 32B 模型 ↗

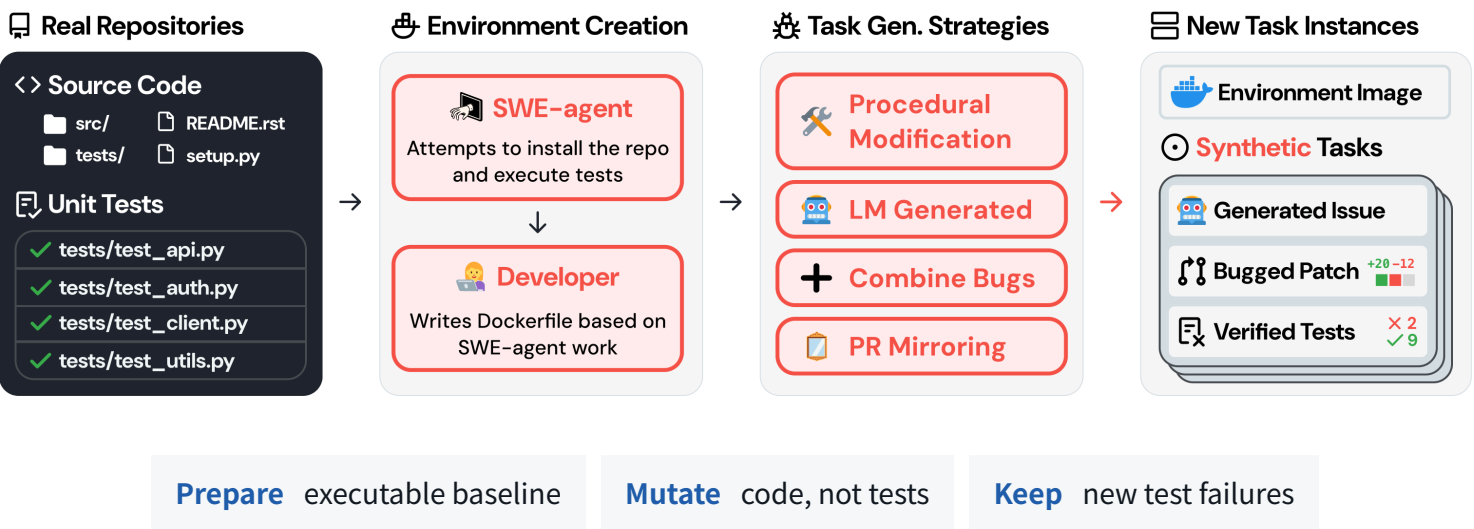
训练数据与推理计算量

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Creating repair tasks by injecting bugs



Yang et al., 2025 · SWE-smith, original pipeline figure ↗

51

通过注入缺陷创建修复任务

真实代码仓库：源代码 src/、测试 tests/、README.rst、setup.py；单元测试 tests/test\_api.py、tests/test\_auth.py、tests/test\_client.py、tests/test\_utils.py。

环境构建：SWE-agent 尝试安装代码仓库并执行测试；开发者根据 SWE-agent 工作编写 Dockerfile。

任务生成策略：过程式修改、语言模型生成、组合缺陷、PR 镜像。

新任务实例：环境镜像、合成任务、生成的问题、有缺陷的补丁、已验证测试。图示示例新增 +20、-12 项变更；验证测试计数显示 ×2、✓9。

准备：可执行基线。修改：修改代码，不修改测试。保留：新的测试失败。

Yang 等, 2025 · SWE-smith · 原始流程图 ↗

通过注入缺陷创建修复任务

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

# Injecting and checking a bug

## Working baseline

```
value = config.get("retries")
return 3 if value is None else value
```

All four existing tests pass.

## Mutate the condition

```
value = config.get("retries")
return 3 if not value else value
```

Zero now becomes three; its test fails.

|               |                         |                 |
|---------------|-------------------------|-----------------|
| Before 4 pass | Mutated 3 pass · 1 fail | Restored 4 pass |
|---------------|-------------------------|-----------------|

[SWE-smith · procedural mutation and execution validation ↗](#)

## 注入并检查缺陷

正常基线：value = config.get("retries"); return 3 if value is None else value。

修改条件：value = config.get("retries"); return 3 if not value else value。

现有四项测试全部通过。

零值现在变成 3；相应测试失败。

修改前：4 项通过 → 注入缺陷后：3 项通过 · 1 项失败 → 恢复后：4 项通过。

SWE-smith · 程序化缺陷注入与执行验证 ↗

## 注入并检查缺陷

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Working across programming languages

Python

pyproject.toml

src/

tests/

pytest

TypeScript

package.json

src/

tests/

npm test

Rust

Cargo.toml

src/

tests/

cargo test

Read the project configuration before choosing build and test commands.

跨编程语言开展工作

Python: pyproject.toml、src/、tests/; pytest。

TypeScript: package.json、src/、tests/; npm test。

Rust: Cargo.toml、src/、tests/; cargo test。

选择构建与测试命令之前，先阅读项目配置。

[Multi-SWE-bench ↗](#) · [SWE-bench Multilingual ↗](#)

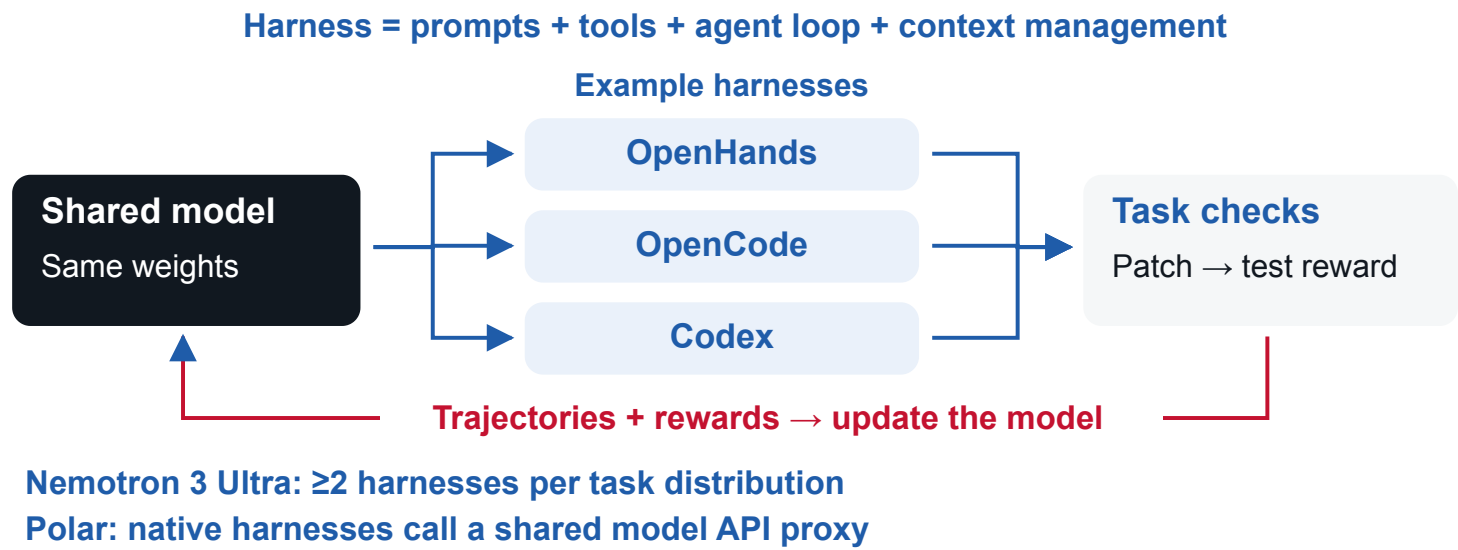
跨编程语言开展工作

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Multi-harness training



[Nemotron 3 Super ↗](#) [Nemotron 3 Ultra ↗](#) [Polar ↗](#) [Harness implementations ↗](#)

54

多执行框架训练

执行框架 (harness) = 提示词 + 工具 + 智能体循环 + 上下文管理。

共享模型：相同权重 → OpenHands / OpenCode / Codex 等示例 执行框架 (harness) → 任务检查：补丁 → 测试奖励。

轨迹 + 奖励 → 更新模型。

Nemotron 3 Ultra：每个任务分布至少使用 2 个 执行框架 (harness) 。

Polar：原生 执行框架 (harness) 调用共享模型 API 代理。

Nemotron 3 Super ↗ · Nemotron 3 Ultra ↗ · Polar ↗ · 执行框架实现 ↗

多执行框架训练

## 第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

前端开发

# Frontend development

55

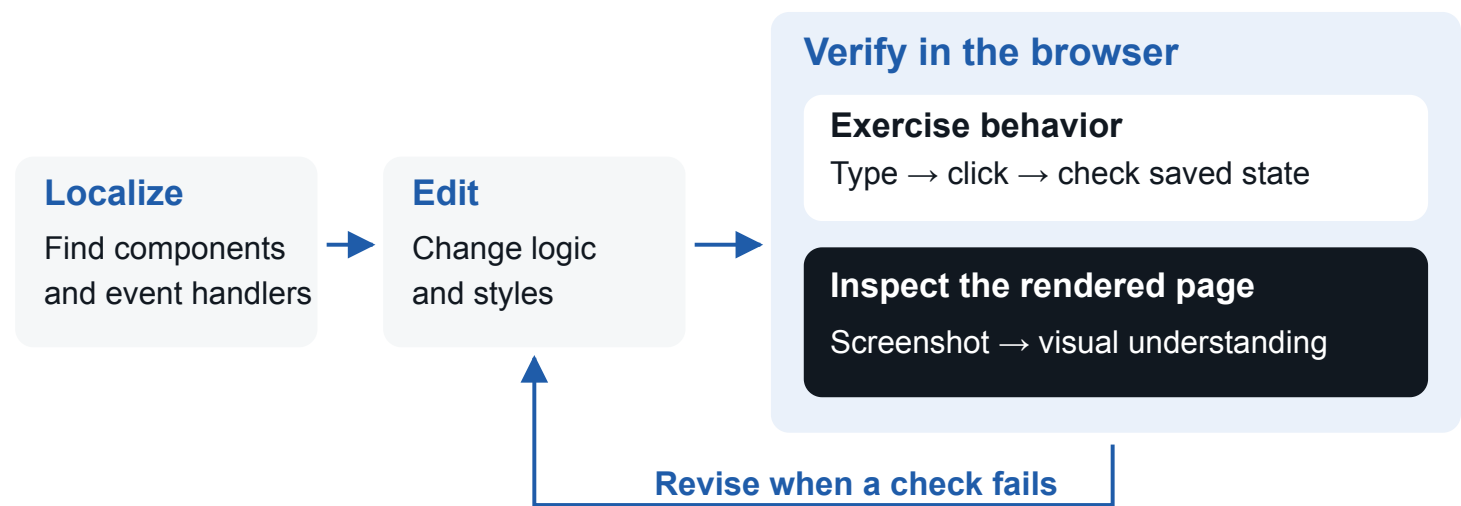
前端开发

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Agent-powered frontend development



Yang et al., 2024 · SWE-bench Multimodal · browser and screenshot tools for SWE-agent ↗

56

由智能体驱动的前端开发

定位：查找组件和事件处理器 → 编辑：修改逻辑与样式 → 在浏览器中验证：运行交互——输入 → 点击 → 检查已保存状态；检查渲染页面——截图 → 视觉理解。  
检查失败时进行修改。

Yang 等, 2024 · SWE-bench Multimodal · 为 SWE-agent 提供浏览器与截图工具 ↗

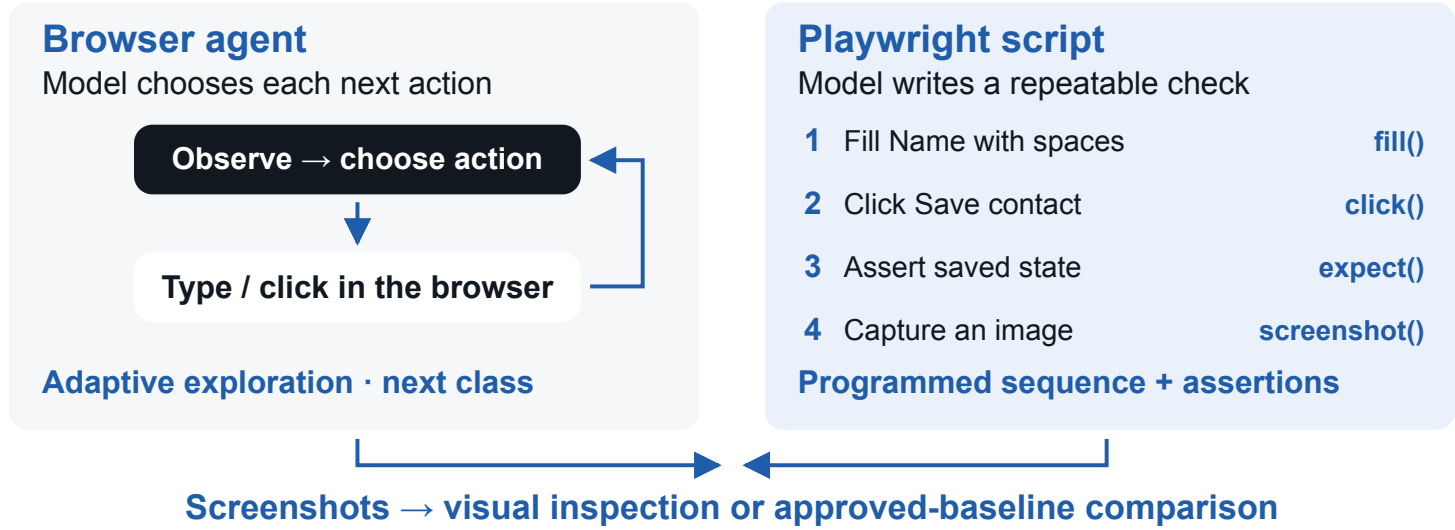
由智能体驱动的前端开发

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Browser agents and automation scripts



Koh et al., 2024 · VisualWebArena · Playwright actions · Assertions · Screenshot comparisons

浏览器智能体与自动化脚本

浏览器智能体：模型选择每个后续动作；观察 → 选择动作；在浏览器中输入 / 点击；自适应探索 · 下一讲。

Playwright 脚本：模型编写可重复的检查。1 填写 Name（名称）并输入空格——fill()；2 点击 Save contact（保存联系人）——click()；3 断言保存状态——expect()；4 截取图像——screenshot()。

程序化步骤 + 断言。

截图 → 视觉检查或与获准的基准图比较。

Koh 等, 2024 · VisualWebArena · Playwright 操作 · 断言 · 截图比较

浏览器智能体与自动化脚本

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

# Verifying a frontend repair

Before: blank record saved

Original teaching fixture - original bug

Create a contact

A name must contain at least one non-space character.

Name

Save contact

Saved records: 1

After: blank name reiected

Original teaching fixture - repaired

Create a contact

A name must contain at least one non-space character.

Name

Enter a non-blank name.

Save contact

Saved records: 0

Edit value.length → value.trim().length

Behavior

0 records; valid names still save

Appearance

Error visible and readable beside Name

## 验证前端修复

### 验证前端修复

修改前：保存了空记录。修改后：拒绝空姓名。

编辑 value.length → value.trim().length。

行为：记录数为 0；有效姓名仍可保存。

外观：错误提示在 Name（名称）旁边可见且易读。

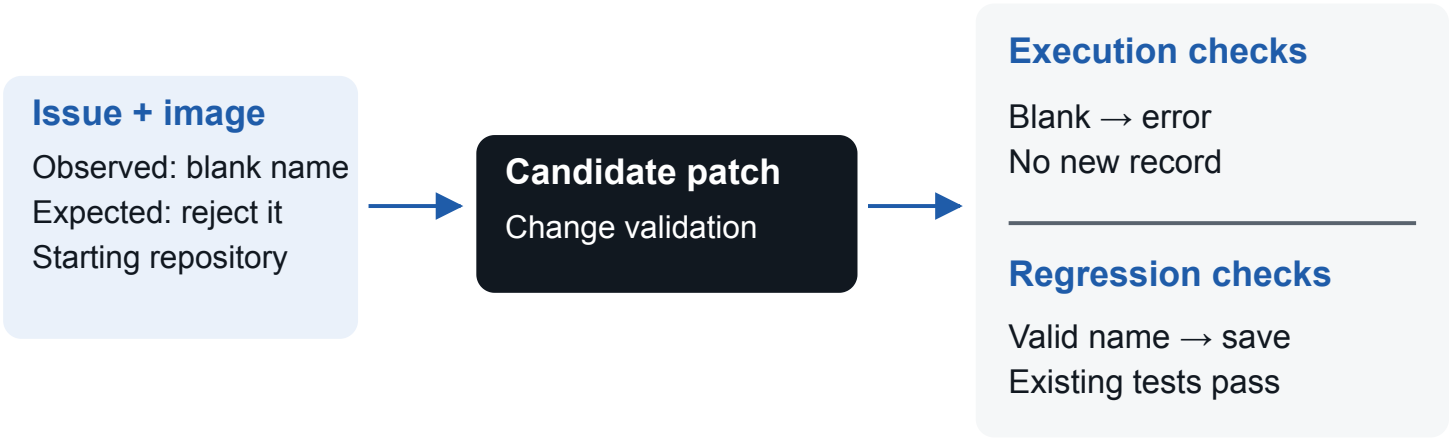
可运行课堂示例 ↗ · Playwright 断言 ↗ · 视觉检查 ↗

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Evaluating visual issue resolution



An image in the issue does not make image similarity the grading rule.

[SWE-bench Multimodal · issue images and test-based patch evaluation ↗](#)

59

评估视觉问题修复

问题 + 图像：观察到空姓名；预期拒绝；起始代码仓库。候选补丁：修改验证逻辑。

执行检查：空值 → 错误；不会创建新记录。

回归检查：有效姓名 → 保存；现有测试通过。

问题描述中有图像，并不意味着评分规则就是图像相似度。

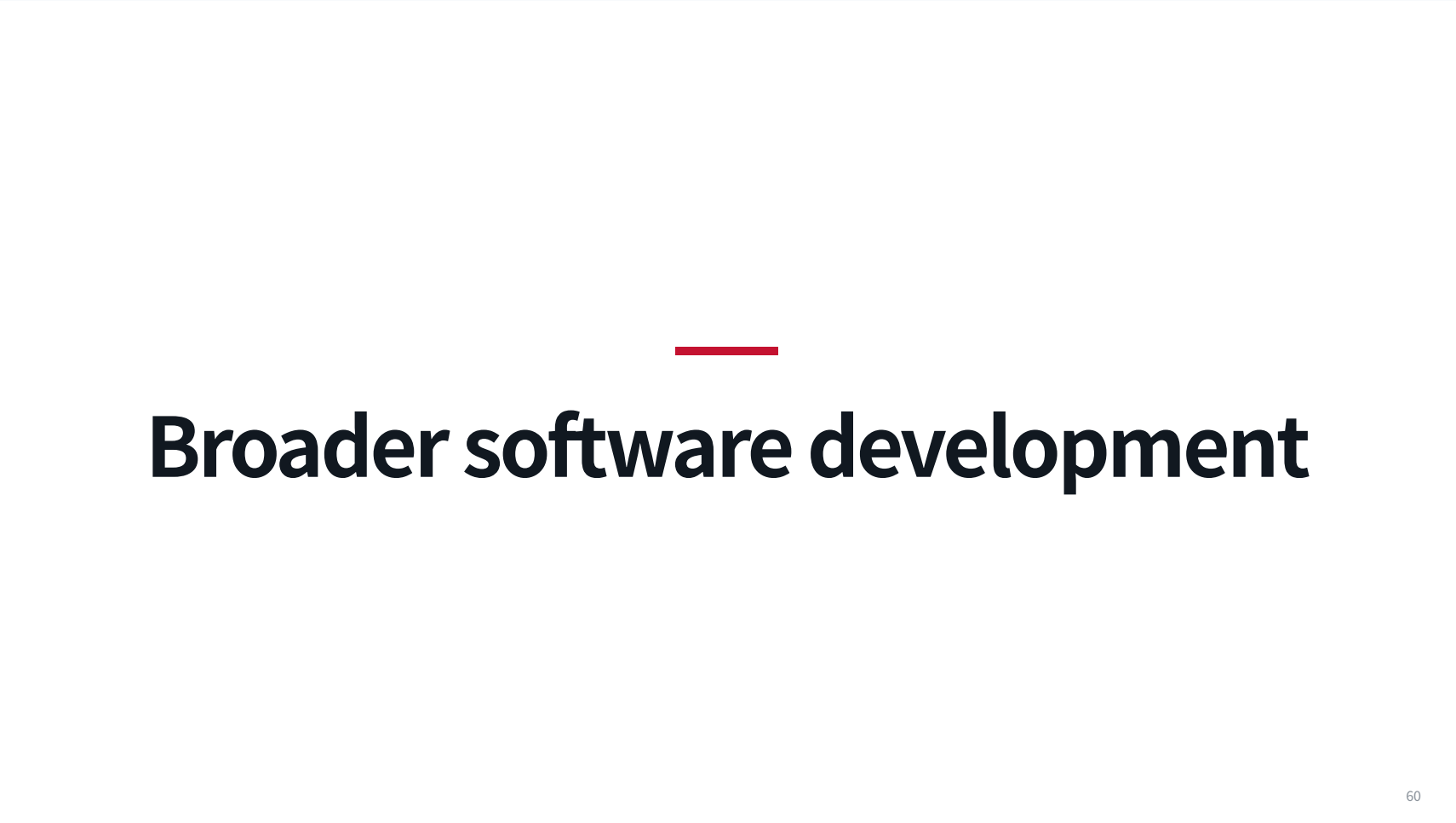
SWE-bench Multimodal · 问题图像与基于测试的补丁评估 ↗

评估视觉问题修复

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分



更广泛的软件开发

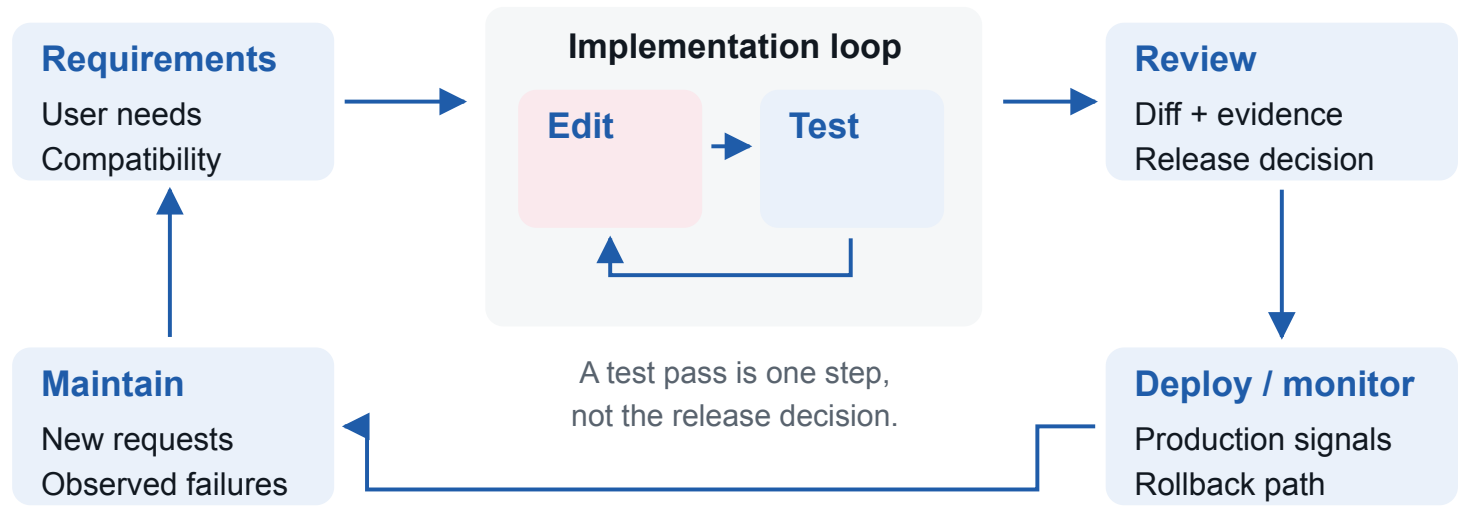
更广泛的软件开发

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

From requirements to maintenance



61

需求：用户需要 · 兼容性 → 实现循环：编辑 · 测试 → 评审：差异 + 证据 · 发布决策。

维护：新需求 · 观察到的故障。

测试通过只是一个步骤，不是发布决策。

部署 / 监控：生产环境信号 · 回滚路径。

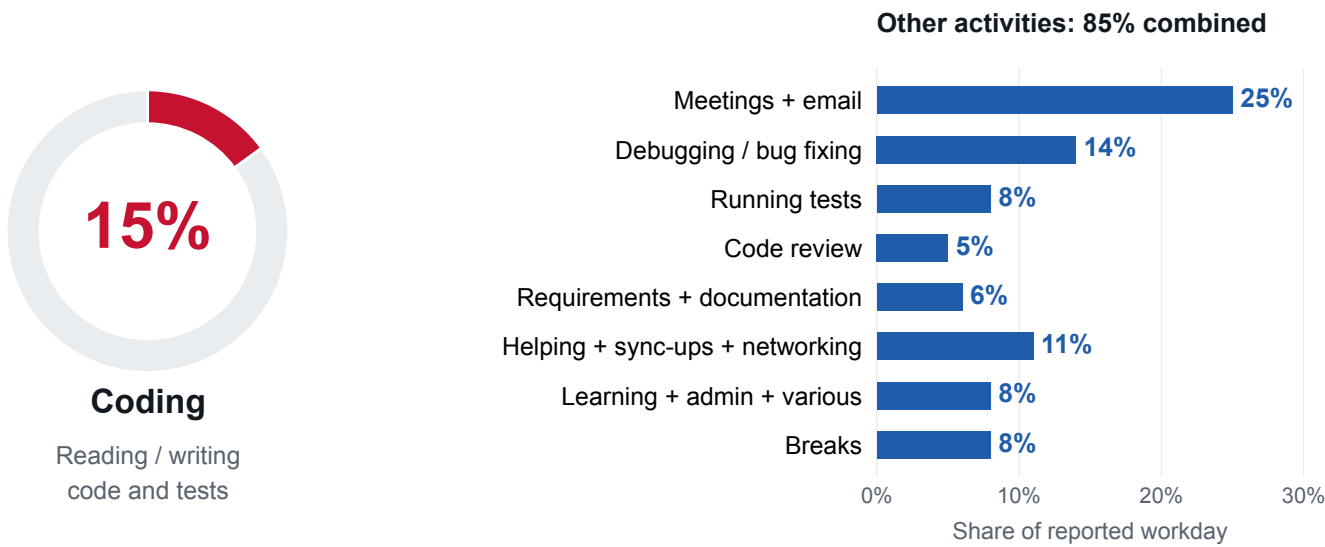
从需求到维护

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Where developers spend their time



Meyer et al., 2019 · Table 2 · 5,928 self-reported Microsoft workdays ↗

62

开发者把时间花在哪里

编码：15%。阅读 / 编写代码和测试。

其他活动：合计 85%。

报告的工作日时间占比：会议 + 电子邮件 25%；调试 / 修复缺陷 14%；帮助他人 + 同步沟通 + 建立协作关系 11%；运行测试 8%；学习 + 行政 + 其他 8%；休息 8%；需求 + 文档 6%；代码评审 5%。

横轴：报告工作日时间占比，刻度 0%、10%、20%、30%。

Meyer 等，2019 · 表 2 · 5,928 个微软员工自报工作日 ↗

开发者把时间花在哪里

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Comparing software development tasks

| Task             | Artifact              | Environment / verifier                   | Horizon         |
|------------------|-----------------------|------------------------------------------|-----------------|
| Repair           | Patch                 | Repository / regression tests            | One issue       |
| Library creation | Implementation        | Scaffold / package tests                 | Many functions  |
| Test generation  | New test              | Buggy + fixed versions / discrimination  | One behavior    |
| CI repair        | Code or configuration | Workflow / required checks               | Build run       |
| Evolution        | Accumulated changes   | Persistent app / milestone + regressions | Dependent tasks |

63

比较软件开发任务

任务 | 工件 | 环境 / 验证器 | 时间范围

修复 | 补丁 | 代码仓库 / 回归测试 | 一个问题

库创建 | 实现 | 脚手架 / 包测试 | 多个函数

测试生成 | 新测试 | 缺陷版本 + 修复版本 / 区分测试 | 一种行为

CI 修复 | 代码或配置 | 工作流 / 必需检查 | 一次构建运行

演进 | 累积变更 | 持久化应用 / 里程碑 + 回归测试 | 相互依赖的任务

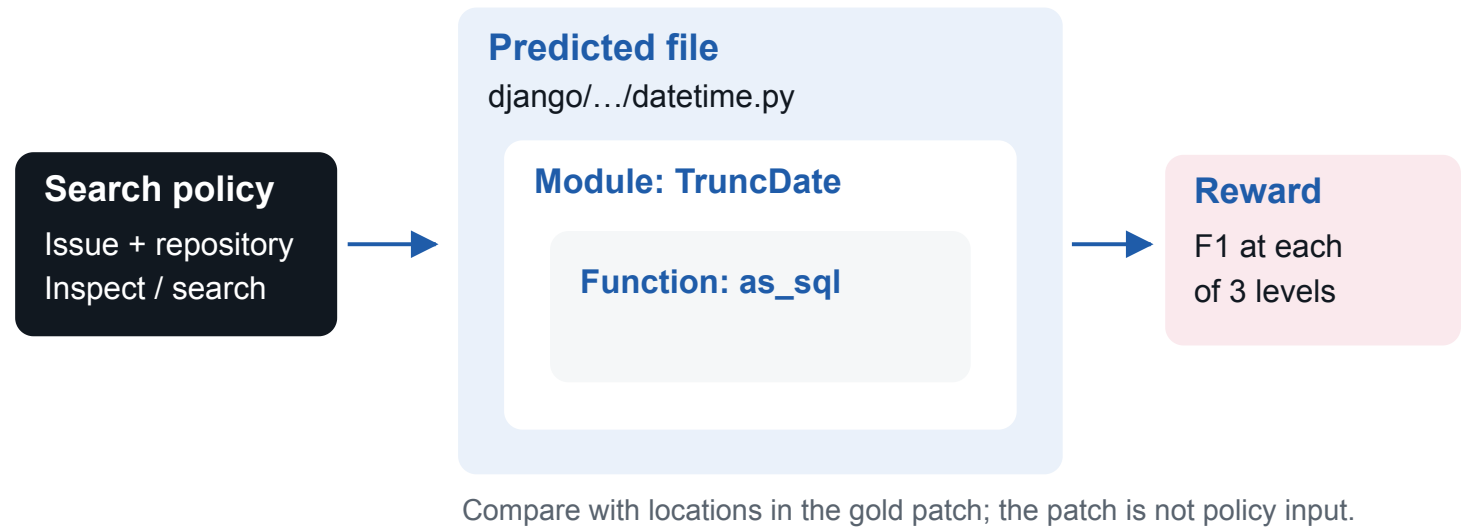
比较软件开发任务

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Task: localization



[CodeScout, 2026 ↗](#)

64

任务：代码定位

搜索策略：问题 + 代码仓库 → 检查 / 搜索。

预测文件：django/.../datetime.py；模块：TruncDate；函数：as\_sql。

奖励：与标准补丁中的位置比较；在 3 个层级分别计算 F1。

标准补丁不能作为策略的输入。

[CodeScout, 2026 ↗](#)

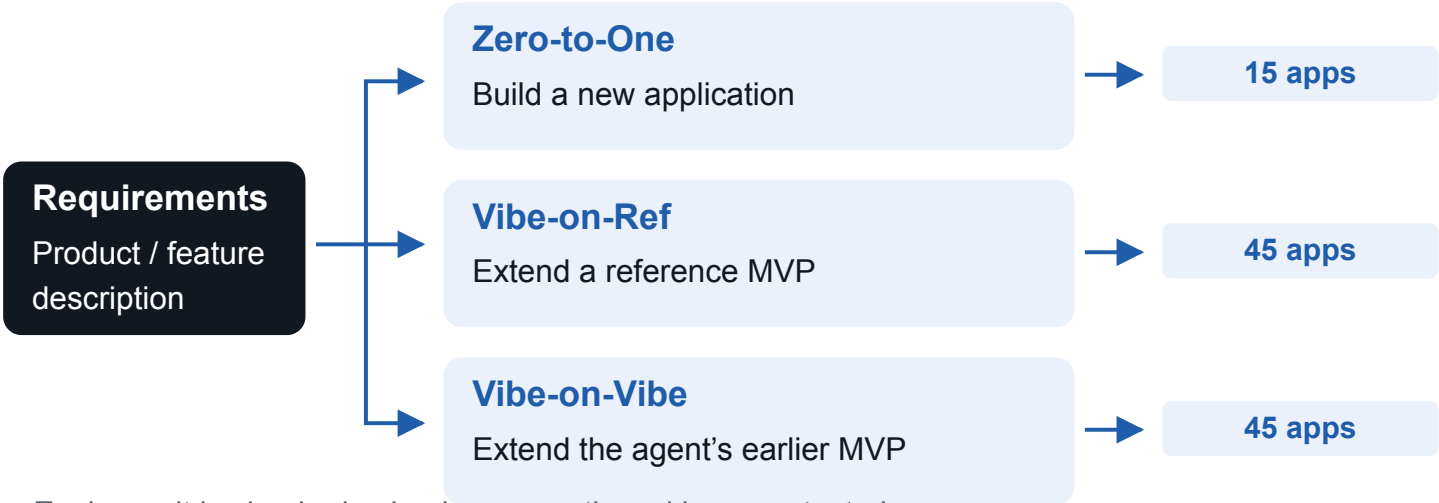
任务：代码定位

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Task: app creation and modification



Each result is checked using human-authored browser test plans.

ViBench · collection and evaluator definitions checked 2026-09-07 ↗

65

任务：应用创建与修改

需求：产品 / 功能描述。

从零到一：构建新应用；15 个应用。

Vibe-on-Ref：扩展参考 MVP；45 个应用。

Vibe-on-Vibe：扩展智能体早先创建的 MVP；45 个应用。

每个结果都使用人工编写的浏览器测试计划进行检查。

ViBench · 数据集与评估器定义已于 2026-09-07 核对 ↗

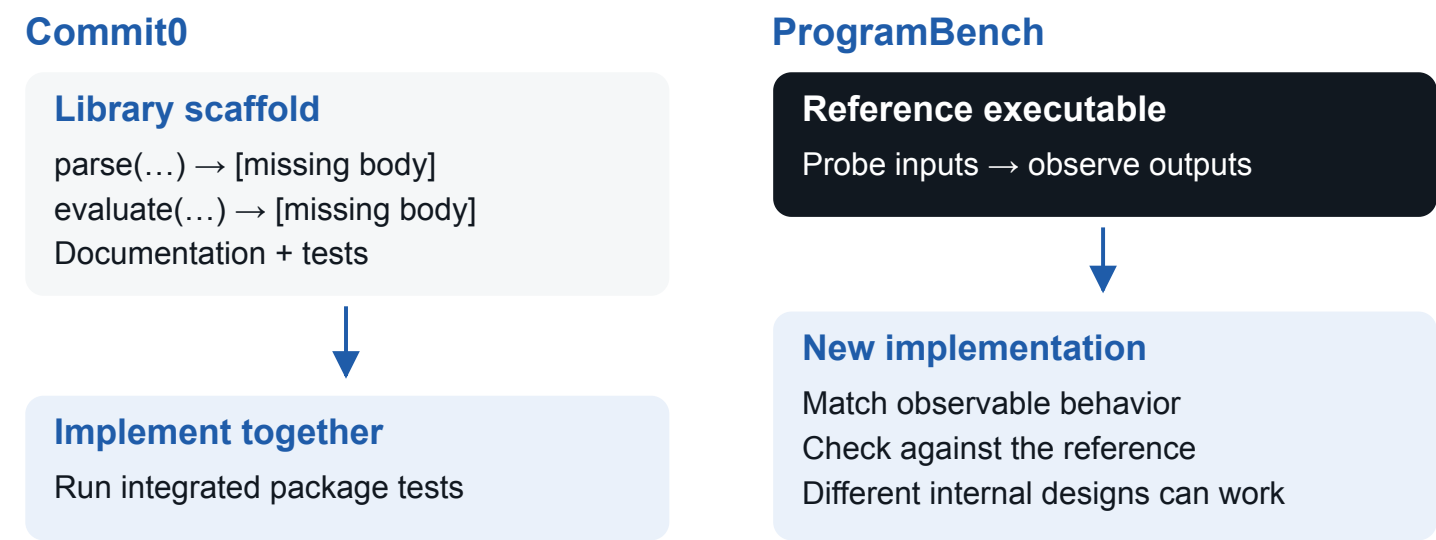
任务：应用创建与修改

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Task: library implementation



[Commit0 ↗](#) [ProgramBench ↗](#)

66

任务：库实现

Commit0：库脚手架——parse(...) → [缺失函数体]；evaluate(...) → [缺失函数体]；文档 + 测试。

一起实现：完成功能并运行集成包测试。

ProgramBench：参考可执行程序——输入探测 → 观察输出。

新实现：匹配可观察行为；根据参考实现进行检查；不同的内部设计也可能有效。

Commit0 ↗ · ProgramBench ↗

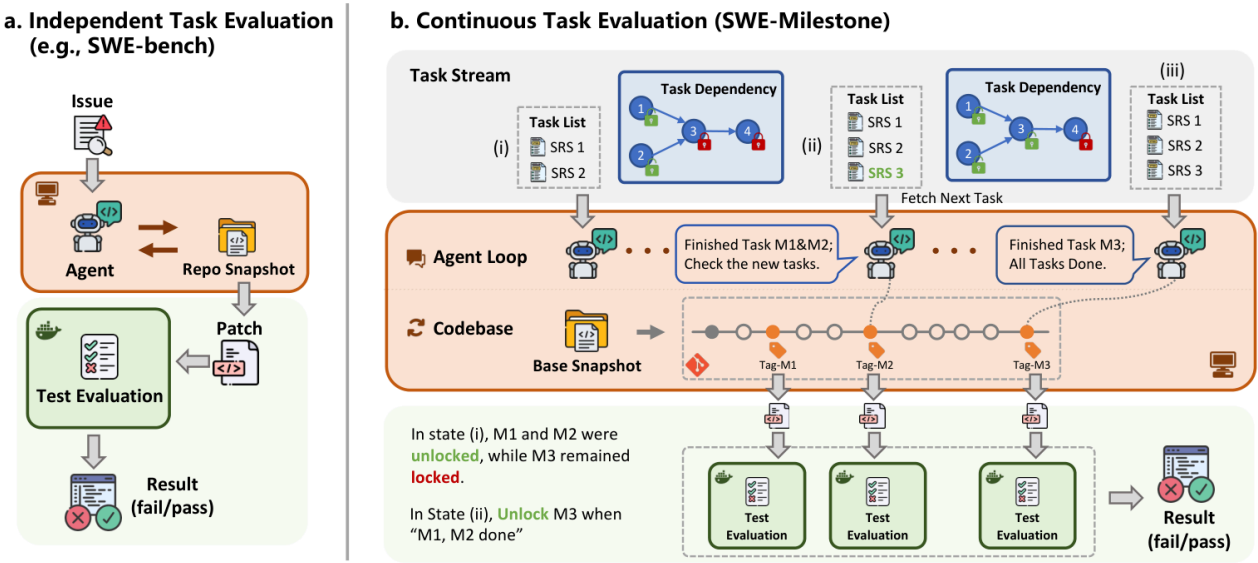
任务：库实现

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Task: software maintenance



SWE-Milestone, 2026 ↗

67

任务：软件维护

a. 独立任务评估（例如 SWE-bench）：问题 → 智能体 ↔ 代码仓库快照 → 补丁 → 测试评估 → 结果（失败 / 通过）。

b. 连续任务评估（SWE-Milestone）：任务流、任务依赖关系、任务列表（SRS 1、SRS 2、SRS 3）；智能体循环与代码库随任务持续推进。

阶段 (i)：M1 与 M2 已完成；检查新任务。阶段 (ii)：获取下一个任务；“M1、M2 已完成”时解锁 M3。阶段 (iii)：M3 完成；所有任务均已完成。

代码库：基础快照 → Tag-M1 → Tag-M2 → Tag-M3；分别在这些状态上进行测试评估；输出结果（失败 / 通过）。

状态 (i)：M1、M2 已解锁，M3 仍锁定。状态 (ii)：当“M1、M2 完成”后解锁 M3。

SWE-Milestone, 2026 ↗

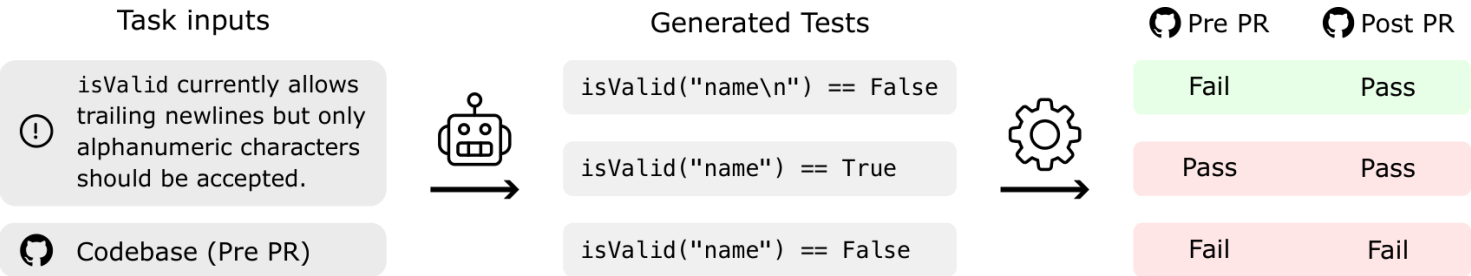
任务：软件维护

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Task: test generation



Mündler et al., 2024 · SWT-Bench ↗

68

任务：测试生成

任务输入：`isValid` 目前允许尾随换行符，但只应接受字母数字字符。代码库（PR 前）。

生成的测试：`isValid("name\n") == False`；`isValid("name") == True`；`isValid("name") == False`。

PR 前 / PR 后：失败 → 通过；通过 → 通过；失败 → 失败。

Mündler 等, 2024 · SWT-Bench ↗

任务：测试生成

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Task: CI repair

Observed build failure

```
Runtime: Node 16
Package requires: Node >=18
npm ci: EBADENGINE
```

Inspect the package’s supported runtime.

Causally relevant change

```
# setup-node configuration
with:
  node-version: '20'
```

Rerun install, build, and tests.

任务：修复 CI

观察到的构建失败：运行时 Node 16；软件包要求 Node >=18；npm ci：EBADENGINE。

因果相关的修改：# setup-node configuration；with:；node-version: '20'。

检查软件包支持的运行时。

重新运行安装、构建和测试。

LCA CI Builds Repair · 数据集修订版 ebf12dad ↗

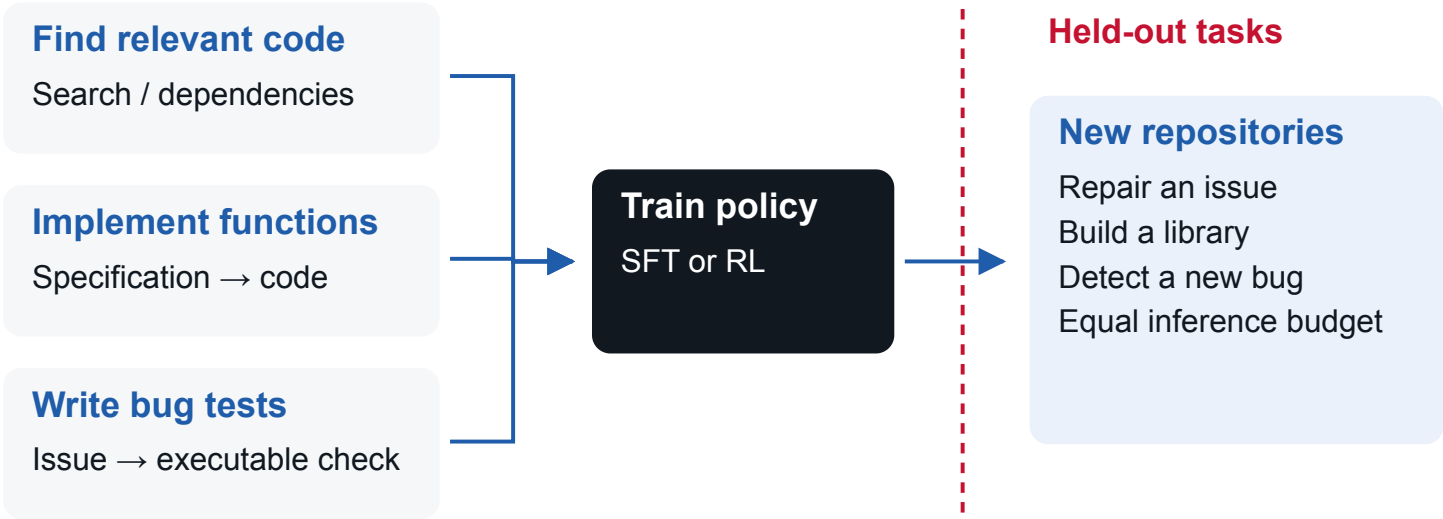
任务：修复 CI

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Learning skills for new tasks



70

为新任务学习技能

查找相关代码：搜索 / 依赖关系。

实现函数：规格说明 → 代码。

编写缺陷测试：问题 → 可执行检查。

训练策略：SFT 或 RL。

留出任务：新代码仓库；修复问题；构建库；检测新缺陷；保持相同推理预算。

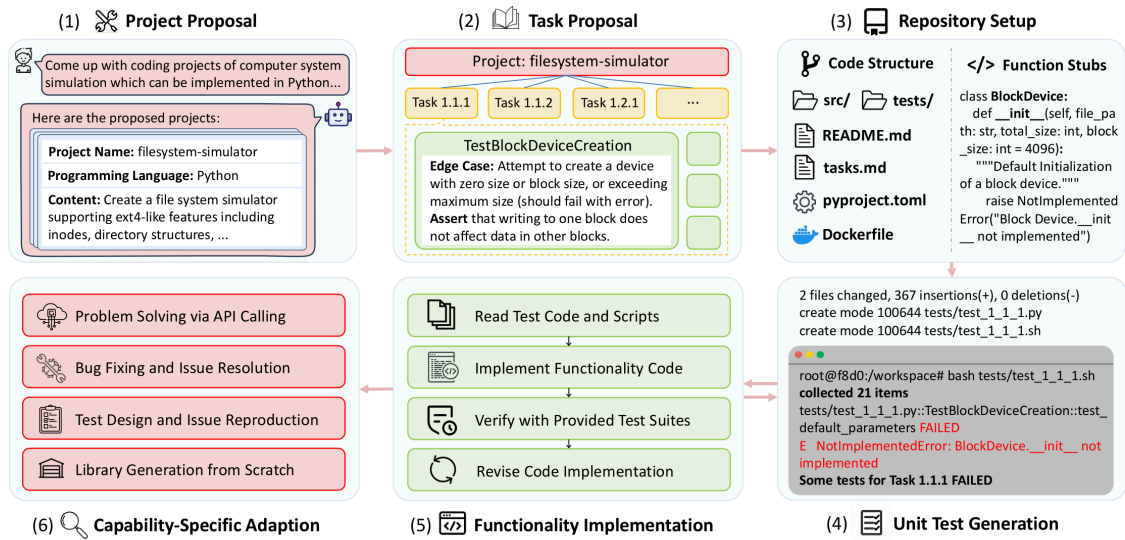
为新任务学习技能

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Generating projects for coding practice



Zhu, Gandhi & Neubig · SWE-Playground ↗

71

生成用于编码练习的项目

生成用于编码练习的项目

(1) 项目提案：提出可以用 Python 实现的计算机系统模拟编码项目。提案示例：项目名 filesystem-simulator；编程语言 Python；内容：创建支持类似 ext4 功能的文件系统模拟器，包括 inode、目录结构等。

(2) 任务提案：项目 filesystem-simulator；任务 1.1.1、1.1.2、1.2.1 等；测试 TestBlockDeviceCreation。边界情况：尝试创建零大小、块大小为零或超过最大大小的设备（应报错）。断言：写入一个块不会影响其他块中的数据。

(3) 代码仓库设置：代码结构 src/、tests/、README.md、tasks.md、pyproject.toml、Dockerfile。函数存根：class BlockDevice: def \_\_init\_\_(self, file\_path: str, total\_size: int, block\_size: int = 4096):; 默认初始化块设备；raise NotImplementedError("Block Device.\_\_init\_\_ not implemented")。

(4) 单元测试生成：变更 2 个文件，新增 367 行 (+)，删除 0 行 (-)；创建 tests/test\_1\_1\_1.py 和 tests/test\_1\_1\_1.sh。执行 bash tests/test\_1\_1\_1.sh：收集到 21 项；test\_block\_device\_creation::test\_default\_parameters 失败；NotImplementedError: BlockDevice.\_\_init\_\_ not implemented；任务 1.1.1 的部分测试失败。

(5) 功能实现：读取测试代码和脚本 → 实现功能代码 → 使用提供的测试套件验证 → 修改代码实现。

(6) 能力专项适配：通过 API 调用进行问题求解；修复缺陷与解决问题；测试设计与问题复现；从零开始生成库。

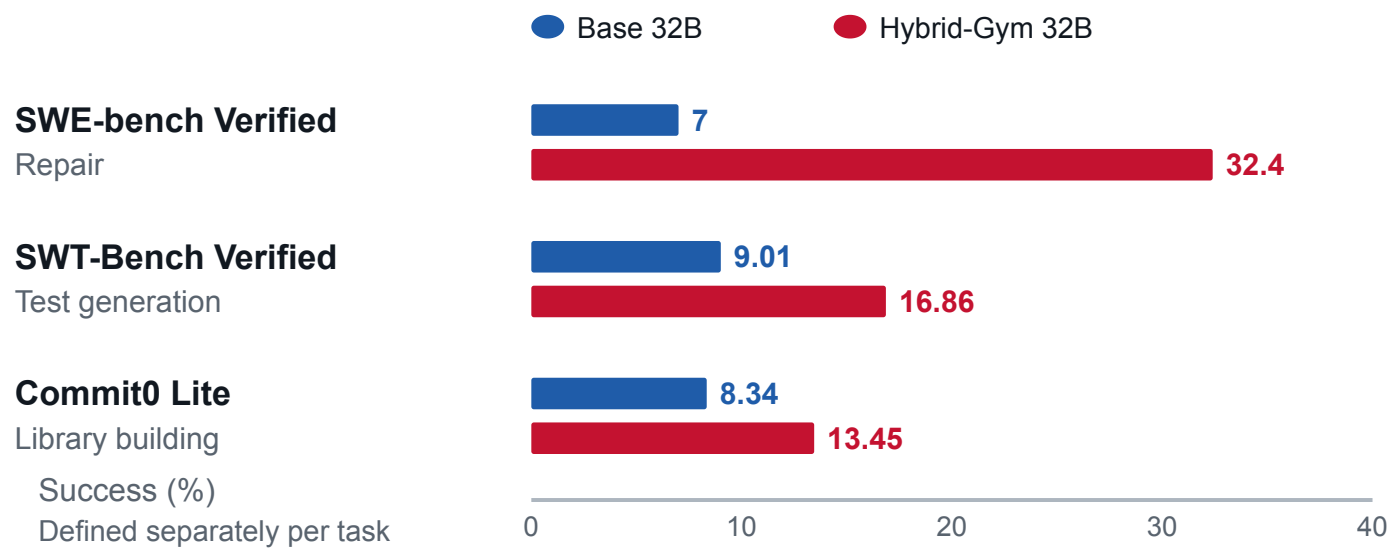
Zhu、Gandhi 与 Neubig · SWE-Playground ↗

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Transferring training to new development tasks



Hybrid-Gym, 2026 · selected 32B results ↗

72

将训练迁移到新的开发任务

基准 | 任务 | Base 32B | Hybrid-Gym 32B

SWE-bench Verified | 修复 | 7 | 32.4

SWT-Bench Verified | 测试生成 | 9.01 | 16.86

Commit0 Lite | 库构建 | 8.34 | 13.45

成功率（%），每项任务分别定义；横轴刻度 0、10、20、30、40。

Hybrid-Gym, 2026 · 32B 模型部分结果 ↗

将训练迁移到新的开发任务

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

预测代码行为的模型



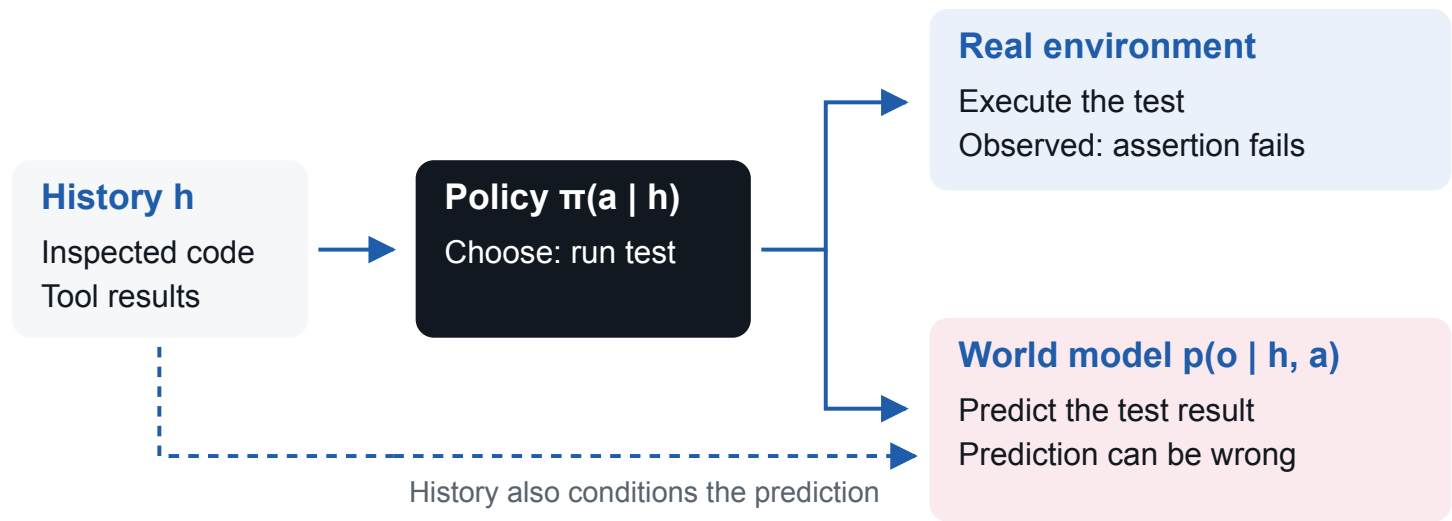
预测代码行为的模型

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Choosing actions and predicting their effects



74

选择动作并预测其影响

历史  $h$ ：已检查的代码、工具结果  $\rightarrow$  策略  $\pi(a \mid h)$ ：选择运行测试。

历史也会作为预测条件。

真实环境：执行测试；观测到断言失败。

世界模型  $p(o \mid h, a)$ ：预测测试结果；预测可能出错。

选择动作并预测其影响

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Predicting execution: shared references

Predict the next state

```
items = [1]
alias = items
alias.append(2)
result = len(items)
```

What are items and result?

True execution

```
items  = [1, 2]
alias  = [1, 2]
result = 2
```

Wrong prediction: treating alias as a copy.

预测执行：共享引用

预测下一状态：items = [1]; alias = items; alias.append(2); result = len(items); 问题：items 和 result 的值是什么？

真实执行：items = [1, 2]; alias = [1, 2]; result = 2。

错误预测：把 alias 当作副本。

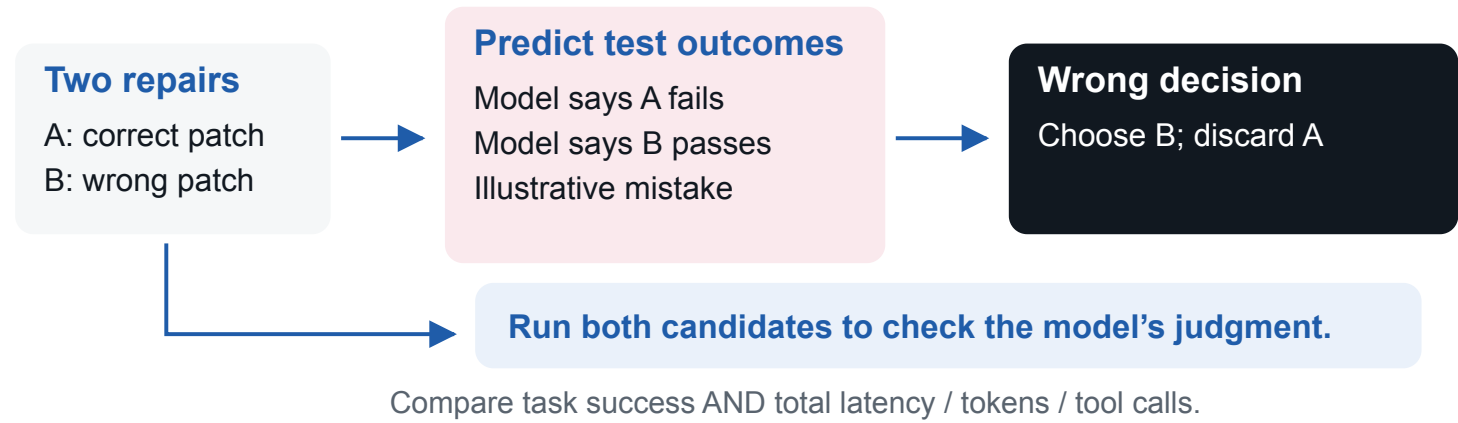
预测执行：共享引用

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

When predictions help an agent



76

预测何时能帮助智能体

两个修复方案：A 正确补丁；B 错误补丁。

预测测试结果：模型认为 A 会失败；模型认为 B 会通过。

错误决策：选择 B；丢弃 A。

图示为一种错误示例。

运行两个候选方案，检查模型的判断。

同时比较任务成功率和总延迟 / 词元数 / 工具调用数。

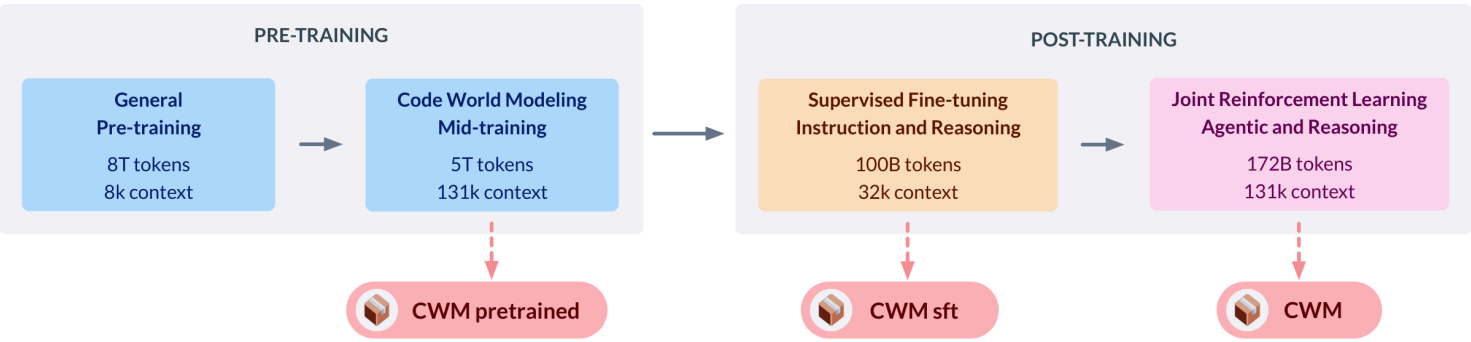
预测何时能帮助智能体

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Learning from program execution



FAIR CodeGen Team et al., 2025 · CWM ↗

77

从程序执行中学习

预训练：通用预训练——8T 个词元，8k 上下文 → 代码世界建模中期训练——5T 个词元，131k 上下文。

后训练：监督微调——指令与推理，100B 个词元，32k 上下文 → 联合强化学习——智能体与推理，172B 个词元，131k 上下文。

阶段检查点：CWM 预训练、CWM SFT、CWM。

FAIR CodeGen 团队等，2025 · CWM ↗

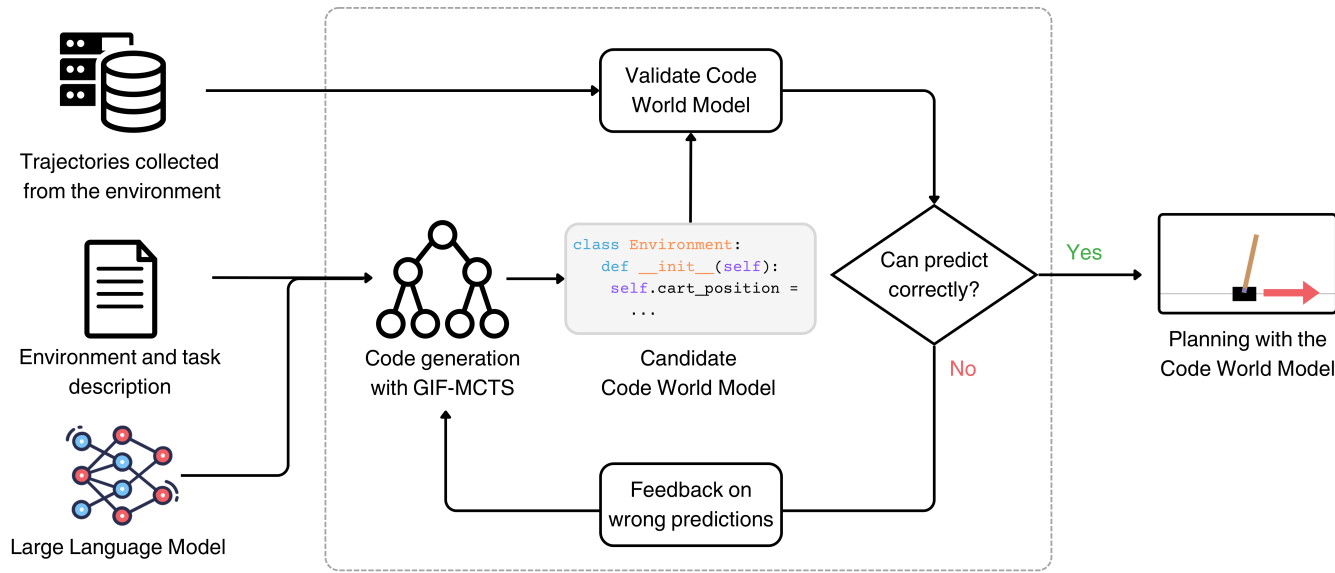
从程序执行中学习

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Building and using simulators



Dainese et al., 2024 · Code World Models / GIF-MCTS ↗

78

构建并使用模拟器

从环境收集的轨迹 + 环境和任务描述 + 大语言模型 → 使用 GIF-MCTS 生成代码 → 候选代码世界模型（示例：class Environment: def \_\_init\_\_(self): self.cart\_position = ...）。

验证代码世界模型 → 判断预测是否正确？是：使用代码世界模型进行规划；否：针对错误预测提供反馈，并重新生成代码。

Dainese 等, 2024 · Code World Models / GIF-MCTS ↗

构建并使用模拟器

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Open questions about predicting code behavior

- **Better decisions?** Measure prediction errors and task success on the same problems.
- **Lower cost?** Include latency, tokens, and real environment calls.
- **New settings?** Hold out programs, dependencies, and environments.
- **When to execute?** Vary the fallback threshold and measure failures as well as savings.

79

预测代码行为的开放问题

更好的决策？在相同问题上测量预测错误与任务成功率。

更低的成本？将延迟、词元数和真实环境调用都计入。

新的设置？留出程序、依赖项和环境。

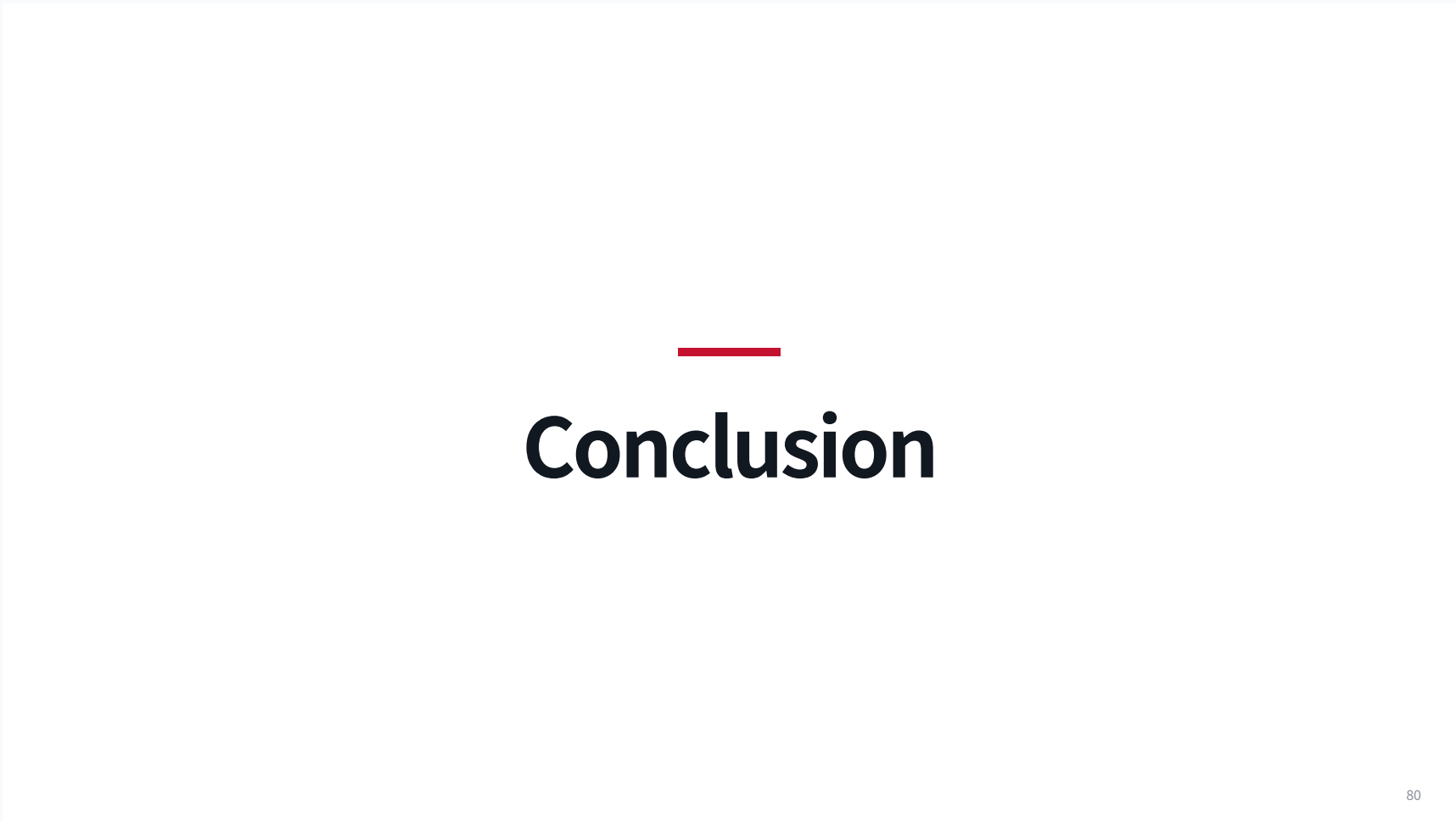
何时执行？改变回退阈值，并同时衡量失败与节省。

预测代码行为的开放问题

## 第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分



总结

总结

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Designing a coding agent

- **Code models:** pre-training, mid-training, and RL from execution rewards.
- **Agentic coding:** localize, edit, and verify with the right tools.
- **Agent training:** runnable tasks, repair rewards, and diverse harnesses.
- **Frontend development:** browser interaction and visual verification.
- **Broader development:** localization, apps, libraries, maintenance, tests, and CI.
- **Code-behavior prediction:** learned models that support execution and planning.

81

设计编码智能体

代码模型：预训练、中期训练，以及基于执行奖励的 RL。

智能体式编码：使用恰当的工具完成定位、编辑与验证。

智能体训练：可运行任务、修复奖励和多样化的 执行框架（harness）。

前端开发：浏览器交互与视觉验证。

更广泛的软件开发：代码定位、应用、库、维护、测试和 CI。

代码行为预测：用于支持执行和规划的学习模型。

设计编码智能体

第 6 讲 · 面向编码与软件开发的智能体

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分



# Questions

Next Class: Domains 2 — GUI Agents

82

问题

下一讲：领域 2——图形界面智能体（GUI Agents）

问答