ChatUniTest：基于大语言模型的测试生成框架

Yinghao Chen 浙江大学 中国
yh_ch@zju.edu.cn

Zehao Hu 浙江大学 中国
Huzehao@zju.edu.cn

Chen Zhi*浙江大学 中国
zjuzhichen@zju.edu.cn

Junxiao Han 杭州城市大学 中国
hanjx@hzcu.edu.cn

Shuiguang Deng 浙江大学 中国
dengsg@zju.edu.cn

Jianwei Yin 浙江大学 中国
zjuyjw@cs.zju.edu.cn

arXiv:2305.04764v2 [cs.SE] 7 May 2024

摘要

单元测试是一项必不可少但往往十分艰巨的任务。为缓解这一挑战，人们已提出各种自动化单元测试生成工具。尤其是，近年来，基于大语言模型（LLM）的方法受到了广泛关注，并展现出良好的效果。然而，基于 LLM 的工具在生成准确的单元测试方面仍存在局限性。本文介绍了 ChatUniTest，一种基于 LLM 的自动化单元测试生成框架。ChatUniTest 引入自适应焦点上下文机制，以在提示中纳入有价值的上下文信息，并遵循生成-验证-修复机制，以纠正生成的单元测试中的错误。随后，我们开发了 ChatUniTest Core，这是一个实现核心工作流的通用库，并配套构建了 ChatUniTest 工具链，该工具链由一组无缝集成的工具组成，用于增强 ChatUniTest 的能力。我们的有效性评估表明，在所评估项目中的一半项目中，ChatUniTest 的表现优于 TestSpark 和 EvoSuite，并实现了最高的总体行覆盖率。此外，用户研究的结果表明，ChatUniTest 能够为软件测试领域的各类利益相关者带来显著价值。ChatUniTest 可通过 https://github.com/ZJU-ACES-ISE/ChatUniTest 获取，演示视频可通过 https://www.youtube.com/watch?v=GmfxQUqm2ZQ 观看。

CCS 概念

• **软件及其工程 → 软件测试与调试**。

关键词

大语言模型、自动化单元测试生成

ACM 引用格式：

Yinghao Chen、Zehao Hu、Chen Zhi、Junxiao Han、Shuiguang Deng 和 Jian- wei Yin。2024。ChatUniTest：一种基于大语言模型的测试生成框架。载于第 *32* 届 *ACM* 国际会议伴随论文集 软件工程基础会议（*FSE Companion '24*），*2024*年*7*月*15*日至*19*日，巴西 *Porto de Galinhas*。ACM，纽约，美国纽约州，5 页。 https://doi.org/10.1145/3663529.3663801

*Chen Zhi 是通讯作者。

特此免费授予许可，允许为个人或课堂使用而制作本作品全部或部分内容的数字或纸质副本，前提是这些副本不得用于营利或商业利益，且副本首页须包含本声明及完整引文。必须尊重本作品中由作者以外的其他方拥有版权的组成部分。注明出处后允许摘录。若以其他方式复制、再版、发布到服务器或分发至列表，则须事先获得具体许可和/或支付费用。请通过 permissions@acm.org 申请许可。*FSE Companion '24*，*2024*年*7*月*15*日至*19*日，*Porto de Galinhas*，巴西 ©2024 版权所有归所有者/作者所有。出版权授权给 ACM。ACM ISBN 979-8-4007-0658-5/24/07https://doi.org/10.1145/3663529.3663801

1 引言

单元测试是软件开发中的一项关键实践，有助于确保软件应用程序的质量。然而，手动编写和维护高质量的单元测试可能是一项艰巨的任务。自动化单元测试生成工具应运而生，旨在减轻开发人员编写必要单元测试的负担。现有工具 [4, 6, 8, 17, 20] 主要依靠程序分析技术来生成单元测试。例如，EvoSuite [17], 这一著名的基于搜索的工具，展现了实现合理覆盖率的有效性。然而，这些工具生成的测试往往缺乏可解释性和可读性，这是它们的一个常见缺点。

近年来，利用深度学习技术，尤其是大语言模型（LLM）生成单元测试的探索展现出巨大潜力。TestPilot [26], 是 GitHub 开发的一款领先的基于大语言模型的工具，它引入了一种基于 Codex [16] 的自适应测试生成机制。TestSpark [9], 是一款 JetBrains IDEA 插件，提供两种测试生成模式：一种利用 OpenAI 和 JetBrains 的 AI 进行基于大语言模型的生成，另一种则使用 EvoSuite 进行基于搜索的生成。

然而，我们发现基于大语言模型的工具存在两个主要局限。首先，上下文长度限制了大语言模型处理所有相关信息并生成全面单元测试的能力。尽管近期的大语言模型能够处理更长的输入上下文，但出于经济成本以及“中间信息丢失”效应 [19]的考虑，向大语言模型提供简洁且精确的上下文仍然至关重要。其次，由于验证机制不足，大语言模型经常生成包含各种错误的不正确测试，例如在编译期间出现“找不到符号”，以及在运行时出现“*AssertionFailedError*”，需要投入大量精力进行人工修复。

本文介绍了 ChatUniTest，这是一种基于大语言模型自动生成单元测试的框架。ChatUniTest 采用自适应焦点上下文生成机制来克服第一个局限，并实现了生成-验证-修复机制来解决第二个局限。这些改进显著提升了大语言模型生成单元测试的效果。我们的工作主要贡献如下：

- 我们推出了 ChatUniTest，这是一个基于大语言模型的单元测试生成框架。通过采用自适应焦点上下文和生成-验证-修复等创新机制

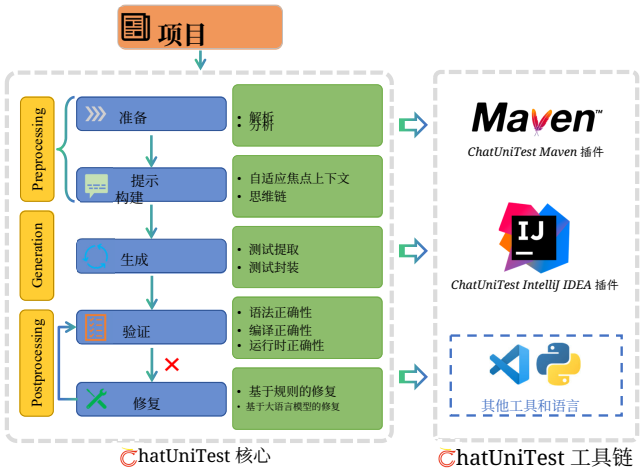

图1：chatunitest-core 及其衍生工具概览

机制，我们开发了 **ChatUniTestCore** [10]，以协助研究人员和工具构建者。

- 我们已实现**ChatUniTest 工具链**，其中包括 Maven 插件 [11] 和 IntelliJ IDEA 插件 [12]。它为用户提供便利，并满足各种场景的需求。

2 相关工作

自动化单元测试生成的研究历史跨越了数十年。然而，传统的 [17, 20] 单元测试生成方法在可解释性和可读性方面存在显著不足。近年来，大语言模型在各种编程任务中展现出卓越的性能。为此，研究人员提出了许多基于这些模型的测试生成方法 [9, 15, 22, 23,25–30]，能够生成解释充分且易于阅读的单元测试。值得注意的是，TestPilot[26] 和 TestSpark[9] 是典型的基于大语言模型的测试生成解决方案，并为此提供了用户友好的服务或工具。TestPilot[26] 可自动为 JavaScript 程序生成单元测试。它利用从文档中提取的函数签名、实现和用法示例。然而，它依赖文档，因此对于没有文档或文档较少的程序，适用性有限。TestSpark[9] 在 IntelliJ IDEA 中引入了一种新颖的单元测试生成方法。TestSpark 同时利用基于搜索的测试生成工具和大语言模型（LLM），在集成开发环境与大语言模型之间建立反馈循环，从而提高测试生成的效果。

3 方法

如图1所示，**ChatUniTest Core** 建立了面向研究人员和工具构建者的整体架构。该架构包含三个阶段和五个步骤：预处理（包括准备和提示构造）、生成，以及后处理（包括验证和修复）。预处理阶段主要关注如何释放大语言模型的能力，例如设计高效且有效的提示。后处理阶段主要关注如何缓解大语言模型的局限性，例如修复生成的测试用例。

3.1 准备工作

ChatUniTest 首先需要进行准备工作，以优化并加速生成过程。这包括解析目标项目、分析依赖项，以及对代码进行混淆以保护隐私。

解析。 解析步骤的主要目标是从代码中提取原始信息。在此阶段，ChatUniTest 会扫描项目文件夹，以识别所有 Java 文件。随后，每个 Java 文件都会被转换为抽象语法树（AST）。ChatUniTest 遍历 AST 时，会收集类和方法层面的信息。对于类，会记录包、导入项、继承关系、实现关系、字段和方法签名。对于方法，则会记录方法签名、方法体、字段访问、getter/setter 调用、依赖类名和方法调用。

分析。 分析步骤主要侧重于运用静态分析技术提取更深入的见解，以促进单元测试生成。例如，我们计划通过程序切片提取 API 调用示例代码，并进行依赖分析以构建对象构造图（OCG [18]），从而为大语言模型生成驱动代码提供指导。

3.2 提示构造

ChatUniTest 采用自适应焦点上下文生成机制和提示模板，为大语言模型构造输入提示。自适应焦点上下文生成机制旨在克服主要限制。该机制会构造一个上下文，其中包含有关焦点方法的大量关键信息，同时排除焦点类中的无关信息。提示模板提供了一种针对不同需求设计特定提示的方法，也可以直接使用默认模板。

自适应焦点上下文生成。 自适应焦点上下文生成机制旨在确保每次添加的信息都不超出令牌限制，同时尽可能向提示中加入有价值的信息。这种方法会优化焦点方法的上下文，以便整合类上下文和相关方法的上下文。

该机制根据焦点方法的依赖关系动态生成上下文。初始上下文包含焦点方法和焦点类的信息。根据依赖分析的结果，还可能将其他相关信息（例如依赖方法和类的签名）整合到上下文中。

未来，我们计划改进自适应焦点上下文生成机制。我们的最终目标是根据每个焦点方法的属性生成最优上下文，使大语言模型能够更清晰地理解上下文，进而提升生成的测试的质量。

根据自适应焦点上下文生成机制，ChatUniTest 首先解析模板文件，并将标签和占位符替换为实际数据。如果提示中的词元数量超出令牌限制，ChatUniTest 将自动从用户提示中删除最不相关的数据，通常是最后一个标签或占位符。这样，提示中的词元数量便不会超出限制。

提示模板。 ChatUniTest 使用 FreeMarker Java 模板引擎 [2] 根据模板和

变化的上下文生成提示。这种方法结合自适应焦点上下文机制，既让用户能够灵活设计符合自身需求的多样化提示模板，又为大语言模型预留了充足的词元，以生成有用的响应。

思维链。 基于预处理期间构建的依赖图，我们正在实现用于创建思维链（COT）提示的方案。先前的研究 [21] 已证明，思维链能够有效提升大语言模型（LLM）处理软件工程任务的能力。我们认为，思维链可以帮助大语言模型更全面地理解代码，进而提高生成的测试的质量。

3.3 生成

生成组件利用生成的焦点上下文和用户提供的提示模板（若未提供，则使用默认模板），将上下文中的关键信息整合到提示中。为确保测试用例生成有足够的词元，设置了一个固定阈值来控制提示可占用的词元数量。随后，在该阈值范围内填入上下文信息。之后，通过大语言模型 API 调用大语言模型。

收到响应后，ChatUniTest 会使用其 CodeExtrac- tor 组件从响应中提取测试。我们的实验表明，大语言模型可能会提供单个测试方法，也可能提供完整的测试类。为此，我们实现了 TestSkeleton，它可以自动封装测试方法以构建完整的测试类。生成测试类后，测试生成过程即告完成。

受 CAT-LM [23], 的启发，我们也构建了**ChatUniTest 模型** [13], ，它基于**Code Llama**[24]为 Java 测试生成任务提供微调模型。

3.4 语法、编译和运行时验证

提取出的测试将被转交给验证组件，以进一步验证其正确性。验证过程包含三个子步骤：语法验证、编译验证和运行时验证。只有通过所有验证步骤，测试才被视为成功生成。

语法验证。ChatUniTest 首先使用 Java Parser 验证生成的测试的语法。如果在此阶段检测到语法错误，通常可以由基于规则的修复组件进行修复。例如，JavaParser 中的错误“解析错误。发现 *<EOF>*”表明代码中可能缺少*“;”*、*“}”*或其他闭合符号。

编译验证。 语法验证完成后，ChatUniTest 会继续编译测试。此步骤不仅会检查语法错误，还会检查代码中的某些语义问题。如果编译失败，工具便会进入修复阶段。例如，错误 "找不到符号 " 表示可能缺少依赖项。通常可以通过添加合适的导入语句来解决。

运行时验证。 完成编译验证后，ChatUniTest 开始执行测试。如果遇到任何运行时错误，该工具将进入修复阶段。例如，错误 *"org.mockito.exceptions.misusing"* 表明 Mockito 方法使用不当。例如，使用 *when()* 方法时，未将模拟对象上的方法调用作为参数传入。

3.5 基于规则和基于大语言模型的修复

如果验证步骤中出现错误，ChatUniTest 会启动修复过程进行纠正。修复策略分为两种：基于规则的修复和基于大语言模型的修复。前者用于修复特定错误，后者则作为备用措施，用于修复范围更广的意外错误。

基于规则的修复。 ChatUniTest 首先使用其基于规则的修复组件来纠正测试中的特定错误。从概念上讲，修复规则由两部分组成：触发器和 Transformer模型。触发器定义何时应用修复规则，而 Transformer模型则规定如何重写代码以解决错误。目前，ChatUniTest 实现了两条简单的修复规则。

- 当大语言模型的响应超过令牌限制时，可能导致测试被截断和语法错误。为解决这一问题，ChatUniTest 首先添加必要的结尾大括号，以验证代码结构。如果验证失败，则搜索 *" @Test"* 注解，截断其之前的代码，并添加闭合大括号。如果修复失败，或者移除后不再有测试，则终止该过程。
- 如果在编译验证过程中出现“找不到符号” 错误，ChatUniTest 会尝试通过将焦点类中的所有导入语句直接复制到生成的测试中来解决该问题。

基于大语言模型的修复。 如果生成的测试中仍然存在错误，那么应用所有基于规则的修复后，应将其交由基于大语言模型的修复。在此修复阶段，ChatUniTest 收集有关错误类型和错误消息的详细信息，将其与错误测试的上下文相结合，并使用预定义模板生成提示。如果测试在指定轮数后仍未解决，则终止整个过程。

4 示例说明

如图 1所示，我们基于 **ChatUniTest 工具链** 构建了 **ChatUniTest Core**。该工具链提供了针对各种场景定制的插件，为用户带来更加流畅的使用体验。为了展示 ChatUniTest 的功能，我们考察了一个由 IntelliJ IDEA 插件生成的具体测试用例。

<escape>考虑*equals* 方法位于 *StringUtils* 类中，如图2所示。该方法旨在判断给定的两个 *CharSequence* 对象*cs1* 和 *cs2*是否相等。</escape>

```
public static boolean equals(
        CharSequence cs1, CharSequence cs2) {
    如果 (cs1 == cs2) {
        返回 true;
    }
    <escape>if (cs1 == null || cs2 == null) {</escape>
        返回 false;
    }
    返回 cs1。equals(cs2);
}
```

图2：Apache Commons 中有缺陷的方法*equals*

开发者可以调用 ChatUniTest 为方法生成单元测试*equals*。图 3 展示了生成结果。生成的测试会在第 15 行抛出 *"AssertionFailedError"* ，因为

StringBuilder 实现了 *CharSequence*，但未重写 Object 类中的 *equals* 方法。因此，对于 *StringBuilder*，*equals* 方法检查的是引用相等性，而非值相等性。因此，该测试检测出了焦点方法中的缺陷。

```
public class StringUtilsEqualsTest {

    @测试(超时 = 8000)
    public void testEquals() {
        assertTrue(StringUtils.equals(null, null));
        assertFalse(StringUtils.equals("test", null));
        assertFalse(StringUtils.equals(null, "test"));
        assertTrue(StringUtils.equals("test", "test"));
        <escape>assertFalse(StringUtils.equals("test", "Test"));</escape>
        assertFalse(StringUtils.equals("测试", "test"));
        assertTrue(StringUtils.equals("", ""));
        assertTrue(StringUtils.equals(" ", " "));
        断言为假(StringUtils.equals(" ", ""));
        断言为假(StringUtils.equals("", " "));
        assertTrue(StringUtils.equals(
                new StringBuilder("test"), new StringBuilder("test")));
        <escape>assertFalse(StringUtils.equals(</escape>
                新建StringBuilder("test"), 新建 StringBuilder("Test")));
    }
}
```

图3：ChatUniTest 生成的测试用例。

5 评估

有效性评估。我们的初步评估旨在通过评估 ChatUniTest 生成的测试质量，确定其有效性。为了建立比较基准，我们选择了两种具有代表性的工具：EvoSuite（代表基于程序分析的工具）和 TestSpark（代表基于大语言模型的工具）。值得注意的是，由于 TestPilot 不支持 Java，因此未将其纳入比较。在实验设置中，我们采用 EvoSuite 的默认配置，并确保 ChatUniTest 与 TestSpark 的配置保持一致。其中包括使用 gpt-3.5-turbo-0613 作为基础模型，每轮包含一次生成尝试，随后进行五次修复尝试。这种标准化配置确保了对所选工具进行公平、无偏的评估。

如表1所示，我们为实验选取了四个 Java 项目：Commons-Cli[1]，Commons-Csv[3]，电子商务微服务[5]，和 Binance-connector[7]。前两个项目在相关工作中被广泛使用[15, 27]，也很可能包含在 GPT-3.5-turbo 的训练数据中。后两个则是未见过的热门项目（星标数超过 100），不属于训练数据的一部分（因为它们是在训练数据创建之后才创建的）。考虑到实验中需要反复操作这些 IDEA 插件，我们采用了置信水平为 95%、误差范围为 5% 的随机抽样。因此，我们从这四个项目共 835 个焦点方法中随机选取了 264 个。在实验过程中，我们尝试为每个选定的方法生成一个测试。

表1：实验结果

项目概况			行覆盖率		
Name	版本	未见	ChatUniTest	TestSpark	EvoSuite
Cli [1]	1.5.0	no	70.9%	78.4%	**91.8%**
逗号分隔值 [3]	1.10.0	no	**73.3%**	-	28.3%
电子商务 [5]	695a6d4	yes	26.7%	**36.7%**	-
币安 [7]	2.0.0	yes	**49.2%**	29.7%	20.8%
总体覆盖率	-	-	**59.6%**	42.1%	38.2%

在表1中，针对四个项目对 ChatUniTest、EvoSuite 和 TestSpark 的行覆盖率进行了全面比较。值得注意的是，TestSpark 未能为 Csv项目生成测试，这归因于构建的提示超过了模型的令牌限制。此外，在Ecommerce项目中，EvoSuite 未能为焦点方法生成测试，主要原因是 EvoSuite 与 Ecommerce项目之间存在 JDK 版本兼容性问题。

总而言之，ChatUniTest 在不同项目中展现出相对可靠的行覆盖率，达到了令人印象深刻的 59.6%。这一表现优于 EvoSuite 和 TestSpark，凸显了 ChatUniTest 作为单元测试生成工具的稳健性和有效性。

有用性评估。我们开展了一项用户研究，以调查 ChatUniTest 的有用性。我们向 19 位曾为我们的项目点星、派生项目、在 GitHub 上提交问题或通过电子邮件联系我们的人士发放了问卷（详情请见 [14]）。我们共收到 9 份回复。受访者包括五名学生和四名资深软件开发人员。所有参与者都具备软件测试的基础知识，其中五人对该领域颇为熟悉。此外，每位参与者都有使用大语言模型进行编程的经验。

调查结果表明，89% 的受访者使用 ChatUniTest 辅助编写测试用例。所有受访者都认为，使用 ChatUniTest 有助于编写测试用例，其中 33% 的受访者认为它非常有帮助。对于初级开发者，尤其是学生而言，ChatUniTest 能有效地帮助他们编写单元测试。此外，33% 的受访者正在进一步拓展 ChatUniTest，例如增加对其他语言的支持、集成新的大语言模型，以及将其纳入自己的开发工作流。值得注意的是，一些受访者还提出了宝贵的建议，包括集成性能测试并支持更多测试框架，尤其是内部测试框架。他们还建议在真实的生产环境中衡量并验证生成的测试代码的可靠性。这些建议不仅为 ChatUniTest 提供了指导，也有助于该领域的未来发展。总体而言，这些结果体现了用户对 ChatUniTest 的高度认可，以及他们对其未来发展的期待。

6 结论与未来工作

我们提出了 ChatUniTest，这是一种自动化单元测试生成框架，利用大语言模型的能力，并提供一套易于使用的 API，帮助开发者完成软件测试任务。未来，我们计划改进该框架，以生成最优上下文，并扩展准备和验证流程，从而支持更多编程语言。此外，我们还计划提供更广泛的基准实现（例如 SysPrompt [25] 和 ChatTester [30]），这些实现均基于 ChatUniTest 构建。

致谢

本研究部分得到了国家自然科学基金委员会项目（U20A20173 和 62125206）、国家重点研发计划项目（2022YFF0902702）、浙江省尖兵（尖兵）项目（2023C01045）、宁波市重点研发计划项目（2023Z235）以及浙江大学-恒生金融科技研发中心的资助。

参考文献

[1] 2007. https://github.com/apache/commons-cli. [2] 2007. Apache FreeMarker. https://freemarker.apache.org/. [3] 2014. https://github.com/apache/commons-csv. [4] 2017. https://github.com/wrdv/testme-idea. [5] 2021. https://github.com/SelimHorri/ecommerce-microservice-backend-app. [6] 2022. https://github.com/UnitTestBot/UTBotJava. [7] 2022. https://github.com/binance/binance-connector-java. [8] 2023. https://github.com/ArtemUntila/kex-intellij-plugin. [9] 2023. https://github.com/JetBrains-Research/TestSpark. [10] 2023. https://github.com/ZJU-ACES-ISE/chatunitest-core. [11] 2023. https://github.com/ZJU-ACES-ISE/chatunitest-maven-plugin. [12] 2023. https://github.com/ZJU-ACES-ISE/ChatUniTest_IDEA_Plugin. [13] 2024. https://github.com/ZJU-ACES-ISE/chatunitest-models. [14] 2024. https://github.com/ZJU-ACES-ISE/chatunitest-core/tree/main/docs/ 调查。 [15] Saranya Alagarsamy、Chakkrit Tantithamthavorn 和 Aldeida Aleti。2023。A3Test：断言增强的自动化测试用例生成。arXiv 预印本 arXiv:2302.10352（2023）。[16] Mark Chen、Jerry Tworek、Heewoo Jun、Qiming Yuan、Henrique Ponde de Oliveira Pinto、Jared Kaplan、Harri Edwards、Yuri Burda、Nicholas J oseph、Greg Brockman 等。2021。评估基于代码训练的大语言模型。arXiv 预印本 arXiv:2107.03374（2021）。[17] Gordon Fraser 和 Andrea Arcuri。2011。EvoSuite：面向对象软件的自动化测试套件生成。载于第 19 届 ACM SIGSOFT 研讨会暨第 13 届欧洲软件工程基础会议论文集。416–419。[18] Yun Lin、You Sheng Ong、Jun Sun、Gordon Fraser 和 Jin Song Dong。2021。基于图的种子对象合成，用于基于搜索的单元测试。载于第 29 届 ACM 欧洲软件工程会议与软件工程基础研讨会联合会议论文集。1068–1080。[19] Nelson F Liu、Kevin Lin、John Hewitt、Ashwin Paranjape、Michele Bevilacqua、Fabio Petroni 和 Percy Liang。2023。迷失在中间：语言模型如何使用长上下文。arXiv 预印本 arXiv:2307.03172（2023）。[20] Carlos Pacheco 和 Michael D Ernst。2007 年。Randoop：面向 Java 的反馈导向随机测试。载于《第 22 届 ACM SIGPLAN 面向对象编程系统与应用大会伴随会议论文集》。815–816。[21] Yun Peng、Chaozheng Wang、Wenxuan Wang、Cuiyun Gao 和 Michael R Lyu。2023 年。面向 Python 的生成式类型推断。arXiv 预印本 arXiv:2307.09163（2023）。[22] Juan Altmayer Pizzorno 和 Emery D Berger。2024 年。CoverUp：覆盖率引导的基于大语言模型的测试生成。arXiv 预印本 arXiv:2403.16218（2024）。[23] Nikitha Rao、Kush Jain、Uri Alon、Claire Le Goues 和 Vincent J Hellendoorn。2023 年。CAT-LM：在对齐的代码与测试上训练语言模型。载于《2023 年第 38 届 IEEE/ACM 自动化软件工程国际会议（ASE）》。IEEE，409–420。[24] Baptiste Roziere、Jonas Gehring、Fabian Gloeckle、Sten Sootla、Itai Gat、Xiao-qing Ellen Tan、Yossi Adi、Jingyu Liu、Tal Remez、Jérémy Rapin 等。2023 年。Code Llama：代码开放基础模型。arXiv 预印本 arXiv:2308.12950（2023）。[25] Gabriel Ryan、Siddhartha Jain、Mingyue Shang、Shiqi Wang、Xiaofei Ma、Mu-rali Krishna Ramanathan 和 Baishakhi Ray。2024 年。代码感知提示：使用大语言模型在回归测试场景中进行覆盖率引导的测试生成研究。arXiv 预印本 arXiv:2402.00097（2024）。[26] Max Schäfer、Sarah Nadi、Aryaz Eghbali 和 Frank Tip。2023 年。使用大语言模型进行自适应测试生成。arXiv 预印本 arXiv:2302.06527（2023）。[27] Michele Tufano、Dawn Drain、Alexey Svyatkovskiy、Shao Kun Deng 和 Neel Sundaresan。2020 年。使用 Transformer 模型和焦点上下文进行单元测试用例生成。arXiv 预印本 arXiv:2009.05617（2020）。[28] Junjie Wang、Yuchao Huang、Chunyang Chen、Zhe Liu、Song Wang 和 Qing Wang。2024 年。使用大语言模型进行软件测试：综述、现状与展望。IEEE软件工程汇刊（2024）。[29] Chen Yang、Junjie Chen、Bin Lin、Jianyi Zhou 和 Ziqi Wang。2024 年。通过程序分析增强难覆盖分支的基于大语言模型的测试生成。arXiv 预印本 arXiv:2404.04966（2024）。[30] Zhiqiang Yuan、Yiling Lou、Mingwei Liu、Shiji Ding、Kaixin Wang、Yixuan Chen 和 Xin Peng。2023 年。不再需要手动测试？评估并改进 ChatGPT 的单元测试生成。arXiv:2305.04207 [cs.SE]

收稿日期：2024年1月29日；录用日期：2024年4月15日