评估代码内容上的人工智能生成内容 检测器

Jian Wang
jian004@e.ntu.edu.sg 南洋理工大学 新加坡

Shangqing Liu
liu.shangqing@ntu.edu.sg 南洋理工大学 新加坡

Xiaofei Xie
xfxie@smu.edu.sg 新加坡管理大学 新加坡

Yi Li yi_li@ntu.edu.sg 南洋理工大学 新加坡

arXiv:2304.05193v1 [cs.SE] 11 Apr 2023

摘要

人工智能生成内容（AIGC）凭借其令人印象深刻的性能获得了广泛关注，其中 ChatGPT 已成为领先的 AIGC 模型，能够在包括软件开发和维护在内的各种应用中生成高质量的响应。尽管 ChatGPT 潜力巨大，但其滥用也引发了重大担忧，尤其是在教育和安全关键领域。目前，已有许多 AIGC 检测器被开发出来，并在自然语言数据上进行了评估。然而，它们在检测 ChatGPT 生成的代码相关内容方面的表现仍未得到探索。

为填补这一空白，本文开展了首项实证研究，评估现有 AIGC 检测器在软件领域的表现。我们创建了一个综合数据集，其中包含**492.5K**个样本，涵盖 ChatGPT 生成的代码相关内容，包括问答（**115K**）、代码摘要（**126K**）和代码生成（**226.5K**）等常见软件活动。我们评估了六种 AIGC 检测器，其中包括三种商业方案和三种开源方案，并考察了它们在该数据集上的表现。此外，我们还开展了一项人工研究，以了解人类的检测能力，并将其与现有 AIGC 检测器进行比较。结果表明，与自然语言数据相比，AIGC 检测器在代码相关数据上的表现较差。微调可以提升检测器的性能，尤其是对于同一领域内的内容；但泛化仍然是一项挑战。人工评估显示，人类进行检测颇具挑战性。

1 引言

近年来，人工智能生成内容（AIGC）引起了学术界和工业界的广泛关注与兴趣。AIGC 指通过先进的生成式人工智能技术生成的内容。随着人工智能技术日益成熟，生成内容的质量显著提升，并被广泛用于各种任务。OpenAI 发布的 ChatGPT [2], 已成为最受瞩目的方法之一。ChatGPT 是一个基于 GPT-3.5 系列微调而成的大语言模型（LLM），通过基于人类反馈的强化学习（RLHF） [18, 53, 59]构建对话式人工智能系统。GPT-3.5 系列中海量的学习知识，以及基于 RLHF 精心设计的微调流程，使 ChatGPT 能够针对不同领域和情境下的用户问题生成高质量回答。

ChatGPT 在生成跨多个领域的内容方面展现出了卓越的能力。它能够理解上下文、遵循指令并生成连贯的内容，因此尤其适用于起草电子邮件、生成文章、创作诗歌、编写故事以及制作社交媒体内容等任务。此外，OpenAI 强调了 ChatGPT 在软件开发方面的能力 [44]。ChatGPT 已广泛应用于软件开发任务，例如编写文档、创建用户手册、生成代码片段、审查代码和修复代码。

尽管ChatGPT 为用户提供了诸多益处，但仍需考虑其被滥用的可能性。例如，在教育领域，学生可能会使用 ChatGPT 在考试中作弊或抄袭作业，这违反了学术诚信。为避免滥用，一些大学限制了ChatGPT 的使用，正如近期报告所示[1]。同样，在行业中，必须仔细考虑 ChatGPT 生成内容的来源，尤其是在安全和关键安全场景中。人工智能生成的内容可能质量较低或包含错误（例如有害内容或漏洞），从而导致严重后果 [14]。例如，为防止在回答问题时恶意使用 ChatGPT 生成的内容，StackOverflow 宣布暂时禁止 ChatGPT生成的内容，[2] 因为“*ChatGPT*给出正确答案的平均比率过低，发布由 *ChatGPT*创建的答案会对网站以及那些提出问题并寻找正确答案的用户造成实质性伤害”。

随着 ChatGPT 在包括软件开发在内的广泛领域中的使用日益增加，开发有效的工具来检测人工智能生成内容变得愈发重要。例如，学术界和工业界已经开发了许多 AIGC检测器 [1, 3–7、9–11, 24, 42] ，用于检测 GPT 系列模型（包括 GPT-2 [50], GPT-3 [15], 和 ChatGPT）生成的内容。尽管这些工具已被提出用于检测 ChatGPT 生成的内容，但这些工具的有效性仍不明确，尤其是在软件开发领域的背景下。

为填补这一空白，本文迈出早期一步，开展了一项全面的实证研究，以评估现有检测器，包括开源和商业检测器，在检测 ChatGPT 生成的代码相关内容（例如代码和文档）方面的能力。具体而言，本研究旨在

[1]https://www.universityworldnews.com/post.php?story=20230222132357841[2] https://meta.stackoverflow.com/questions/421831/temporary-policy-chatgpt-is-banned

回答以下关键问题：当前用于检测 *ChatGPT* 生成的代码相关内容的工具有多准确？在检测 *ChatGPT* 生成的自然语言内容和代码相关内容方面，性能有何差异？对检测工具进行微调能否提升其识别 *ChatGPT*生成的内容的能力？检测工具在检测基于*ChatGPT*生成的内容修改而成的内容时具有多强的鲁棒性？与现有的人工智能生成内容检测器相比，人类的检测能力如何？

为开展这项研究并回答这些问题，我们通过分别在编程和自然语言领域使用 ChatGPT 生成相关内容，构建了两个数据集，即代码相关内容数据集（*CCD*）和自然语言相关内容数据集（*NLCD*）。*CCD* 包含三个不同代码相关场景中的**467.5K** 个样本，即来自 Stack Overflow 的问答（**115K**）、代码到文本生成 [28]（**126K**）以及文本到代码生成 [26, 30]（**226.5K**）。*NLCD*包含 **25K** 个样本，其构建方式是使用ChatGPT 对维基百科中的内容进行润色 [47]。请注意，*CCD* 和*NLCD* 中的每个样本都是一个数据对，包含人类生成的数据和 ChatGPT 生成的数据。

基于该数据集，我们设计了全面的实验，评估现有检测工具的能力，包括三种开源检测器（GPT-2 检测器 [1], 、RoBERTa-QA [24] 和 DetectGPT [42]）和三种商业检测器（GPTZero [6], 、Writer [10] 和 Text Classifier [11]）。我们评估了所选工具检测 Chat-GPT 生成的程序内容的性能，并将其与检测人类生成内容的性能进行了比较。此外，我们还开展了一项人工研究，以了解人类的检测能力，并将其与现有的 AIGC 检测器进行比较。

大量实验表明，与自然语言数据相比，当前的AIGC 检测器难以检测代码数据。尽管微调能够提升性能，但其泛化能力仍然有限。一项人工研究还表明，人类也会遇到类似的困难，尤其是在处理代码数据时，由于代码的复杂性，这可能就像盲目猜测一样。总的来说，本文的主要贡献总结如下：

• 我们开展了一项全面的实证研究，评估了六种AIGC 检测器的性能，包括三种开源检测器和三种商业检测器，用于检测由 ChatGPT 生成的代码相关内容。据我们所知，这是首项专门评估不同 AIGC 检测器对 ChatGPT 生成的代码相关内容的检测性能的研究。

• 我们构建了两个大规模数据集，即 *CCD* 和*NLCD*，包含 **467.5K** 个代码相关样本和**25K** 个自然语言相关样本。

• 我们开展人工研究，以研究检测 ChatGPT 生成内容的难度，并将其与 AIGC 检测器的性能进行比较。

2 背景

2.1 ChatGPT

ChatGPT 是由Ope-nAI开发的前沿生成式人工智能模型，基于 GPT-3.5 架构，而该架构是最新的迭代版本在 GPT系列模型中，继 GPT [49], GPT-2 [50] 和 GPT-3 [15]之后。ChatGPT 主要使用大量未-标注文本数据进行训练。为了增强其在对话中生成更自然、更类似人类的回答的能力，ChatGPT 通过结合 InstructGPT数据集和人类对话进行监督微调。在这一过程中，人类对话被转换为对话格式，并与 InstructGPT 数据集结合使用，以进一步训练模型。ChatGPT利用基于人类反馈的强化学习（RLHF）算法进一步优化其回答，使在通用语料库文本数据上训练的模型与复杂的人类价值观保持一致。互联网训练数据中截至 2021 年所学到的海量知识与精心设计的微调过程相结合，使 ChatGPT能够在多个领域生成高质量的答-案。其强大的对话能力引发了人们对通用人工智能的兴趣，并且有望提高许多行业的生产力。例-如，它可以作为写作助手，帮助完成摘要、改写和翻译等任务；也可以作为聊天机器人，就各种主题进行对话并回答问题。

然而，与任何人工智能模型一样，人们也对其可靠性 [14], 伦理 [61] 和鲁棒性 [56]有所担忧。近期研究探讨了 ChatGPT生成的内容可能引发的问题，例如跨领域性能、生成偏见或敏感内容时涉及的伦理考量，以及抵御对抗攻击的鲁棒性。这些问题很大程度上源于 ChatGPT 依赖大量训练数据。鉴于 ChatGPT生成的内容可能存在相关问题，在依赖此类内容时应保持谨慎，尤其是在敏感或专业领域。为降低这些风险，检测内容是否由 ChatGPT 或其他人工智能模型生成至关重要，这有助于提升透明度和问责制。如此一来，用户便能对所呈现信息的可信度和可靠性做出明智判断，并减少造成伤害或传播错误信息的可能性。

2.2 AIGC 检测

为确保负责任且合乎伦理地使用人工智能生成内容，人们开发了各种检测器，用于识别给定内容是否由人工智能模型生成。截至 2023 年 3 月，我们收集了多个检测器 [1, 3–7, 9–11, 24, 42] ，其详细信息见 **??**。“支持的模型”列列出了检测器支持的模型类型，其中“未知”表示官方文档中未明确说明支持的模型。“接口”列表示可用于访问检测工具的接口类型。例如，“网站”表示只能通过检测器的官方网站访问，“API”表示支持通过标准编程接口访问。可以看出，大多数商业检测器都支持检测 GPT3 和 ChatGPT，而开源检测器仅支持一种类型的检测（即 GPT2 或 GPT3）。不过，值得注意的是，一些支持 API 访问的商业检测器可能并非免费，或者每天仅允许有限次数的访问。例如，Sapling [7]每月收费 25 美元提供 API 访问，而 Writefull [9] 则会在达到每日配额后限制访问。最后，

表 1：现有人工智能生成内容检测器的详细信息。

	检测器	支持的模型 GPT-2	GPT-3	ChatGPT	未知	接口 网站	API	输入长度
开源	GPT-2检测器 [1]	✓					✓	≤ 512词元
	DetectGPT [42]	✓				✓	✓	≤ 512词元
	RoBERTa-QA [24]			✓		✓	✓	≤ 512词元
商业	Contentatscale [4]		✓	✓		✓		25 个词至 2.5 万个字符
	Copyleaks [5]		✓	✓		✓	✓	150 个字符至 2.5 万个字符
	GPTZero [6]	✓	✓	✓		✓	✓	250 个字符至 2 万个字符
	Sapling [7]		✓	✓		✓	✓	50 个单词至 2 万个字符
	Writefull [9]		✓	✓		✓	✓	50词至2000词
	Writer [10]				✓	✓	✓	≤1.5k 个字符
	Compilatio [3]		✓	✓		✓		200 个字符至 2k 个字符
	AITextClassifier [11]			✓		✓		≥1k 个字符

如“输入长度”列所示，各检测器对输入文本长度的要求可能有所不同。例如，开源检测器最多只能处理 512 个词元的输入文本。商业检测器接受的输入文本长度范围从 25 个单词到 25K 个字符不等。

3 研究设计

本节将详细介绍我们的研究设计。我们的研究围绕 ChatGPT 如何在典型场景中被用于支- 持软件开发活动展开。在每个使用场景中，我们分别从ChatGPT 和人类专家那里收集数据，然后比-较不同检测器的性能。接下来，我们将介绍每个场景的设置、进行比较的检测器，以及待研究的研究问题。

3.1 场景和数据收集

ChatGPT 已广泛应用于软件开发活动中。例如，它可用于回答与编程相关的问题、使用自然语言总结代码片段，以及根据自然语言描述生成代码。在本研究中，我们聚焦于软件开发中最常见的三个场景：（1）编程主题问答；（2）代码到文本生成；（3）文本到代码生成。为开展研究，我们首先从人类和 ChatGPT 两方面收集了相关数据。随后，我们评估了不同检测器在这些场景中检测 ChatGPT 生成内容的能力。为了更好地了解检测器在代码相关内容上的性能，我们还收集了一个新的参考数据集，其中包含纯自然语言数据（来自自然语言润色场景），以供比较。接下来，我们将介绍这些场景和数据收集过程。

3.1.1 问答。 程序员遇到编程任务相关问题时，通常会在互联网上搜索答案。Stack Overflow 等问答网站正是为此而设计的。Stack Overflow 收集并整理相关答案，已成为当今软件开发者不可或缺的资源。Stack Overflow 希望真正的专家提供高质量答案，以建立一个健康且可持续发展的社区。出于对答案质量的担忧，Stack Overflow 禁止发布由 ChatGPT 生成的答案。然而，要执行这一规定，就必须有效且准确地检测人工智能生成的内容。

此类策略。因此，我们的第一个场景侧重于研究 AIGC 检测器识别 ChatGPT 生成的编程相关答案的有效性。

数据收集。 为了评估 AIGC 检测器在问答场景中的有效性，我们使用了来自 StackExchange [8]的Stack Overflow 数据集，其中包含 2021 年 9 月至 2022 年11月期间发布在 St ack Overflow 平台上的问题和答案。对于每个问题，我们将得票最高的回答视为人类生成的答案，并使用 ChatGPT 针对同一问题生成另一个答案。总计，我们获得了针对这 **115K**个问题的 **115K** 对人类生成答案和ChatGPT生成的答案。该数据集（记作 Q&A-GPT）为评估检测器在编程相关问答场景中识别 ChatGPT生成的内容的性能提供了全面的基准。

3.1.2 代码到文本生成。 为给定代码片段生成自然语言描述一直是学术界广泛研究的长期挑战 [12, 29, 37]。准确的代码描述可以帮助程序员更好地理解其功能，并提高软件开发效率。然而，撰写准确的代码注释是一项耗时且费力的任务。ChatGPT 在生成代码的自然语言描述方面展现出了出色的能力，使其成为自动化此任务的有力方案。然而，由于缺乏质量保证，目前仍不建议使用机器生成的代码摘要替代人类撰写的代码注释。因此，有必要使用能够识别 ChatGPT 生成的代码摘要的检测器，以发现机器生成代码注释的大规模使用。

数据收集。具体而言，我们采用了广泛使用的基准 CodeSearchNet [28],，其中每个样本都是一对（代码，描述），涵盖六种编程语言：Ruby、JavaScript、Go、Python、Java 和 PHP。每个样本都包含一个“docstring”字段，其中记录了人类专家对代码的描述。为了从 ChatGPT 获取代码描述，我们设计了一个提示词，要求 ChatGPT 总结给定的代码。提示词如下：

> 你将获得一个 <*LANG*> 函数代码，你的任务是详细总结其行为和功能。你的总结应清楚地

表2：收集数据的统计信息

划分	Wiki-GPT	Q&A-GPT	Code2Doc-GPT	Doc2Code-GPT	CONCODE-GPT	APPS-GPT
训练	0	100K	100K	100K	70K	0
Test	25K	15K	26K	26K	23K	7.5K
总计	25K	115K	126K	126K	93K	7.5K

> 解释该函数的作用、工作方式，以及它所需的输入参数和输出值。你应使用清晰简洁的语言进行说明。代码： *<CODE>*

其中， <LANG>是六种编程语言之一，<CODE> 是我们希望生成摘要的目标代码。我们总共从 CodeSearchNet 中选取了**126K** 个样本，并使用 ChatGPT 生成答案，将其记为 Code2Doc-GPT。每种编程语言对应的样本数量各不相同。Ruby、JavaScript、Go、Python、Java 和 PHP 的样本数量分别为 3,457、7,974、22,897、34,639、22,843 和 34,300。该数据集为评估检测器在多种编程语言中识别由 ChatGPT 生成的代码自然语言描述的性能，提供了全面的基准。

3.1.3 文本到代码生成。 随着人工智能技术的发展，尤其是大型语言模型的进步，人们对根据自然语言描述自动生成代码的兴趣日益高涨。近期研究 [16, 45] 表明，GPT-3 系列擅长遵循人类指令编写程序。例如，GPT-3.5 在 Python 编码任务上的准确率达到 48.1% [17],，而 OpenAI 发布的最新版本 GPT-4 [45] 在同一数据集上的准确率甚至达到 67.0%。在某些情境下，例如出于学术诚信政策的要求，可能禁止使用人工智能生成的代码。我们的第三种场景研究现有检测器的有效性，以更好地了解区分 ChatGPT 生成的代码与人类编写的代码在技术上是否可行。

数据收集。 具体而言，我们基于三个不同的代码生成数据集收集了ChatGPT生成的代码，即APPS [26],CONCODE [30], 以及第3.1.2节所述的Code2Doc-GPT数据集。

• APPS数据集 [26] 是一个Python数据集，由从各种公共网站收集的编程问题组成。数据集中的每个问题都附带其描述、真实解答以及用于验证所实现解答的测试用例。我们将真实解答视为人类专家提供的答案。为了从ChatGPT获取解答，我们创建了如图1a所示的提示词，其中“{question}”、“{test_case}”和“{starter}”分别是APPS中提供的问题描述、测试用例和函数名称的占位符。最终，我们从ChatGPT收集了**7.5K**个样本，记为APPS-GPT数据集。

• CONCODE [30] 是收录于 theCodeXGLUE [40] 集合中的 Java 数据集。其目标是根据自然语言描述和类环境提供的程序上下文，为 Java 类生成成员函数；类环境包括成员变量和其他成员函数。类别。为了从ChatGPT获取答案，我们创建了一个提示词，如图1b所示，其中“{desc}”和“{class}”是描述和类别环境的占-位符。CONCODE中提供的真实代码被视为来自人类专家的答案。我们总共收集了**93K** 样本，将其称为CONCODE-GPT数据集。

• 第3.1.2节构建的Code2Doc-GPT数据集包含由ChatGPT生成的给定代码的详细描述。我们可以自然地要求ChatGPT基于Code2Doc-GPT数据集中由ChatGPT生成的代码描述来生成代码。为此，我们设计了一个用于生成代码的提示词，如下所示：

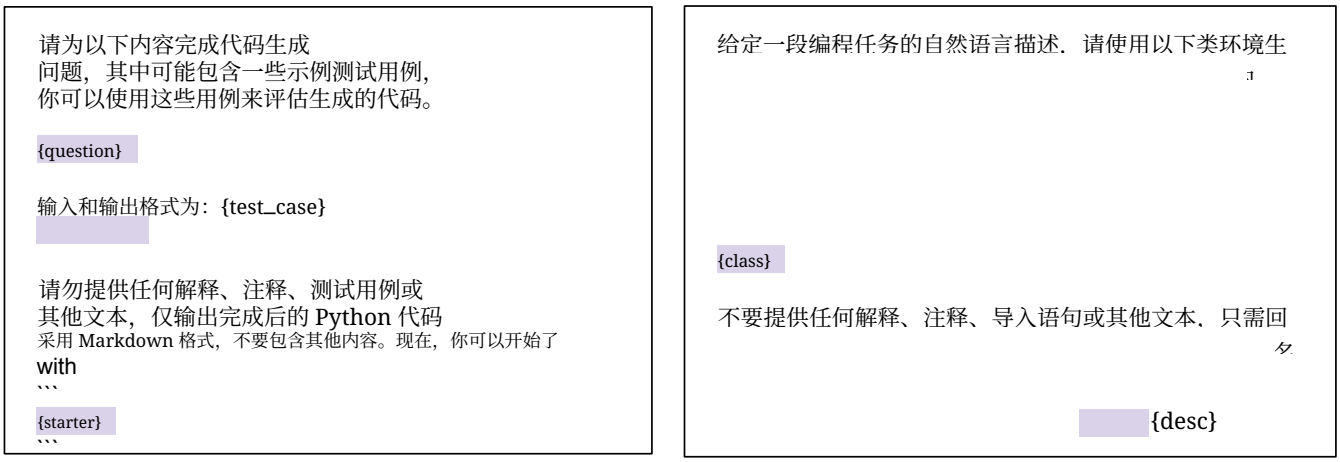
> 我们将为你提供一个*<LANG>*函数的详细描述，你的任务是生成一个实现程序行为的 *<LANG>*函数，具体实现应基于该描述。你应根据描述尽可能准确地编写函数代码，不提供任何额外的解释或假设。你的实现应符合*<LANG>* 语法和编码规范的标准。，

其中， <LANG> 表示以下编程语言之一：Ruby、JavaScript、Go、Python、Java 或PHP。Code2Doc-GPT数据集中的原始代码被视为来自人类专家的数据。此外，我们收集了总计**126k**个由ChatGPT生成的代码样本，并将其称为Doc2Code-GPT数据集。

在代码生成场景中，我们共生成了 **226.5K**对代码样本，每对样本都包含人类生成的代码和ChatGPT生成的代码。

3.1.4 自然语言润色。为了了解所研究的检测器在代码相关内容和自然语言内容上的表现可能有何不同，我们还创建了一个仅由 ChatGPT 生成的自然语言文本组成的数据集。该数据集可作为其他代码相关任务的参考。润色任务是要求 ChatGPT 用自己的措辞改写给定的文本段落。

数据收集。我们利用ChatGPT 对维基百科数据集 [47]中的文本进行润色。为此，我们设计了如下格式的提示词：“请润色以下内容： *<TEXT>*”，其中<TEXT>是来自维基百科数据集的原始文本的占位符。原始文本由人类专家提供，而润色版本则是由 ChatGPT 生成的对应文本。我们总共收集了**25K** 样本，并将其称为 Wiki-GPT 数据集。总而言之，我们总共收集了 **492.5K** 样本。*CD* 的统计数据见表2。由于我们的研究问题之一是探索微调对所收集数据集的帮助程度，因此我们需要将收集的数据划分为训练集和测试集。具体而言，我们从



(a) APPS 提示模板 **(b) CONCODE 提示模板**

图1：为 APPS 和 CONCODE 数据集设计的提示词。

使用 Q&A-GPT、Code2Doc-GPT 的数据集构建 *NLCD-Train*，使用 Doc2Code-GPT 的 100K 个样本和 CONCODE-GPT 的 70K 个样本构建 *CCD-Train*。其余数据用于测试集，即 *NLCD-Test* 和 *CCD-Test*。

数据预处理。通过仔细检查生成的 ChatGPT 答案，我们发现其中一些包含可识别的符号或短语，表明它们是由 ChatGPT 生成的。为提高我们整理的数据集的质量，我们删除了所有可能泄露真实情况的内容。具体来说，如果生成内容中出现“As an 人工智能语言 model...”或“As a language model...”之类的句子，我们就将其删除。对于代码片段，我们只保留代码块，并删除其周围任何多余的内容。

3.2 所选检测器

本节介绍本研究中将进行比较的人工智能生成内容检测器。这些检测器分为两类：商业检测器和开源检测器。我们选择以下三种商业检测器，因为可以对这些检测器进行大规模测试。其他一些商业检测器可能不是免费的，或者每天只允许有限次数的访问。此外，据我们所知，目前有三种开源检测器，因此我们将其用于比较。

3.2.1 商业检测器。

• GPTZero [6]。根据官方网站，它是一种用于预测内容是否由语言模型生成的分类模型，并提供句子、段落和文档级别的预测。具体而言，它是在大量人工撰写文本和AI生成文本语料库上训练的。人工撰写文本来自学生撰写的文章、新闻文章，以及涵盖科学和人文学科多个领域的问答数据集。对于每篇人工撰写文本，AI生成文本均由人工智能工具生成，以确保训练数据集保持平衡。该模型在一组此前从未见过的人类和人工智能文章以99%的准确率错误地区分人类撰写的文章，并以85%的准确率正确地区分人工智能生成的文章。

• Writer [10]。这是一款免费的人工智能内容检测工具。官方网站尚未公布更多技术细节。不过，网上有一项研究[3]，使用 10 篇人类撰写的内容和 10 篇人工智能撰写的内容来评估其有效性。研究结果显示，10 篇人类撰写的内容中有 4 篇被正确检测出来，而 10 篇人工智能撰写的内容中有 8 篇被成功检测出来。因此，根据所使用的 20 个样本来看，Writer 检测人工智能撰写内容的准确度似乎高于检测人类撰写内容。

• AITextClassifier [11]。这是 OpenAI 发布的官方检测器，是经过微调的 GPT 模型，用于预测文本由人工智能生成的概率。具体而言，该检测器的训练数据集包含 AI生成文本和人工撰写文本，其中人工撰写文本来自三个来源：维基百科数据集、WebText数据集，以及一组来自 InstrcutGPT 的人工示例 [46]；而 AI生成文本则与模型生成的人工撰写文本配对。随后，研究人员使用这个平衡数据集训练检测器。在验证集（其分布与训练集相同）和挑战集（由人工撰写的补全文本以及来自强大语言模型的补全文本组成）上的评估表明，该检测器的表现优于此前发布的分类器 GPT-2检测器 [1]。

3.2.2 开源检测器。

• GPT-2检测器 [1]。它是由 OpenAI 发布的 RoBERTa [39]微调检测器，其训练数据集来自拥有 15 亿参数的 GPT-2 模型的输出。根据其官方网站上的官方介绍，[4] 检测 15 亿参数 GPT-2 生成文本的准确率接近 95%。

[3]https://www.bloggersgoto.com/writer-com-ai-content-detector-review [4] https://openai.com/research/gpt-2-1-5b-release

• DetectGPT [42]。这是一个用于检测 GPT-2生成内容的开源工具。与其他检测器不同，其他检测器需要训练一个单独的分类器来进行检测，而该工具只需将候选段落在待检测模型下的对数概率，与另一个通用预训练语言模型（例如BART [31], T5 [51]）对该段落进行多次扰动后所得的平均对数概率进行比较。如果扰动后段落的平均概率比原始段落低出一定幅度，则该候选段落很可能是模型生成的。对分布内数据和分布外数据的评估表明，在分布内数据上，DetectGPT的表现可与基于学习的检测器 GPT-2检测器 [1] 相媲美，而在分布外数据上则优于后者。

• RoBERTa-QA [24]。Guo 等人提出在 GLTR Test-2 特征 [22], 上训练逻辑回归模型、RoBERTa-single 模型和 RoBERTa-QA 模型。这些模型在 Human ChatGPT Comparison Corpus（ HC3）数据集上进行训练。该数据集包含 3.7 万个问题，涵盖金融、医疗、法律、心理学和开放领域，用于检测由 ChatGPT 生成的内容。具体而言，逻辑回归模型利用 HC3 数据集中的词元特征进行分类。基于 RoBERTa 的检测器对预训练的 RoBERTa 模型进行微调，其中 RoBERTa-single 模型利用 ChatGPT 生成的文本进行微调，而 RoBERTa-QA 模型则利用 ChatGPT 生成的文本与输入问题的组合进行微调。评估结果证实，基于 RoBERTa 的检测器比回归模型具有更好的鲁棒性。此外，RoBERTa-QA 的检测效果优于 RoBERTa-single。本文选择 RoBERTa-QA 模型进行评估。

3.3 实验设计与研究问题

本节将介绍我们设计的研究问题，以及针对每个研究问题的实验设置。

3.3.1 研究问题*1*：现有检测器在检测 *ChatGPT* 生成的内容方面效果如何？我们将评估六种检测器在数据集*CCD-Test* 和*NLCD-Test*上的表现。

评估设置：具体而言，如果答案的输入长度超过所选检测器要求的最大序列长度，我们会对其进行截断以满足要求。某些检测器需要设置阈值，以便该检测器区分 ChatGPT 数据与人类数据。输出概率高于设定阈值，表示该内容由 ChatGPT 生成。我们遵循这些检测器的默认设置：GPTZero 的阈值设为 0.8，其余检测器均设为 0.5。

3.3.2 研究问题*2*：微调在多大程度上能够提升检测性能？由于我们比较的检测器主要是为检测自然语言内容而设计的，因此在我们的代码相关数据集上可能无法发挥最佳性能。因此，在这一研究问题中，我们旨在探究微调是否能够提升检测器的性能。

实验设置：我们选择了开源检测器 RoBERTa- QA [24] 进行微调。具体而言，我们微调了六个检测器。使用数据集 *NLCD-Train* 和*CCD-Train*。前三个检测器分别使用 Q&A-GPT、Code2Doc-GPT 以及 Q&A-GPT 和Code2Doc-GPT的组合所对应的训练数据集进行微调，这些数据集与检测自然语言数据的相关性更高；而另外三个检测器则分别在 CONCODE-GPT、Doc2Code-GPT 以及 CONCODE-GPT 和 Doc2Code-GPT 的组合上进行微调，这些数据集与代码数据的相关性更高。微调后的检测器在相同的测试数据集上进行评估，即*NLCD-Test* 和*CCD-Test*。

3.3.3 RQ3：当 *ChatGPT* 生成的数据经过轻微修改时，这些检测器的鲁棒性如何？在真实场景中，ChatGPT 生成的内容可能不会直接使用，部分生成内容也可能经过修改，以便进行定制或规避检测。因此，我们旨在研究检测器的鲁棒性，具体而言，探究经过不同变换修改后，人工智能生成内容是否仍能被检测出来。

实验设置：我们选取了三种变换操作来修改文本数据，并选取了三种变换操作来修改代码数据。我们将这些变换操作分别应用于人类生成的数据和 ChatGPT 生成的数据，并在变换后的数据集上评估检测器的性能。对于每个测试数据，我们均应用三种变换操作中的每一种（如适用），以生成变换后的数据。

针对基于代码的数据，三种变换如下：
• 函数名重命名。将函数名替换为一个新名称，该名称从基于 CodeSearchNet [28] 构建的函数名列表中随机选取。• 变量名重命名。在函数中随机选取一个变量，并将其名称替换为一个新名称，该名称从基于 CodeSearchNet 构建的变量名列表中随机选取。

• 代码语句插入。我们插入一条新的赋值语句，并更改变量名，以实现语句插入。新变量名从根据 CodeSearchNet 构建的变量名列表中随机选取。我们使用现有的 NLP 变换 [41] 来修改文本数据：

• 字符插入。从文本样本的某个单词中随机插入一个字符。• 字符删除。从文本样本的某个单词中随机删除一个字符。• 字符替换。将文本样本某个单词中的一个字符随机替换为另一个随机字符。

3.3.4 研究问题*4*：人类区分 *ChatGPT* 生成的内容的能力如何？为回答这一研究问题，我们对经验丰富的软件开发人员开展了在线调查，并观察他们区分机器生成内容的能力。

实验设置。 在本研究中，我们邀请了**50** 名至少拥有五年编程经验的资深开发者参加在线调查。截至2023年3月，其中**27** 人提供了有效答卷。我们随机将参与者分为两组，分别称为“示例组”和“无示例组”，两组分别收到14份和13份答卷。“示例组”的参与者会看到每项任务的一个示例，其中包含一对由人类和 ChatGPT 生成的内容，并明确标注了真实标签。“ 无示例组 ”的参与者则不会看到任何示例。

表 3：六种检测器的比较结果，其中粗体数字表示相应列中的最佳性能。

检测器	NLCD-Test									CCD-Test								
	Wiki-GPT			Q&A-GPT			Code2Doc-GPT			CONCODE-GPT			Doc2Code-GPT			APPS-GPT		
	曲线下面积	假阳性率	假阴性率	曲线下面积	假阳性率	假阴性率	曲线下面积	假阳性率	假阴性率	曲线下面积	假阳性率	假阴性率	曲线下面积	假阳性率	假阴性率	曲线下面积	假阳性率	假阴性率
GPT-2检测器	0.42	0.89	**0.21**	0.31	0.94	0.40	0.65	0.81	0.53	**0.63**	0.71	**0.11**	0.41	0.92	**0.16**	0.57	0.74	0.25
DetectGPT	0.54	0.35	0.37	0.40	0.06	0.34	0.16	0.02	0.62	0.55	**0.00**	1.00	0.00	**0.00**	1.00	0.54	**0.00**	0.99
RoBERTa-QA	0.54	0.35	0.33	0.42	0.05	0.32	0.23	0.01	0.48	0.46	**0.00**	1.00	0.02	**0.00**	1.00	0.43	**0.00**	0.99
GPTZero	**0.63**	**0.03**	0.83	0.85	**0.02**	0.56	0.90	**0.00**	0.54	0.50	0.17	0.87	**0.76**	0.25	0.31	0.57	0.01	0.98
Writer	0.60	0.07	0.83	0.72	0.04	0.77	0.56	0.15	0.62	0.34	0.29	0.89	0.56	0.08	0.86	0.38	0.06	0.95
AITextClassifier	0.62	0.61	0.26	**0.95**	0.63	**0.01**	**1.00**	0.82	**0.00**	0.46	0.81	0.24	0.56	0.68	0.29	**0.61**	0.99	**0.00**
Avg	0.56	0.38	0.47	0.61	0.29	0.40	0.58	0.30	0.46	0.49	0.33	0.69	0.39	0.32	0.60	0.52	0.30	0.69

表 4：Doc2Code-GPT 生成的不同编程语言的比较结果。

检测器	Go			Java			JavaScript			PHP			Python			Ruby		
	曲线下面积	假阳性率	假阴性率	曲线下面积	假阳性率	假阴性率	曲线下面积	假阳性率	假阴性率	曲线下面积	假阳性率	假阴性率	曲线下面积	假阳性率	假阴性率	曲线下面积	假阳性率	假阴性率
GPT-2检测器	0.38	0.96	0.10	0.40	0.92	0.18	0.38	0.89	0.23	0.39	0.90	0.22	0.46	0.91	0.12	0.46	0.89	**0.10**
DetectGPT	0.00	**0.00**	1.00	0.00	**0.00**	1.00	0.00	**0.00**	1.00	0.00	**0.00**	1.00	0.00	**0.00**	1.00	0.00	**0.00**	1.00
RoBERTa-QA	0.01	**0.00**	1.00	0.02	**0.00**	1.00	0.01	**0.00**	1.00	0.02	**0.00**	1.00	0.00	**0.00**	1.00	0.05	**0.00**	1.00
GPTZero	0.59	0.02	1.00	**0.90**	0.13	**0.09**	**0.90**	0.09	**0.15**	0.55	0.68	**0.21**	**0.92**	0.08	**0.11**	**0.75**	0.06	0.59
Writer	0.52	0.03	0.96	0.60	0.08	0.84	0.63	0.11	0.79	**0.61**	0.10	0.80	0.51	0.09	0.89	0.52	0.11	0.91
AITextClassifier	**0.95**	0.60	**0.01**	0.47	0.51	0.52	0.37	0.87	0.26	0.52	0.78	**0.21**	0.43	0.74	0.38	0.44	0.35	0.74
Avg	0.41	0.27	0.68	0.40	0.27	0.60	0.38	0.33	0.57	0.35	0.41	0.57	0.39	0.30	0.58	0.37	0.24	0.72

问卷包含50道题，涵盖5类与代码相关的数据（即 Q&A-GPT、Code2Doc-GPT、APPS-GPT、CONCODE-GPT 和 Doc2Code-GPT），每类10道题。每道题都会向参与者展示一段自然语言文本或一段代码，他们可以从三个选项中选择：“人类”“ChatGPT”和“无法判断”。每道题的作答时间最多为100秒。问卷最后还要求参与者指出影响其判断的最重要因素。

3.3.5 评估指标。为评估我们的检测器的性能，我们采用其曲线下面积（AUC）分数作为评估指标；AITextClassifier [11]等现有研究也普遍使用这一指标。我们还将假阳性率（FPR）和假阴性率（FNR）纳入评估指标。

曲线下面积（AUC）分数。检测器的AUC分数可解释为模型准确分类类别的能力所对应的概率，取值范围为0到1，其中1表示最佳，0.5则与随机猜测效果相当。例如，AUC分数为0.5意味着模型在为样本分配概率时，其表现仅与随机猜测相当。分类器的AUC分数越高，其区分正类和负类的能力就越强。我们将ChatGPT生成的数据视为正类。

假阳性率（FPR）。它指假阳性率，计算方式为 $FPR = \frac{FP}{FP+TN}$，其中 FP 是假阳性的数量（即被错误分类为ChatGPT生成的样本），TN 是真负例的数量（即被正确分类为人类专家撰写的样本），$N = FP + TN$ 是真实负例的总数（即标记为人类专家撰写的样本）。

假阴性率。它指的是假阴性率，计算公式为 $FNR = \frac{FP}{FN+TP}$ 其中 FN 是假阴性（即被错误分类为人类专家生成的样本）的数量，TP 是真阳性（即被正确分类为由 ChatGPT 生成的样本）的数量，$P = FN + TP$ 是真实正例总数（即标记为由 ChatGPT 生成的样本数）。

较低的假阳性率表示模型将人类创作内容误标为机器生成内容的可能性更低，而较低的假阴性率则表示相反。

4 研究结果

在本节中，我们将展示实验结果并进行分析，以尝试回答每个研究问题。

4.1 研究问题1：现有检测器的有效性

表3展示了六种检测器在不同数据集上的性能。第一列列出参与比较的检测器，第二列展示其在*NLCD-Test*上的结果，第三列展示其在*CCD-Test*上的结果。AVG 行显示了六种检测器在特定测试集上各项指标的平均值。

总体而言，结果表明，检测 ChatGPT生成的内容颇具挑战性：在*NLCD- Test* 上，平均曲线下面积、假阳性率和假阴性率分别为 0 46、0 58、0 32、0 44、0 32、066、. . 和 .；在 *CCD-Test* 上，分别为 . . 和 .。总体结果表明，检测 ChatGPT 生成的内容仍然非常困难。具体而言，*NLCD-Test* 上的曲线下面积高于*CCD-Test*，而*NLCD-Test* 上的假阴性率低于*CCD-Test*。这些结果表明，检测 ChatGPT生成的代码比检测自然语言内容更加困难，这可能是因为现有检测器使用的自然语言数据多于代码数据进行训练。

通过分析现有检测器的假阳性率（FPR）和假阴性率（FNR）结果，我们发现它们往往会偏向预测正例或负例。具体而言，一些检测器的假阳性率较高、假阴性率较低，表明它们倾向于预测结果为正例（即由 ChatGPT 生成）；另一些检测器的假阴性率较高、假阳性率较低，则表明它们倾向于预测结果为负例（即由人类生成）。例如，

表5：基于 RoBERTa-问答的微调模型在不同*NLCD-Train* 数据集上的结果。

NLCD-Train	NLCD-Test								
	Wiki-GPT			Q&A-GPT			Code2Doc-GPT		
	曲线下面积	假阳性率	假阴性率	曲线下面积	假阳性率	假阴性率	曲线下面积	假阳性率	假阴性率
RoBERTa-QA	0.54	**0.35**	0.33	0.42	0.05	0.32	0.23	0.01	0.48
Q&A-GPT	0.76	0.53	0.16	**0.99**	0.02	0.07	**1.00**	**0.00**	0.03
Code2Doc-GPT	0.66	0.51	0.26	0.93	**0.01**	0.54	**1.00**	**0.00**	**0.00**
Composite-NL	**0.78**	0.63	**0.10**	**0.99**	0.06	**0.02**	**1.00**	**0.00**	**0.00**

RoBERTa-QA 的假阳性率和假阴性率分别为 0 和 1，这表明所有样本都被预测为负例（即由人类生成）。我们推测，这种偏差可能是由这些检测器所用训练数据的差异，以及用于区分正例和负例的阈值设置造成的。例如，尽管 GPT2-Detector 和 RoBERTa-QA 都使用 RoBERTa 训练分类器，但 GPT2-Detector 使用 GPT-2 的输出进行训练，而 RoBERTa-QA 使用 ChatGPT 的输出。此外，检测器采用的阈值可能并不适用于我们的数据集。例如，AITextClassifier 在 Code2Doc-GPT 上的曲线下面积为 1，但其假阳性率为 0.82。我们采用了这些检测器的默认阈值，但这可能并非最适合我们数据集的阈值。因此，针对不同测试集选择合适的阈值，也可能是 AIGC 检测中的另一项挑战。

总体而言，商业 AIGC 检测器在检测 ChatGPT 生成的内容方面往往优于开源检测器。具体来说，我们观察到，商业检测器在*NLCD-Test*上的曲线下面积得分更高。在评估的检测器中，AITextClassifier 在*NLCD-Test*上的表现更好。此外，我们还观察到，三种商业检测器在 Wiki-GPT 数据集上的曲线下面积得分低于在 Q&A-GPT 和 Code2Doc-GPT 数据集上的得分，这可能是因为 ChatGPT 生成的 Wiki-GPT 数据与原始维基百科内容的文本相似度较高，这是由于生成的 Wiki-GPT 数据是基于维基百科内容润色而成的。

在*CCD-Test*中，我们观察到，所有受测 AIGC 检测器的表现都未达到令人满意的水平。GPTZero 和 AITextClassifier 在 Doc2Code-GPT 和 APPS-GPT 上表现出不错的结果（例如，曲线下面积得分分别为 0.76 和 0.61），但效果仍不理想。在三个与代码相关的数据集中，我们发现三种商业检测器在 CONCODE-GPT 上的表现都更差。进一步分析发现，CONCODE-GPT 中函数的规模相当小，因此 ChatGPT 生成的数据与人类编写的数据非常相似。

我们进一步研究了检测器在不同编程语言代码上的表现。表 4 展示了在 Doc2Code-GPT数据集（包含 6 种编程语言）上的比较检测结果。分析发现，检测器在不同编程语言上的表现水平各不相同。例如，GPTZero 在检测 Java、JavaScript、Python 和 Ruby 代码时表现出色，曲线下面积得分分别为 0.90、0.90、0.92 和 0.75，而在 Go 和 PHP 上表现不佳（仅为 0.59 和 0.55）。相反，AI-TextClassifier 在检测 Go 语言代码时表现优异（曲线下面积得分为 0.95），但在其余语言上的表现较差。值得注意的是，虽然 GPTZero 在 Doc2Code-GPT数据集中的 Java 代码上表现良好，但在.....上表现不佳。

表 6：基于 RoBERTa- QA 的微调模型在不同*CCD-Train* 数据集上的结果。

CCD-Train	CCD-Test								
	CONCODE-GPT			Doc2Code-GPT			APPS-GPT		
	曲线下面积	假阳性率	假阴性率	曲线下面积	假阳性率	假阴性率	曲线下面积	假阳性率	假阴性率
RoBERTa-QA	0.46	**0.00**	1.00	0.02	**0.00**	1.00	0.43	**0.00**	0.99
CONCODE-GPT	0.95	0.11	0.12	0.82	0.65	**0.02**	0.73	0.68	**0.06**
Doc2Code-GPT	0.90	0.37	0.10	0.93	0.39	0.04	0.76	0.61	0.08
Composite-Code	**0.98**	0.12	**0.05**	**0.94**	0.35	0.04	**0.77**	0.60	0.08

CONCODE-GPT数据集（曲线下面积分数为0.50），该数据集同样使用Java编写。这表明检测器的性能对具有不同分布的数据集很敏感，即使它们使用相同的编程语言编写。

研究问题1的答案：现有的人工智能生成内容检测器通常在自然语言数据上的表现优于在代码数据上的表现，这表明检测 ChatGPT生成的代码是一项更具挑战性的任务。尽管商业检测器的表现优于开源检测器，但它们在检测 ChatGPT生成的代码时仍面临困难。

4.2 研究问题2：微调的性能

鉴于现有检测器主要关注自然语言内容，我们进一步研究在代码数据集上进行微调是否能够提升对代码数据的检测性能。我们使用不同的训练样本对 RoBERTa-QA 进行了微调。表5展示了使用来自 *NLCD-Train* 的不同训练数据集对 RoBERTa-QA 进行微调的结果。随后，我们在来自 *NLCD-Test* 的不同测试集上评估该模型。第二行展示了原始检测器的性能，作为与三个新微调检测器进行比较的参考。“Composite”行表示将两个训练集（即来自 *NLCD-Train* 的 Q&A-GPT 和 Code2Doc-GPT）合并后用于微调模型。类似地，表6展示了使用来自 *CCD-Train* 的训练数据集进行微调的结果。请注意，我们没有使用 APPS-GPT 训练集进行微调，因为其中的样本数量有限（总计 75k 个）。

结果表明，微调能够显著提升现有检测器在自然语言和代码数据上的性能。在相应的测试数据集上，微调模型的曲线下面积得分较高，介于 0.9 至 1.0 之间，假阳性率/假阴性率较低，大多低于 0.1；Wiki-GPT 和 APPS-GPT 除外。Wiki-GPT 和 APPS-GPT 上的结果较差，是因为没有使用 Wiki-GPT 和 APPS-GPT 数据集进行微调。总体结果凸显了微调对于提升检测器在不同领域中性能的重要性。

此外，我们观察到，在一个数据集上训练的微调模型对其他数据集也具有一定的泛化能力。例如，与原始检测器相比，在 Q&A-GPT 上训练的微调模型在 Code2Doc-GPT 上表现更好。该模型在 Code2Doc-GPT 上的曲线下面积从 0.23 提升至 1.00，而假阳性率和假阴性率分别降至 0.00 和 0.03。类似地，在 Doc2Code-GPT 上微调的模型，其曲线下面积在 CONCODE-GPT 上从 0.46 提升至 0.90，在 APPS-GPT 上则从 0.43 提升至 0.76。值得一提的是，尽管这些微调模型并未在 Wiki-GPT 和 APPS-GPT 上训练，但仍取得了显著提升。因此，我们推测，不同场景的人工智能生成内容中可能存在一些共同模式，而微调模型能够学习这些模式

表 7：微调模型的鲁棒性分析。

模型	变异集	AUC	FPR	FNR	C_{con}	H_{con}
Composite-NL	Wiki-GPT	0.68	0.33	0.39	0.39	0.99
	QA-GPT	0.98	0.02	0.15	0.73	**1.00**
	Code2Doc-GPT	**1.00**	**0.00**	**0.01**	0.98	**1.00**
Composite-Code	CONCODE-GPT	0.96	0.22	0.05	**0.99**	0.76
	Doc2Code-GPT	0.94	0.34	0.04	**0.99**	0.97
	APPS-GPT	0.76	0.64	0.07	0.98	0.86

中。最后，我们发现，通过组合训练数据（即表5中的 Composite-NL 和表6中的 Composite-Code），微调模型的性能还能进一步提升。

RQ2 的回答：对收集到的 ChatGPT 生成内容进行微调，可以显著提升检测器的检测性能。此外，在一个数据集上训练的微调模型对其他数据集也具有一定的泛化能力。

4.3 RQ3：鲁棒性分析

本节评估 RQ2 中介绍的微调模型的鲁棒性，尤其是在 ChatGPT 生成的数据经过修改时的表现。结果如表7所示，其中第一列列出了分别基于表5和表6中的复合数据集训练的两个微调模型。第二列列出了使用我们的变异算子进行变异的六个测试数据集。后续各列展示了曲线下面积、假阳性率和假阴性率。列 C_{con} 和 H_{con} 显示了变异前后能够被正确检测的样本比例，代表预测的一致性。

将表7的结果与表5和表6最后几行的结果进行比较，可以发现总体性能略有下降，但无论是在自然语言还是代码数据上，下降幅度都不显著。这可能是因为我们采用了较为保守的变异算子，只对内容的一小部分进行修改，因此对模型的影响可能不大。不过，有一个明显的例外：Wiki-GPT 的变异版本上的性能大幅下降，表明模型在该数据集上的鲁棒性较低。这可能是因为我们没有将 Wiki-GPT 数据纳入微调训练数据集。未来我们将尝试更多变异策略。

通过分析列 C_{con} 和 H_{con}中的结果，我们可以观察到，在自然语言数据中，这些模型在人工数据上仍然表现良好，但在 ChatGPT 生成的数据中会出现一些错误，因而。
之前能够被正确预测的内容现在却被漏检了。相反，在代码数据中，模型仍然能够正确检测修改后的代码，但之前能够被正确预测的一些人类编写的代码现在却无法被检测到。然而，考虑到模型的整体性能并没有大幅下降，也存在一些此前被错误预测的数据现在得到了正确预测的情况。

RQ3 的答案：微调模型在我们保守的变异算子下通常具有鲁棒性。然而，对于未纳入微调数据集的数据集，模型的鲁棒性可能会受到影响。

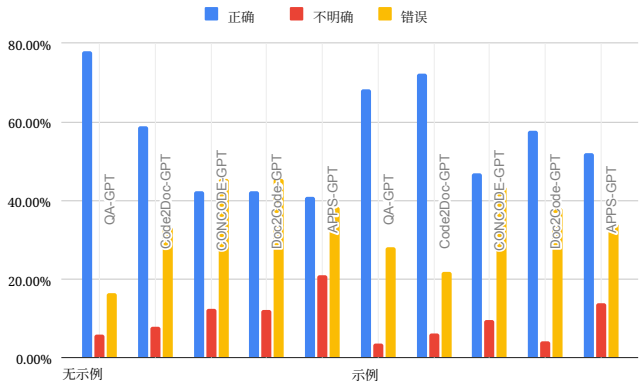


图2：调查中人类参与者的表现。

4.4 研究问题4：人工研究

我们在线调查收集的结果如图 2 所示。我们展示了“示例”组和“无示例”组在每种代码相关任务上的平均准确率。总体而言，在所有任务中，“示例”组的准确率略高，为 59.5%，而“无示例”组为 52.5%。两组在 Q&A-GPT 和 Code2Doc-GPT 任务中的表现都不错，准确率接近或超过 60%，最高达 77.9%。在这两项任务中，呈现给参与者的内容仅包含自然语言文本，不包含任何代码片段。表现较好的一个可能解释是，自然语言文本或许能透露更多线索，例如所用的语言模式、语气以及传达的情绪。调查受访者还将“语言使用——重复性／公式化的语言模式”评为最能让他们相信某段内容由人工智能生成的因素；其他因素还包括“连贯性与结构”“语气与风格”以及“情感吸引力”。对于仅涉及代码的任务，受访者的表现则不尽如人意。两组的平均答题正确率均为 47.1%，与随机猜测相近。

我们还比较了人类与同一组问题上表现最佳的一些检测器的性能。值得注意的是，我们还通过向ChatGPT 提出这些调查问题，测试了 ChatGPT 识别由其自身或人类生成的内容的性能。表8展示了比较结果的摘要。准确率表示调查问题中回答正确的比例。对于自然语言内容，我们的人类受试者与大多数检测器的表现相当（Comp-NL除外，该检测器针对同一数据集进行了特别微调）。对于代码片段，人类受试者表现不佳，落后于AITextClassifier（最优秀的商业检测器之一）。在这项实验中，ChatGPT 的表现并不理想。

研究问题4的答案：与现有检测器的研究结果类似，人类更擅长区分自然语言数据，而非代码数据。一些商业检测器在检测 ChatGPT生成的代码方面甚至比人类表现更好。

表8：人类与检测器在调查问题上的性能比较。

场景	AITextClassifier	ChatGPT	人类	Comp-Code	Comp-NL
Q&A-GPT	0.60	0.38	0.68	0.70	**1.00**
Code2Doc-GPT	0.50	0.42	**0.72**	0.50	1.00
CONCODE-GPT	**0.70**	0.50	0.47	**1.00**	0.30
Doc2Code-GPT	0.40	0.44	0.58	0.90	0.40
APPS-GPT	0.60	**0.64**	0.52	0.60	0.40

5 效度威胁

5.1 内部效度

首先，我们用于生成ChatGPT 内容的提示词可能会影响我们的结果。我们设计提示词时，力求模拟普通用户在相应使用场景下可能向 ChatGPT 提供的提示词。这些提示词不一定总是最具代表性。未来，我们计划研究不同提示词所产生的影响。此外，ChatGPT 生成的答案具有非确定性。即使使用相同的提示词，也可能生成不同的答案。但就构建用于评估 AIGC检测器的数据集而言，非确定性的影响是有限的。其次，我们无法始终核实数据集中由人类提供的数据来源。例如，我们认为 Stack Overflow 数据集中的答案由人类用户提供，但实际情况不一定总是如此。有些答案可能由其他工具生成。同样，将 APPS、CONCODE 和 Code2Doc 中的代码片段视为人类编写，也不一定可靠。不过，由于这些数据集中的数据大多是在大语言模型成为主流之前生成的，我们估计其影响是有限的。最后，我们研究的检测器需要设置检测阈值，而在不同设置下，所选的任何阈值都不一定始终能发挥最佳效果。我们承认这一潜在威胁，并采用了每个检测器推荐的数值，以减轻这一问题。

5.2 外部效度

首先，我们在所使用的数据集和所研究的检测器上获得的结果，未必能推广到其他数据和工具。为减轻这一威胁，我们从不同的软件开发场景和背景中收集数据，使其更能代表现实世界中的开发实践。我们选择了商业检测器和开源检测器，认为它们代表了当前最先进的水平。其次，人工研究中参与者的招募可能存在选择偏差，从而影响外部效度。由于我们的调查是在线进行的，我们无法完全控制受访者群体。为减轻这一问题，我们确保调查得到广泛传播，面向符合筛选标准的人群，即拥有至少五年编程经验的软件开发人员或研究人员。

6 相关工作

6.1 软件工程中的人工智能

在该领域的早期阶段，一些参数较少的小型神经网络被用于解决软件工程中的问题，例如源代码摘要 [12, 37],漏洞检测 [35, 60] 和代码搜索 [23, 38]。不同的使用了多种神经网络架构，例如 LSTM [27], Transformer [55],图神经网络 [34]。随着该领域的发展，人们采用了一些更先进的技术来提升性能，例如预训练。早期的代码预训练模型工作，例如 CodeBERT [20], GraphCodeBERT [25]，采用仅编码器 Transformer 作为架构，在代码相关数据上预训练通用模型，然后针对下游任务对该预训练模型进行微调，以取得更优的性能。后续工作进一步改进了模型，例如 PLBART [13] 和 CodeT5 [58]采用编码器-解码器 Transformer 作为模型架构，以提升模型容量。尽管与以往工作相比，这些预训练模型在不同的软件工程任务中取得了显著提升，但仍无法应用于真实场景。OpenAI 更进一步，发布了 CodeX 模型 [16],，该模型基于 GPT 模型，使用 GitHub 上公开的代码进行训练。此外，GitHub Copilot 由 CodeX 的一个独立生产版本提供支持。OpenAI 发布的 ChatGPT 是另一种具有代表性的代码工具，它基于 GPT-3.5 系列进行微调，并通过 RLHF 实现对齐。GPT-3.5 系列中蕴含的大量知识以及强大的对话能力，使 ChatGPT 能够在不同领域生成准确的回答。因此，它也可应用于软件工程领域，例如代码摘要与生成 [43], 以及漏洞检测与修复 [52, 54]。例如，Sobania 等人 [52] 使用 ChatGPT 修复了标准漏洞修复基准 QuixBugs [36] 中的错误，取得了优于当前最先进方法的表现，成功修复了 40 个错误中的 31 个。正是由于 ChatGPT 已广泛应用于软件工程，本文探讨 ChatGPT生成的代码能否被现有检测器检测出来。

6.2 深度伪造检测

深度伪造是指通过深度生成方法创建或操纵面部外观，而深度伪造检测旨在判断图像或视频是由人工智能合成，还是由摄像机自然拍摄的，这与 AIGC内容检测器类似。根据提取的特征，相关方法主要分为基于空间的、基于频率的和基于生物信号的。在现有研究中，基于空间域的深度伪造检测是最受欢迎的技术 [32, 57] ，其旨在观察空间域中各种可见或不可见的伪影，以区分真实与伪造。除了空间域之外，由于真实人脸与合成人脸之间的差异也会体现在频域中，因此也有一些研究 [21, 48] 利用频域差异进行检测。此外，与合成人脸相比，由摄像机拍摄的真实面部图像和视频具有自然性，因此可以利用生物信号进行区分 [19, 33]。与深度伪造检测的对象为图像或视频不同，我们旨在识别由 ChatGPT 合成的文本内容。

7 结论

本文首次开展实证研究，评估现有 AIGC 检测器在软件领域的表现。我们创建了一个全面的数据集，其中包含由 ChatGPT 生成的代码相关内容。研究结果表明，目前的

AIGC 检测器在处理代码相关数据时的表现不如处理自然语言数据。虽然微调可以提升性能，但模型的泛化能力仍然面临挑战。此外，我们开展了一项人工研究，以了解人类识别ChatGPT生成的内容的能力。研究结果表明，人类也面临类似的挑战。这些发现凸显了进一步研究这一领域的必要性，尤其是开发稳健且具有良好泛化能力的 AIGC 检测器。

参考文献

[1] 2019-11. OpenAI：GPT-2 检测器。https://github.com/openai/gpt-2-output-dataset/tree/master/detector [2] 2022-11. ChatGPT：针对对话优化语言模型。https://chat.openai.com [3] 2022-11. Compilatio：AI 检测器评估。https://ai-detector.compilatio.net [4] 2022-11. Contentatscale：AI 检测器。https://contentatscale.ai/ ai-content-detector [5] 2022-11. Copyleaks：AI 内容检测器。https://copyleaks.com/ ai-content-detector [6] 2022-11. GPTZero。https://gptzero.me [7] 2022-11. Sapling：AI 内容检测器。https://sapling.ai/ai-content-detector [8] 2022-11. Stack Exchange（2021-2022）：Stack Exchange 数据转储。Archive.org。数据集。https://ia800107.us.archive.org/27/items/stackexchange/ [9] 2022-11. Writefull：GPT 检测器。https://x.writefull.com/gpt-detector [10] 2022-11. Writer：AI 内容检测器。https://writer.com/ai-content-detector [11] 2023-01. AI 文本分类器。https://beta.openai.com/ai-textclassifier [12] Wasi Uddin Ahmad、SaikatChakraborty、Baishakhi Ray和 Kai-Wei Chang。2020年。用于源代码摘要的基于Transformer 的方法。arXiv 预印本 arXiv:2005.00653（2020）。 [13] Wasi Uddin Ahmad、SaikatChakraborty、Baishakhi Ray 和 Kai-WeiChang。2021 年。用于程序理解和生成的统一预训练。arXiv预印本 arXiv:2103.06333（2021）。[14] Yejin Bang、Samuel Cahyawijaya、Nayeon Lee、Wenliang Dai、Dan Su、Bryan Wilie、Holy Lovenia、Ziwei Ji、Tiezheng Yu、WillyChung 等。2023 年。对ChatGPT在推理、幻觉和交互性方面进行的多任务、多语言、多模态评估。arXiv预印本 arXiv:2302.04023（2023）。[15] Tom Brown、BenjaminMann、Nick Ryder、Melanie Subbiah、Jared D Kaplan、Prafulla Dhariwal、Arvind Neelakantan、Pranav Shyam、Girish Sastry、Amanda Askell等。2020年。语言模型是小样本学习器。神经信息处理系统进展 33（2020），1877–1901。 [16] Bei Chen、Fengji Zhang、Anh Nguyen、Daoguang Zan、Zeqi Lin、Jian-Guang Lou和 WeizhuChen。2022 年。Codet：使用生成的测试进行代码生成。arXiv预印本 arXiv:2207.10397（2022）。[17] Mark Chen、Jerry Tworek、Heewoo Jun、Qiming Yuan、Henrique Ponde de Oliveira Pinto、JaredKaplan、Harri Edwards、Yuri Burda、NicholasJoseph、Greg Brockman等。2021 年。评估在代码上训练的大型语言模型。arXiv预印本 arXiv:2107.03374（2021）。[18] Paul F Christiano、Jan Leike、Tom Brown、MiljanMartic、ShaneLegg和 Dario Amodei。2017 年。从人类偏好中进行深度强化学习。神经信息处理系统进展 30（2017）。 [19] Umur Aybars Ciftci、Ilke Demir 和 LijunYin。2020 年。Fakecatcher：使用生物信号检测合成人像视频。IEEE 模式分析与机器智能汇刊（2020）。[20] Zhangyin Feng、Daya Guo、Duyu Tang、Nan Duan、Xiaocheng Feng、Ming Gong、Linjun Shou、Bing Qin、Ting Liu、Daxin Jiang 等。2020年。CodeBERT：面向编程语言和自然语言的预训练模型。arXiv预印本 arXiv:2002.08155（2020）。[21] Joel Frank、Thorsten Eisenhofer、Lea Schönherr、Asja Fischer、Dorothea Kolossa和Thorsten Holz。2020 年。利用频率分析进行深度伪造图像识别。载于国际机器学习会议。PMLR，3247–3258。 [22] Sebastian Gehrmann、HendrikStrobelt 和 AlexanderM Rush。2019 年。GLTR：生成文本的统计检测与可视化。arXiv 预印本 arXiv:1906.04043（2019）。 [23] Xiaodong Gu、Hongyu Zhang和 Sunghun Kim。2018 年。深度代码搜索。载于第 40 届国际软件工程会议。933–944。[24] Biyang Guo、Xin Zhang、Ziyuan Wang、Minqi Jiang、Jinran Nie、Yuxuan Ding、Jianwei Yue 和YupengWu。2023年。ChatGPT 与人类专家有多接近？比较语料库、评估与检测。arXiv预印本 arXiv:2301.07597（2023）。[25] Daya Guo、Shuo Ren、Shuai Lu、Zhangyin Feng、Duyu Tang、Shujie Liu、Long Zhou、Nan Duan、Alexey Svyatkovskiy、Shengyu Fu 等。2020 年。GraphCodeBERT：利用数据流进行代码表示预训练。arXiv 预印本 arXiv:2009.08366（2020）。

[26] Dan Hendrycks、Steven Basart、Saurav Kadavath、Mantas Mazeika、Akul Arora、Ethan Guo、Collin Burns、Samir Puranik、HoraceHe、Dawn Song 和 Jacob Steinhardt。2021。使用APPS 衡量编程挑战能力。NeurIPS（2021）。[27] SeppHochreiter和JürgenSchmidhuber。1997。长短期记忆。神经计算 9，8（1997），1735–1780。[28] Hamel Husain、Ho-Hsiang Wu、Tiferet Gazit、Miltiadis Allamanis 和 Marc Brockschmidt。2019。CodeSearchNet 挑战：评估语义代码搜索的现状。arXiv预印本 arXiv:1909.09436（2019）。[29] Srinivasan Iyer、Ioannis Konstas、Alvin Cheung 和Luke Zettlemoyer。2016。使用神经注意力模型对源代码进行摘要。在第54 届计算语言学协会年会上（第 1 卷：长篇论文）的会议录中。2073–2083。[30] Srinivasan Iyer、Ioannis Konstas、Alvin Cheung和 Luke Zettle moyer。2018。在程序上下文中将语言映射为代码。arXiv 预印本 arXiv:1808.09588（2018）。[31] Mike Lewis、Yinhan Liu、Naman Goyal、Marjan Ghazvininejad、Abdelrahman Mohamed、Omer Levy、VesStoyanov 和 LukeZettlemoyer。2019。BART：用于自然语言生成、翻译和理解的去噪序列到序列预训练。arXiv 预印本 arXiv:1910.13461（2019）。[32] Haodong Li、Bin Li、ShunquanTan 和 JiwuHuang。2020。利用颜色分量差异识别深度网络生成的图像。信号处理 174（2020），107616。[33] YuezunLi和 Siwei Lyu。2018。通过检测面部变形伪影揭示深度伪造视频。arXiv预印本 arXiv:1811.00656（2018）。[34] Yujia Li、Daniel Tarlow、Marc Brockschmidt 和Richard Zemel。2015。门控图序列神经网络。arXiv预印本 arXiv:1511.05493（2015）。[35] Zhen Li、Deqing Zou、ShouhuaiXu、Xinyu Ou、Hai Jin、Sujuan Wang、Zhijun Deng 和Yuyi Zhong。2018。VulDeePecker：一种基于深度学习的漏洞检测系统。arXiv预印本 arXiv:1801.01681（2018）。[36] Derrick Lin、James Koppel、AngelaChen 和 Armando Solar-Lezama。2017。QuixBugs：基于 Quixey 挑战的多语言程序修复基准集。在 2017 年 ACMSIGPLAN 国际系统、编程、语言与应用：人类软件大会的会议附录中。55–56。[37] Shangqing Liu、Yu Chen、Xiaofei Xie、Jing KaiSiow 和Yang Liu。2021。通过混合 GNN 实现用于代码摘要的检索增强生成。在国际学习表征会议上。https://openreview.net/forum? id=zv-typ1gPxA[38] Shangqing Liu、Xiaofei Xie、Jingkai Siow、Lei Ma、Guozhu Meng 和YangLiu。2023。GraphSearchNet：通过捕获全局依赖关系增强GNN的语义代码搜索。IEEE 软件工程汇刊（2023）。[39] Yinhan Liu、Myle Ott、Naman Goyal、Jingfei Du、Mandar Joshi、Danqi Chen、Omer Levy、Mike Lewis、Luke Zettlemoyer和 Veselin Stoyanov。2019。RoBERTa：一种稳健优化的BERT 预训练方法。arXiv预印本 arXiv:1907.11692（2019）。[40] Shuai Lu、Daya Guo、Shuo Ren、Junjie Huang、Alexey Svyatkovskiy、Ambrosio Blanco、Colin B.Clement、Dawn Drain、Daxin Jiang、Duyu Tang、Ge Li、Lidong Zhou、Linjun Shou、Long Zhou、Michele Tufano、Ming Gong、Ming Zhou、Nan Duan、Neel Sundaresan、Shao Kun Deng、Shengyu Fu 和 Shujie Liu。2021。CodeXGLUE：用于代码理解和生成的机器学习基准数据集。CoRR abs/2102.04664（2021）。[41] Edward Ma。2019。自然语言处理增强。https://github.com/makcedward/nlpaug。[42] Eric Mitchell、Yoonho Lee、Alexander Khazatsky、Christopher D Manning 和 Chelsea Finn。2023。DetectGPT：使用概率曲率进行零样本机器生成文本检测。arXiv预印本 arXiv:2301.11305（2023）。[43] Madhav Nair、Rajat Sadhukhan 和 DebdeepMukhopadhyay。2023。使用ChatGPT生成抵御 CWE 的安全硬件。密码学电子打印档案（2023）。[44] OpenAI。2019。ChatGPT 演示。https://www.youtube.com/watch?v= outcGtbnMuQ&ab_channel=OpenAI。[45] OpenAI。2023。GPT-4 技术报告。arXiv:2303.08774 [cs.CL][46] Long Ouyang、Jeff Wu、Xu Jiang、Diogo Almeida、Carroll L Wainwright、Pamela Mishkin、Chong Zhang、Sandhini Agarwal、Katarina Slama、AlexRay等。2022。训练语言模型遵循具有人类反馈的指令。arXiv预印本 arXiv:2203.02155（2022）。[47] Fabio Petroni、AleksandraPiktus、AngelaFan、Patrick Lewis、Majid Yazdani、Nicola DeCao、James Thorne、Yacine Jernite、Vladimir Karpukhin、Jean Maillard、Vassilis Plachouras、Tim Rocktäschel和Sebastian Riedel。2021。KILT：知识密集型语言任务基准。在2021年北美计算语言学协会分会会议：人类语言技术的会议录中。计算语言学协会，线上，

2523 2544 10 18653 1 2021 200 – 。[48] 2020 https://doi.org/ 。/v / 。naacl-main。Yuyang Qian、Guojun Yin、LuSheng、ZixuanChen 和Jing Shao。。在频率域中思考：通过挖掘频率感知线索进行人脸伪造检测。在《计算机视觉–ECCV 2020：第 16届欧洲会议，英国格拉斯哥，2020 年 8 月23–28 日，会议录，第 XII 部分》中。Springer，86–103。

[49] Alec Radford、Karthik Narasimhan、Tim Salimans、Ilya Sutskever 等。2018。通过生成式预训练提升语言理解。（2018）。[50] Alec Radford、Jeffrey Wu、Rewon Child、David Luan、Dario Amodei、Ilya Sutskever 等。2019。语言模型是无监督的多任务学习者。*OpenAI* 博客1，8（2019），9。[51] Colin Raffel、Noam Shazeer、Adam Roberts、Katherine Lee、Sharan Narang、Michael Matena、Yanqi Zhou、Wei Li 和 Peter J Liu。2020。使用统一的文本到文本转换器探索迁移学习的极限。《机器学习研究期刊》21，1（2020），5485–5551。[52] Dominik Sobania、Martin Briesch、Carol Hanna 和 Justyna Petke。2023。分析 ChatGPT 的自动错误修复性能。*arXiv* 预印本 *arXiv:2301.08653*（2023）。[53] Nisan Stiennon、Long Ouyang、Jeffrey Wu、Daniel Ziegler、Ryan Lowe、Chelsea Voss、Alec Radford、Dario Amodei 和 Paul F Christiano。2020。通过人类反馈学习摘要。神经信息处理系统进展33（2020），3008–3021。[54] Nigar M Shafiq Surameery 和 Mohammed Y Shakor。2023。使用 Chat GPT 解决编程错误。《国际信息技术与计算机工程期刊》（*IJITC*）*ISSN*：*2455-5290*3，01（2023），17–22。[55] Ashish Vaswani、Noam Shazeer、Niki Parmar、Jakob Uszkoreit、Llion Jones、Aidan N Gomez、Łukasz Kaiser 和 Illia Polosukhin。2017。注意力机制就是你所需要的一切。神经信息处理系统进展30（2017）。[56] Jindong Wang、Xixu Hu、Wenxin Hou、Hao Chen、Runkai Zheng、Yidong Wang、Linyi Yang、Haojun Huang、Wei Ye、Xiubo Geng 等。2023。论 ChatGPT 的鲁棒性：从对抗和分布外的角度出发。*arXiv* 预印本 *arXiv:2302.12095*（2023）。[57] Sheng-Yu Wang、Oliver Wang、Richard Zhang、Andrew Owens 和 Alexei A Efros。2020。CNN生成的图像出乎意料地容易识别……至少目前如此。载于*IEEE/CVF*计算机视觉与模式识别会议论文集。8695–8704。[58] Yue Wang、Weishi Wang、Shafiq Joty 和 Steven CH Hoi。2021。Codet5：用于代码理解和生成的标识符感知统一预训练编码器—解码器模型。*arXiv* 预印本 *arXiv:2109.00859*（2021）。[59] Jeff Wu、Long Ouyang、Daniel M Ziegler、Nisan Stiennon、Ryan Lowe、Jan Leike 和 Paul Christiano。2021。通过人类反馈递归地总结书籍。*arXiv* 预印本 *arXiv:2109.10862*（2021）。[60] Yaqin Zhou、Shangqing Liu、Jingkai Siow、Xiaoning Du 和 Yang Liu。2019。Devign：通过图神经网络学习全面的程序语义，实现有效的漏洞识别。神经信息处理系统进展32（2019）。[61] Terry Yue Zhuo、Yujin Huang、Chunyang Chen 和 Zhenchang Xing。2023。探索 ChatGPT 的人工智能伦理：诊断分析。*arXiv* 预印本 *arXiv:2301.12867*（2023）。