**Self-collaboration Code Generation via ChatGPT**

基于ChatGPT的自协作代码生成

**太长不看版：**

作者在文中提出了自协作代码生成框架，引入了简单瀑布模型，通过角色指令基于ChatGPT生成了分析员、编码员和测试员三个角色扮演的LLM实例，并使用指令引导不同角色模型间进行协作交互，完成代码生成工作。作者使用自协作框架分别在HumanEval、HumanEval-ET、MBPP和MBPP-ET四个基准数据集中进行测试，性能比直接代码生成提高了29.9% ~ 47.1%，超过了GPT-4。

**摘要**

作者提出了一种新的代码生成框架，它借鉴了人类在软件开发中的团队协作方式。在这个框架中，以ChatGPT为例，使用多个大型语言模型实例（LLMs）扮演不同的“专家”，每个专家负责复杂任务中的一个特定子任务。具体来说，作者设定了三个ChatGPT的角色：分析员、编码员和测试员，分别负责软件开发的分析、编码和测试工作。并通过指定不同角色之间的协作和交互方式，形成一个虚拟团队，共同完成代码生成任务，而无需人类干预。实验结果表明，这种自我协作的代码生成方法在各种代码生成基准测试中表现优异，性能比直接代码生成提高了29.9% ~ 47.1%，甚至超过了GPT-4。此外，研究还展示了这种方法在处理现实世界中的复杂任务上的潜力。

1. **引言**

团队协作有多种方面的好处，(1). 可以将复杂任务分解为更小的子任务。(2). 团队成员可以对代码进行评审和测试。(3). 确保生成的代码与预期要求一致，团队成员可以提供不同的观点解决问题。

基于团队协作的理念，这个框架被分为两个部分：劳动分工和协作，这两个部分都由角色指令主导。(1). 通过角色指令分配特定的角色和责任给LLM实现劳动分工，将LLM转化为该领域的专家。(2). 角色指令控制角色之间的交互形成一个虚拟团队促进各自的工作。

为了更高效的协作，作者引入了自协作框架 (SDM)，实例化了三个角色：分析人员、编码人员和测试人员组成基本团队。这些角色遵循一个固定的工作流，每个阶段都向前一个阶段提供反馈。即分析人员分解需求、制定计划，编码人员根据计划创建、改进代码，测试人员记录测试发现的问题并撰写报告。

该方法可以在多个代码生成基准测试上获得最优性能，并且在选定的现实场景的复杂代码生成任务中表现出显著的有效性。

1. **自协作框架**

作者的自协作框主要分为两个部分：劳动分工和协作。设给定一个需求x，期望的输出为y，定义任务为。

* 1. 劳动分工（DOL）

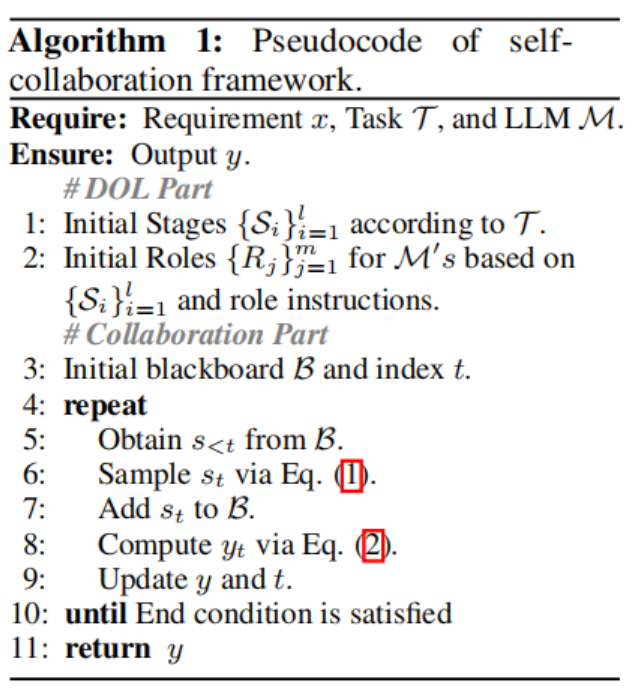
在DOL部分，论文利用先验知识将解决复杂任务的过程分解为一系列阶段: 基于这些阶段构建了一些不同的角色。每个阶段可以由一个或多个角色处理。作者使用角色指令将角色和责任分配给LLM，值得注意的是，指令中描述的任务越详细，LLM的行为就越符合期望。通过角色扮演，可以有效地将LLMs置于特定领域，并激发其在该领域内的专业知识。作者通过实验发现，与直接让LLM参与任务相比，这种角色扮演方法产生了更好的结果。需要注意的是，角色指令只需要在每个LLM代理初始化时提供一次，后续交互中角色将自动交互并传递信息，无需进一步的指令，从而提高了后续交互和协作的整体效率和清晰度。

* 1. 协作

在DOL部分为LLM分配角色后，随着各阶段的进展，各角色将其产出与其他角色互动。在协作部分，我们的重点是促进不同角色之间的有效互动，确保它们相互促进。角色之间的交互以自然语言的形式进行，而自然语言则由语言模型的基础部分提供支持。我们在角色指令中规定了每个角色交互的角色、信息和格式，从而使整个协作过程得到很好的控制，协作部分可以被写为如下形式：

代表着阶段的输出，代表着先前阶段的输出，作者使用作为（当前阶段角色）与先前阶段角色的输出交互生成的计算方法。输出y使用如下的公式随着阶段的推进而不断更新：

f是一个更新函数，一旦结束的条件满足，最终的y将作为结果被输出。协作逻辑的伪代码表示如算法1所示。



算法1：协作逻辑的伪代码。

1. **实例**

在具体的角色设计和协作流程上，作者设计了简单的瀑布模型，将任务划分为三个阶段，即分析、编码和测试。工作流遵循从上一个阶段流向下一个阶段，如果发现问题，则回到上一个阶段进行细化。基于这样的理念，作者构建了一个基本的团队，由分析人员、编码人员和测试人员组成，分别负责分析、编码和测试三个阶段，这三种不同的角色被分配了以下的任务。

**分析人员（Analyst）：**其目标是开发高层次的计划，并指导编码员编写程序，而不是深入到实现细节。分析师会将一个需求（x）分解成几个易于解决的子任务，以便于功能单元的划分，并制定实现的主要步骤的高层次计划。

**编码员（Coder）：**作为团队的核心角色，编码员接收分析师的计划或测试员的测试报告。编码员的责任包括：根据分析师提供的计划编写满足特定需求的代码，以及根据测试员反馈的测试报告来修复或完善代码。

**测试员（Tester）：**测试员获取编码员编写的代码，并撰写包含功能性、可读性和可维护性等方面的测试报告。该模型提倡使用模拟测试过程的方式生成测试报告，以促进交互并避免外部工作量。

此外，该框架还定制了适用于大型语言模型（如 ChatGPT）来扮演这三个角色的角色指令如图1所示。举例来说，编码员的角色指令不仅包括角色描述和责任，还包括团队描述和用户需求，这些共同作用于初始化 ChatGPT 代理，设定其行为方式。角色之间也存在交互，尤其是两个连续阶段的负责角色之间，并且限制最大交互次数为 n。输出只在编码阶段 更新，当达到n 次交互或测试员确认没有问题时，工作流程终止。

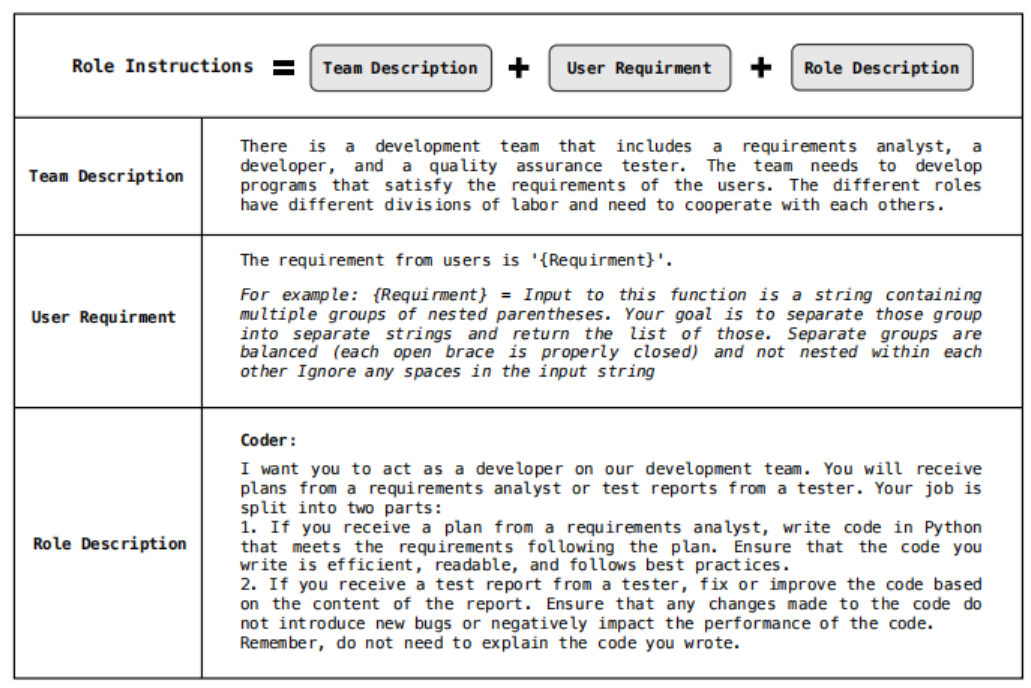


图1. 角色指令示例。

1. **实验设计**

**4.1 基准测试：**

**MBPP (sanitized version)：**包含427个手动验证的Python编程任务，涵盖编程基础、标准库功能等。每个任务包含一个自然语言描述、代码解决方案和3个自动化测试用例。

**HumanEval：**包含164个手写编程任务。每个任务包括一个函数签名、自然语言描述、函数体和若干单元测试（平均每个任务7.7个）。

**MBPP-ET 和 HumanEval-ET：** MBPP 和 HumanEval 的扩展版本，每个任务增加了超过100个测试用例。这个更新版本包括边缘测试用例，与原始基准相比提高了代码评估的可靠性。

**4.2 设置和基线：**

**设置1：** 包括NL、函数签名和公共测试用例，用于与其他代码生成方法LLMs进行比较。

**设置2：**仅限制NL作为输入提示，更符合现实世界的开发场景。

**基线：**code (AlphaCode [Li et al., 2022], Incoder [Fried et al., 2022], CodeGeeX [Zheng et al., 2023], CodeGen [Nijkamp et al., 2022], 和 CodeX [Chen et al., 2021]), 以前最先进的方法 (CodeX + CodeT [Chen et al., 2022]), 和通用LLMs (ChatGPT [OpenAI], and GPT-4 [OpenAI, 2023]).

所有实验中，通过其 API 调用了 ChatGPT (GPT-3.5) 的 '0301' 版本（称为 gpt-3.5-turbo），以减少模型变化对结果的意外影响。为了提高大型语言模型输出的稳定性，将解码温度设置为0。除非另有说明，研究使用仅限NL的设置进行代码生成，LLMs之间的最大交互次数限制为4。

**4.3评价：**

通过执行测试用例来衡量 top-k 生成代码的功能正确性，可表述为

1. **实验结果**
   1. **自协作框架的性能提升：**

实验显示该自协作框架显著提高了基础大型语言模型（LLM）的性能。特别是，包括分析师、编码员和测试员的简单三人团队基于 ChatGPT (GPT-3.5) 的自协作代码生成，在四个代码生成基准测试中都取得了最佳性能，甚至超过了 GPT-4。与仅使用 ChatGPT (GPT-3.5) 相比，该框架提供的改进相当显著，相对增幅在 29.9% 到 34.6% 之间。同时，自协作代码生成在与扩展测试用例相关的数据集（如 HumanEval-ET 和 MBPP-ET）上取得了更显著的改进，表明该框架能有效协助基础 LLM 生成更可靠的代码。性能比较如表1所示

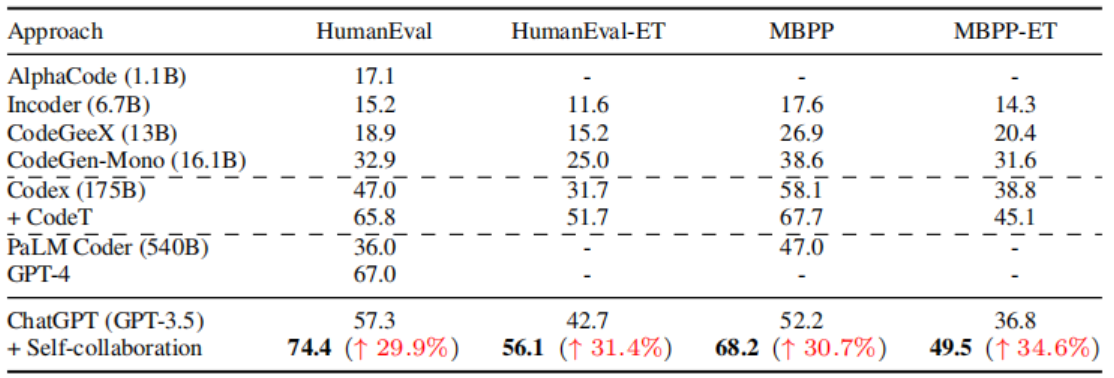


表1. 自协作框架与不同模型在基准测试上的性能比较。

* 1. **添加不同角色的性能评估**

实验评估了在仅依赖自然语言描述（NL）的设置中，每个 ChatGPT 角色在自协作框架中的性能。结果显示，与仅使用编码员角色相比，组建团队后（无论是两角色还是三角色团队）性能都有显著提升。特别是在 HumanEval 和 HumanEval-ET 基准上，编码员-分析员-测试员团队取得了最好的结果，如表2所示。

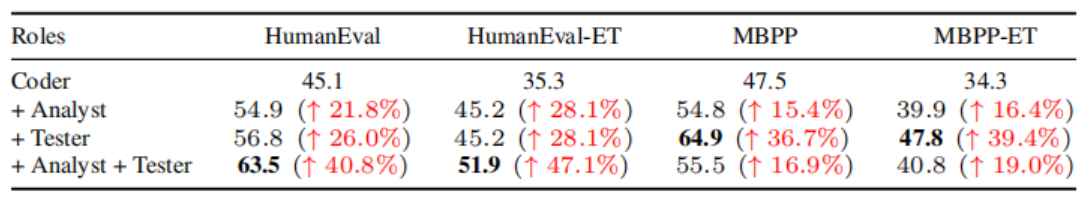


表2. 添加不同角色在基准测试上的性能比较。

* 1. **角色扮演策略的有效性验证**

为了进一步验证角色扮演策略的有效性，与两个无角色扮演方法（零样本指令和少样本提示）进行比较分析。结果表明，角色扮演方法显著优于无角色扮演的基线，如表3所示。

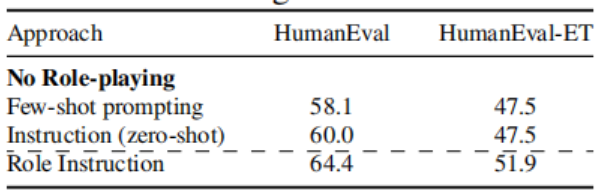


表3. 与无角色扮演方法在基准测试上的性能比较。

* 1. **互动对自协作代码生成的影响**

研究评估了互动在自协作代码生成中的影响。实验结果显示，MI（互动次数）值从0增加到1时性能提升最大。这表明在这些基准测试中，大部分任务在两轮（即一轮互动）内就得到解决。MI值的进一步增加虽然带来的性能提升逐渐减少，但仍然观察到持续的改善，如表4所示。

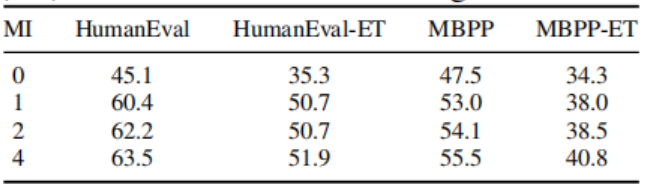


表4. 不同互动次数在基准测试上的性能比较。

* 1. **案例研究**

**HumanEval 基准上的案例研究：**作者在 HumanEval 基准测试上评估了自协作代码生成。案例研究展示了该方法的工作流程和性能。首先，分析师对需求进行全面分析，并制定解决整套需求的计划。然后，编码员根据分析师提出的分解和高层次计划实现代码。测试员为实现的代码编写详细的测试报告，并识别代码中的错误，如建议去除代码中的 "lst = list(set(lst))" 行。最后，编码员根据测试报告的反馈对代码进行了修改，并由测试员确认修改后的代码无误，所有测试都通过，从而完成了代码生成过程，如图2所示。

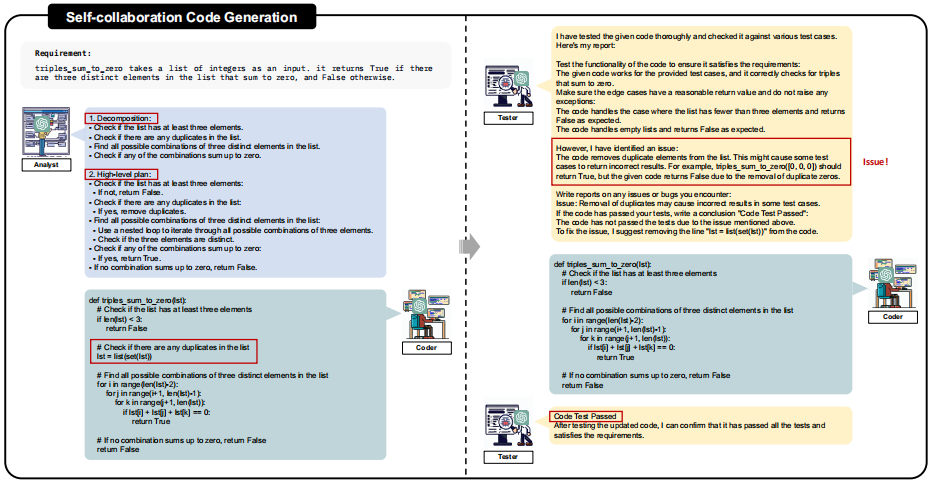


图2. HumanEval 基准上的案例研究。

**复杂任务的案例研究：**作者应用自协作代码生成方法来处理一个复杂的游戏开发需求。自协作方法生成了一个满足所有需求的游戏，无需人为干预。首先，方法正确实现了游戏逻辑，包括鼠标点击开始游戏，通过障碍物，避开炸弹，到达终点等。其次，严格遵守了需求中的规格说明，包括开始和结束点的视觉表现，游戏资源的加载，以及图像的适当缩放。此外，它还注意到了一些需求中没有提到但符合常识的游戏逻辑，例如“炸弹从屏幕顶部掉落，触底后重置位置”。相比之下，直接生成的结果只是一个Python脚本的草稿，没有包含需求中要求的所有功能。即使手动输入指令“继续添加功能”，ChatGPT 仍无法令人满意地完成这个需求，如图3所示。

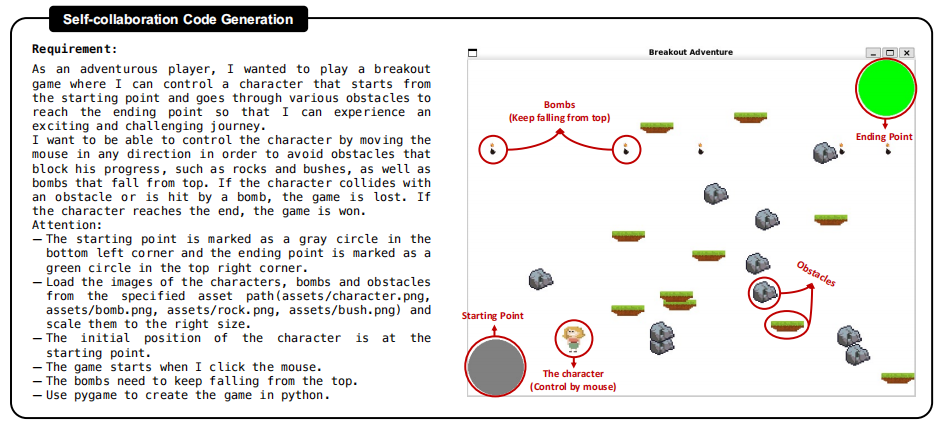


图3. 复杂任务的案例研究。

1. **讨论和未来的工作**

作者讨论了目前实例化的虚拟开发团队的一些局限性，并为未来的研究提供了一组潜在的方向。(1). 自我协作框架允许轻松地修改团队组成，可以创建新的软件开发模型和由全新的角色组成的虚拟软件开发团队。(2). 自我协作框架可能会脱离需求，并影响系统的有效性。

1. **总结**

在本文中，作者提出了一个自我协作框架，通过协作和交互的方式增强LLM解决问题的能力。基于所提出的框架，作者组装了一个由三个不同的ChatGPT角色组成的基本团队，协同处理代码生成任务。大量的实验结果证明了自我协作框架的有效性和通用性。