生成式人工智能技术如何影响专业型工作?

——来自软件工程行业的早期证据

马晔风　陈　楠　崔雪彬*

内容提要　生成式人工智能（AI）技术的发展显著缩小了人工智能与人类认知能力的差距，这一趋势对知识型、专业型工作带来重大影响。本文以软件工程行业为研究对象，分析生成式 AI 在应用早期阶段对专业型工作的影响效果与机制。研究显示，现阶段生成式 AI 对专业型工作的冲击有限，对软件开发工作的影响以效率提升为主，距离大规模广泛替代还有较大差距。AI 工具在通用型业务场景的效率提升效应最高，在行业属性较强的应用场景中提升效应有限。当应用场景涉及云计算、机器学习等前沿技术能力时，生成式 AI 工具的表现最差。原因在于，使用生成式 AI 工具辅助软件开发仍面临训练数据、复杂逻辑、隐性知识、人机交互等制约因素，在理解问题、解决问题、生成代码等环节限制了技术在实际应用中的表现。从长期来看，生成式 AI 有望推动软件开发工作从单向任务自动化转变为“人机合作”模式，从而实现基于“双向互动”的效率提升。本文的研究结果对未来如何建立专业型劳动者与人工智能技术进步之间的良性关系具有启发意义。

关 键 词　生成式人工智能　专业型工作　软件开发　应用场景

一　引言

2022 年底，大型语言模型（Large Language Models，LLMs）及其代表性产品

* 马晔风，中国社会科学院数量经济与技术经济研究所、中国社会科学院经济大数据与政策评估实验室，电子邮箱：mayefeng@ cass. org. cn；陈楠，中国社会科学院数量经济与技术经济研究所、中国社会科学院经济大数据与政策评估实验室，电子邮箱：chennan@ cass. org. cn；崔雪彬，南京大学数字经济与管理学院，电子邮箱：cuixuebin@ nju. edu. cn。作者感谢国家社会科学基金青年项目（项目批准号：22CJY047）的资助。

ChatGPT 在全球快速传播，推动了生成式人工智能（AI）的高速发展。生成式 AI 技术是一种基于机器学习模型的技术，使计算机可以根据特定输入生成新的内容，例如文章、音乐、图像等。这项技术在理解用户语义并自动生成关联内容方面表现异常出色。既有研究显示，先前的自动化浪潮主要影响重复型、程式化的工作任务（Acemoglu & Restrepo，2019a；Acemoglu & Restrepo，2019b）。创造型、专业型工作任务（如写作、编程、医务工作）由于难以被编码，很大程度上避免了自动化。然而，生成式 AI 技术的最新进展有可能快速改变这一模式。与前几代人工智能深度学习模型相比，生成式 AI 模型显著缩小了人工智能与人类认知力的差距，而且模型的能力还在快速提升，并表现出比已知能力更强的学习能力（Korinek，2023）。目前在一些领域，生成式 AI 似乎已经实现了对人类隐性知识和认知能力的自动化（Brynjolfsson et al.，2023）。这一趋势将对知识型劳动者产生重大影响，越来越多的人呼吁围绕人工智能对知识型、专业型工作的影响进行更多的实证研究和理论探索，特别是在非常规认知任务领域（Frank et al.，2019）。

由于生成式 AI 技术仍处于商业应用的早期阶段，当前围绕生成式 AI 技术对劳动力市场影响的研究大多是前瞻性、预测性研究。例如，将生成式 AI 的技术特征与不同职业/工作所依赖的技能、任务、能力等指标相联系，评估不同工作/岗位可能受到的影响（Eloundou et al.，2023；Lou et al.，2023；Tolan et al.，2021），或者从定性层面讨论生成式 AI 技术对经济、社会、就业的广泛影响（陈永伟，2023；徐国庆等，2023；郑世林等，2023）。关于生成式 AI 技术影响劳动就业的回顾性经验证据非常有限，针对专业型工作的经验研究则更加少见。随着生成式 AI 技术商业化程度的加深，那些曾经受自动化浪潮影响较小的专业型工作将受到怎样的影响？生成式 AI 技术会给劳动力市场带来哪些新的变革？本文尝试为这些问题提供技术应用早期的经验证据。

本文的出发点是当前备受舆论关注的“生成式 AI 是否会替代程序员”的争论。从传统认知来看，软件工程师通常需要接受多年的专业技能培训，是具有高技能水平的知识型劳动者。伴随数字经济的快速发展，软件工程师的劳动力市场需求长期处于高位，职业前景和岗位安全性良好，也是人工智能技术发展的重要人才基础。同时，软件开发工作属于典型的非常规认知任务领域，其本质是实现工作任务的编码化和自动化。尽管编程代码也是以文字的形式表现，但其背后涉及复杂的逻辑、技术、业务等方面的知识，想要自动生成满足需求的代码比生成自然语言面临更大的难度。事实上，许多专业型工作都表现出与软件开发相似的特征，尽管大部分工作任务涉及文字或图像的生成（这些都是生成式 AI 技术擅长的内容），但是距离全部任务可编程和岗位完全自

动化还有较远的距离，生成式 AI 技术在真实场景中的影响值得深入探讨。

本文基于与某软件服务公司的合作，通过考察该公司不同业务部门在实际工作中使用生成式 AI 工具（Github Copilot，简称 Copilot）的情况，分析生成式 AI 技术在应用早期阶段对软件开发工作的实际影响，并进一步分析了造成这种影响的原因和机制。本文首先评估了 Copilot 在实际工作场景下解决基本编程和算法问题的能力，研究了 Copilot 对软件开发工作的代码提升效应和效率提升效应的整体情况。研究显示，现阶段生成式 AI 工具对软件开发工作的效率提升作用比较有限，软件开发工作远没有达到被生成式 AI 技术广泛替代的程度。本文进一步分析了 Copilot 在不同应用场景下对软件开发工作的辅助效果，从训练数据和任务属性等方面解释了生成式 AI 工具的局限性以及在不同应用场景下所表现出的异质性。研究发现，生成式 AI 工具在通用型业务场景中的效率提升效应最高，在行业属性较强的应用场景中提升效应有限，且当应用场景涉及云计算、机器学习等前沿技术能力时，生成式 AI 工具的表现最差。

目前，使用生成式 AI 工具辅助软件开发仍面临训练数据、复杂逻辑、隐性知识、人机交互等制约因素，影响着生成式 AI 工具在软件开发工作场景中的表现。具体而言，在理解需求阶段，软件工程师与 Copilot 的交互问答有助于提升 AI 工具的需求理解能力，而复杂逻辑和训练数据缺失则对 AI 工具的表现形成限制。在解决问题阶段，隐性知识和复杂逻辑是生成式 AI 工具需要克服的制约因素，软件开发人员通过问答交互能够在一定程度上弥补行业和机构知识的限制，而技术类隐性知识更难通过提示词或注释形式输入给 AI 工具。在生成代码阶段，生成式 AI 工具与软件开发工程师之间具有更强的交互属性，绝大多数的代码生成过程需要人工参与，以满足编程应用场景下对正确性和逻辑性的高要求（编写的代码有一点逻辑错误就会影响编译和运行）。尽管现阶段生成式 AI 技术对软件开发工作的影响有限，但它给软件工程行业的工作范式带来了新的变革。与传统的编程自动化工具有所不同，生成式 AI 工具正在将软件开发工作模式从单向任务自动化转变为团队合作模式，从而实现基于“双向互动”的效率提升。生成式 AI 工具不只是扮演一个工具的角色，而更像一个“合作伙伴”，这一变化有可能重塑 AI 工具与软件工程师之间的辅助关系。

本文的贡献主要体现在三个方面。第一，本文选取软件开发工作为研究对象，在关于生成式 AI 技术如何影响专业型工作的研究方面提供了来自中国的证据，丰富了相关文献。第二，在软件工程行业，已有研究多采用 LeetCode 等测试场景，检验生成式人工智能对软件工程岗位和任务的影响，受测试样本、任务复杂度、真实工作场景与测试场景差异等限制，相关研究结论值得进一步探讨。本文提供了生成式 AI 在软件工

程行业真实工作场景下的实证证据，能够为进一步丰富该领域相关学术研究、引导生成式 AI 的行业应用和实践提供参考。第三，本文的分析结果表明生成式 AI 在软件工程行业的影响还非常有限，尚未实现对软件开发人员代码编写或整体工作效率的显著提升。本文从应用场景类型（通用型、业务型、技术型）、训练数据积累、生成式 AI 工作范式等角度，对这一问题开展了更为深入的分析，对未来如何提升生成式 AI 工具在软件工程领域的性能和表现、如何更好建立从业人员与人工智能技术进步之间的良性关系提出了意见和建议。本文的结构安排如下：第二部分是文献综述；第三部分是研究设计，介绍了本文的研究数据与分析方法；第四部分分析了生成式 AI 工具对软件开发工作的影响与制约因素；第五部分讨论了生成式 AI 工具对软件开发工作的影响机制；第六部分是总结与建议。

二　文献综述

（一）生成式 AI 技术发展情况及特征

2010 年前后，以机器学习算法为核心的第三代人工智能技术快速发展，并在生产生活各领域实现了广泛应用。与早期基于符号系统和专家系统的“旧式人工智能”相比，新一代人工智能系统摒弃了由上至下、规则驱动的技术路线，转而以海量数据为驱动，通过机器学习算法识别数据之间的关联关系（Taddy，2019）。在发展初期阶段，新一代人工智能技术及其应用以判别式 AI 为主，即基于对已有数据的学习，建立输入与输出之间的关系，并将其应用于预测、分类、回归等领域，在精准营销、个人征信、金融保险等行业都获得了令人满意的表现。然而，这一阶段的 AI 模型依然存在通用性不强、与制造业操作技术等领域知识融合困难等技术限制，表明判别式 AI 技术尚未超越“波兰尼悖论”，即人类的隐性知识依然难以被程式化（Autor，2014）。

2022 年 11 月以来，ChatGPT 及大语言模型的成功推动了生成式 AI 技术的快速发展，并吸引了全球投资机构、互联网企业和传统产业部门的广泛关注。“生成式”一词体现了机器学习技术的转变，即从模式识别到生成自由格式的文本、图像、视频和其他基于数据训练算法生成的人类输出。尽管机器学习、深度学习等新一代人工智能的核心算法没有发生根本性变化，但生成式 AI 还是展现出了有别于判别式 AI 的新特征。第一，基于大语言模型的生成式 AI 表现出了前所未有的通用目的技术潜力（Eloundou et al.，2023；Goldfarb et al.，2023）。以 ChatGPT 的支撑模型 GPT－3.5 为例，模型以 Transformer 为基础架构，通过设置 1750 亿个模型参数、“喂食”大规模高质量训练数

据、引入人类反馈加强学习机制，实现了使用一个大模型解决全部问题的技术路线，不再需要建立适用于个别行业/领域的专业化模型。第二，生成式 AI 正在逐步实现对人类隐性知识和认知能力的编程化、自动化。AI 技术应用由重复性、程式化任务逐步扩展至文本、图像、音频等多模态内容生成领域。生成式 AI 通过学习已有数据的统计特征，生成新的数据，并将其转化为自然语言文本、图像、音频、编程代码等各种模态，可以被广泛应用至新闻稿写作、剧本写作、广告创意、软件编程等创造型专业岗位。早期研究表明，生成式 AI 在实际应用中，正在将高技能员工的数据输入转化为模型生成内容，表现出对人类隐性知识的编程化趋势（Brynjolfsson et al.，2023）。

（二）生成式 AI 技术对劳动就业的影响

人工智能技术主要通过三种方式对劳动就业产生影响：一是替代效应，新技术可能将特定流程自动化，从而减少特定领域的工人需求；二是补偿效应，主要体现在新技术通过提高生产效率，提升企业竞争力和市场占有率，促进企业扩大生产规模，进而提升劳动力需求；三是创造效应，人工智能技术的发展在一些新领域产生新的就业机会，由此带来技术对就业的创造效应（Acemoglu & Restrepo，2019a；Autor & Salomons，2017）。人工智能对就业影响的最终结果将取决于替代效应和增强效应（包含补偿效应和创造效应）之间的竞赛，以及技术采用的速度和工人获得新技能的速度（Autor，2022）。从工作任务和岗位来看，人工智能技术主要影响具有清晰标准或规则、重复性、程式化的工作任务（Autor，2015）。尽管近年来受影响的岗位逐步从体力劳动渗透至初级脑力劳动岗位，但高技能水平、创新型岗位似乎尚未受到技术挑战。Cockburn et al.（2019）在对人工智能技术进行分类后指出，只有当人工智能技术由常见的专用型技术转变为通用目的技术时，才有可能彻底改变传统自动化技术的就业影响机制。

生成式 AI 技术所表现出的通用目的技术潜力似乎正在印证这一判断。尽管实现商业化应用的时间很短，生成式 AI 已经快速渗透到诸多行业，其对劳动力市场的影响也表现出一些新的特征和趋势。首先，生成式 AI 对劳动就业的影响范围明显扩大，许多非程式化、知识型、专业型岗位都面临较大的自动化风险。OpenAI 公司（ChatGPT 软件的开发者）关于 ChatGPT 及其关联技术对美国劳动力市场潜在影响的研究显示，大约 80% 的劳动力可能会有至少 10% 的工作任务受到大模型的影响，其中大约 19% 的劳动力有至少 50% 的任务受到影响，而且这种影响覆盖了所有工资水平的工作，且不限于近期生产率增长较高的行业；在岗位层面，翻译、作家、记者、数学家等专业型、高收入、创意类岗位受到的影响最大；在行业层面，金融投资和保险、数据处理和托

管、信息服务等行业的技术风险系数最高（Eloundou et al.，2023）。Eisfeldt et al.（2023）使用类似方法评估了职业风险系数，并将其与美国上市公司数据进行匹配，提出受生成式 AI 影响最大的职业是涉及非常规、认知任务的职业，且工资水平越高的职业或行业，越有可能接触到前沿技术，受生成式 AI 的影响也越大。Zarifhonarvar（2024）的研究也显示，专业人员、技术人员和辅助专业人员分类下的职业受 ChatGPT 的影响最大，研究将影响程度分为“完全影响”“部分影响”和“完全不受影响”三个等级，上述两类职业分别有 75.4% 和 54.5% 的职业受到完全影响。上述研究结果表明，生成式 AI 对劳动就业的影响范围和程度呈现扩大趋势，涉及非常规、认知任务的专业型工作成为此次技术变革冲击的重点领域。

其次，在基于人类自然语言的多个应用场景中，生成式 AI 工具能够有效提高工作效率和产出质量，通过对低技能水平员工的内容辅助，大幅降低相关岗位的技能门槛。已有研究表明，生成式 AI 在专业写作、客户服务、科研辅助、法律助理等领域已经实现了较好的应用效果，其表现能够媲美高技能员工的工作水平，且技术应用对低技能水平员工的效率提升作用更为显著（Brynjolfsson et al.，2023；Noy & Zhang，2023）。在以聊天问答为主要工作模式的客户服务领域，生成式 AI 会话助手能够有效提高客服人员的工作效率，每小时解决问题的数量提升了 14%，且 AI 工具对新手和低技能水平员工的影响最大，对经验丰富和高技能员工几乎没有产生提升作用（Brynjolfsson et al.，2023）。在专业写作领域，ChatGPT 可以显著提高中级写作人员的工作效率，单位工作时间减少了 0.8 个标准差，而写作产出质量提高了 0.4 个标准差。ChatGPT 通过更多帮助低技能水平员工压缩整体效率分布，因而不同技能水平员工群体之间的不平等得以减少（Noy & Zhang，2023）。

最后，生成式 AI 展现出对人类隐性知识的编程潜力。人工智能技术的应用扩展了人类在理解、注意力、搜索以及概念化等关键领域的认知能力（Tolan et al.，2021）。这些领域的认知能力通常被视为人类的隐性知识，长期以来难以被编程和自动化。生成式 AI 技术在这方面取得了显著的突破。多项实验表明，生成式 AI 对低技能员工更加友好，主要是因为生成式 AI 技术的作用机制是对高技能员工隐性知识的编程化。Noy & Zhang（2023）观察到，在使用 ChatGPT 完成写作任务的过程中，68% 的写作人员直接提交了 ChatGPT 的初始输出内容而没有对内容进行编辑；而在执行大量（可能来自 ChatGPT）文本粘贴后，参与者的活跃时间与最终产出质量之间也不再具有显著相关性，即没有证据表明人工编辑技能可以改善 ChatGPT 的文本输出。Brynjolfsson et al.（2023）对客服人员对话文本内容的分析表明，使用 AI 工具后，低技能和高技

能员工群体的对话文本相似性显著提升，这一结果表明 AI 工具在一定程度上实现了对高技能员工隐性知识的编程化。

（三）生成式 AI 技术对软件编程工作的影响

软件工程/开发的核心是代码编程，但整体来看是一项系统工程。一个软件开发项目主要包括需求分析、设计、编码、测试、系统运营和维护几个主要环节①。随着数字技术的发展，软件工程经历了多次范式的转变，自动化工具在软件开发生命周期中扮演着越来越重要的角色，显著提高了软件工程师的生产力和软件的质量（Ozkaya，2022）。近期生成式 AI 技术和工具的发展进一步拓展了软件工程领域的自动化潜能，在软件开发的不同环节都有重要的应用场景（Russo，2023）。国外科技巨头公司相继推出由生成式 AI（大模型）驱动的代码生成工具，例如 OpenAI 与微软合作开发的人工智能代码助手 GitHub Copilot、DeepMind 公司的 AlphaCode 以及亚马逊公司的 CodeWhisperer，这些系统都尝试通过生成式 AI 技术使编程变得更有成效（Li et al.，2022）。

计算机领域相关研究对生成式 AI 技术在代码生成、代码解释、代码修复等多个任务场景下的表现进行了丰富的检验，不过目前整体评价相对偏低。在代码自动生成方面，现阶段生成式 AI 工具只适用于生成代码片段，还无法生成整段代码（Tian et al.，2023）。一项基于 ChatGPT 与传统编程问答平台 Stack Overflow 的对比研究表明，在 ChatGPT 给出的答案中 52% 不正确，77% 过于冗长，ChatGPT 答案的正确性、一致性、全面性和简洁性都有待进一步检验（Kabir et al.，2023）。不同编程语言的技术成熟度也存在明显差异，在要求 Copilot 使用 4 种编程语言完成 33 个 LeetCode 问题的测试中，Copilot 给出的 Java 代码建议正确性得分最高（57%），而 JavaScript 的正确性得分最低（27%）（Nguyen & Nadi，2022）。在解决编程错误方面，与传统调试工具相比，ChatGPT 提供了一种成本更低、速度更快、专业要求更低的方式解决代码错误，但是 ChatGPT 代码修正的准确性还不够稳定，且依赖于训练数据的质量（Surameery & Shakor，2023）。

由于目前仍处于技术发展的早期阶段，关于生成式 AI 对软件工程就业影响的研究非常有限，已有研究主要通过对实验或测试场景下的数据进行分析。Peng et al.（2023）开展了一个使用 GitHub Copilot 进行 JavaScript 编程的对照实验，结果表

① 这些环节也被称为“软件开发生命周期”，参见 https://bigwater.consulting/2019/04/08/software-development-life-cycle-sdlc/。

明，使用 AI 工具的编程人员完成任务的速度比对照组快 55.8%。但作者们指出该实验还存在较多局限：第一，实验对标准化编程任务进行了检验，与真实工作场景的复杂度相差太远；第二，没有考察 AI 工具对代码质量的影响，与真实工作要求也存在较大差异。因此，有必要对生成式 AI 在软件开发领域的应用及影响开展进一步的分析。

（四）文献评述和研究思路

已有研究结果表明，生成式 AI 的突破和商业化落地是人工智能技术发展历程中的重要里程碑，进一步增强了人工智能成为通用目的技术的潜力。围绕人工智能技术对劳动就业的影响，生成式 AI 展现出了不同于早期 AI 技术的特征和影响趋势。生成式 AI 的主要就业冲击已经开始扩大到高技能水平、非常规认知工作任务和岗位，许多传统意义上的高薪行业和知识型、专业型工作可能面临前所未有的自动化风险，特别是与文字、图像、视频创作等相关的非常规认知任务及工作岗位。同时，进一步的研究表明，即使是在高技能水平、专业型工作岗位范围内，生成式 AI 的影响程度和机制也存在差异。生成式 AI 对软件工程行业相关工作的影响（无论是替代效果还是生产率提升效果）明显滞后于线上客服、写作、翻译等专业型工作。因此，本文通过分析生成式 AI 工具在真实工作场景下的使用情况，从软件开发工作的行业类型、任务类型等角度，分析生成式 AI 应用于专业型工作中所遇到的难点。

总体来说，本文聚焦于中国市场环境下生成式 AI 技术对软件工程行业劳动就业的影响。首先，分析生成式 AI 应用对软件开发工作的整体影响效果，探讨其是否实现了对软件开发工作的劳动替代或效率提升。其次，分析生成式 AI 工具在不同类型应用场景下对软件工程师的影响，从训练数据和任务属性等方面解释生成式 AI 工具现阶段的局限性。最后，结合编程用例评分数据和软件开发人员使用 Copilot 的记录，对生成式 AI 工具辅助软件开发工作中的制约因素展开分析，并提炼生成式 AI 工具对软件开发等专业型工作的影响机制。本研究有助于更好地了解生成式 AI 的工作原理及其对专业型工作的影响机制，同时也为软件工程行业引入生成式 AI 工具提供建议。

三　研究设计

（一）数据来源

我们与一家提供跨行业解决方案的信息技术服务公司（以下简称“A 公司”）合

作，基于该公司的实际业务，测试和评估生成式 AI 辅助编程工具在不同应用场景下的使用情况。A 公司是国内信息技术服务领域的知名上市公司，为互联网、汽车、金融、企业服务、云服务等诸多行业提供信息技术服务。本研究主要关注生成式 AI 工具在编码阶段的应用情况，特别是生成式 AI 工具如何支持软件工程师的工作。在研究准备阶段，我们与 A 公司各业务部门的核心开发团队和技术专家进行了数轮讨论，选取了各部门有代表性的业务场景和软件开发任务，同时为确保信息安全对测试用例进行了脱敏处理，最终确定的应用场景如表 1 所示。在 9 类应用场景下，分别设有不同的软件开发任务，每个任务包含一个或多个编程用例，总共获得 112 个编程用例评分样本，作为生成式 AI 工具评分分析的基础单元。本次研究仅覆盖 Java 和 Python 语言的开发场景。

表 1　研究涵盖的应用场景

编号	应用场景	应用场景描述	编程用例数量
1	云服务	提供企业级云原生微服务开发、运管服务业务开发等	20
2	数据智能	大数据采集、加工、处理开发、机器学习数据建模训练等	13
3	企业应用	企业级项目管理场景、泛企业资源规划（ERP）场景、病害管理等业务场景开发	19
4	企业服务	结构维修、数字化转型大屏等通用类企业服务开发	14
5	金融应用	金融场景下的基础开发和业务开发	11
6	智能车	智能车云平台、路测数据处理、车企数字化平台开发等	13
7	智能物联网	物联网设备通信、可穿戴系统设备开发和测试开发等	9
8	智能终端	终端服务交互请求开发	5
9	通用工具	通用技术和通用数据采集、分析和展示开发	8

本研究选择了 OpenAI 公司与微软公司合作开发的人工智能代码助手 GitHub Copilot 作为测试工具。Copilot 是生成式 AI 领域最具影响力的辅助编程产品之一，于 2021 年 6 月发布。它结合了 OpenAI 公司强大的大模型能力和 GitHub 丰富的代码训练样本，在软件开发界引起了广泛关注。Copilot 利用大型语言模型，由 OpenAI 的 Codex 模型驱动，该模型是 GPT－3 的后代，是在 GitHub 存储库的大量代码上训练出来的，能够为开发者提供代码生成的智能建议①。Copilot 有三个主要功能：根据注释和函数名称生成代

① 参见 https：//github. com/features/copilot。

码、为已经实现的代码生成测试以及为重复的代码自动填入。该工具的操作界面简单易用，通过键入少量的代码提示语句，即可生成完整的代码段，能够实现代码补全、代码注释、函数调用等多种代码生成功能，其支持的语言也非常广泛，包括 C、C ++ 、C#、Go、Java、JavaScript、PHP、Python 等。本研究通过分析 A 公司软件开发人员在真实工作场景下使用 Copilot 的情况，研究其对软件开发工作的影响和作用。当前已有一些研究对 Copilot 的使用情况进行实证评估（Nguyen & Nadi，2022；Peng et al.，2023），这些研究主要关注 Copilot 能够提供的原始代码建议和可读性，而不是 Copilot 与软件开发人员之间的关系。此外，这些研究通常利用 Copilot 解决 LeetCode 等网站的算法问题，难以反映软件工程师在真实工作场景中遇到的编程问题类型。本文研究的重点不是 Copilot 可以处理的编程任务类型或难度级别，而是当它被用于协助软件开发工程师解决实际工作任务时，所发挥的作用和扮演的角色，以及工具的使用对软件开发工作带来的影响。

（二）研究方法

针对每一项应用场景，我们在该场景所属业务部门选择 3 名资深软件开发工程师进行小组测试（共 9 组，27 人参与测试）。测试人员通过使用 Copilot 产品生成对应开发场景下的软件开发任务的代码，并评估其效果。之所以选择资深的软件开发工程师，是为了尽可能保证评估的准确性。已有研究显示，Copilot 被新手开发者使用时，他们可能会因为缺乏专业知识而无法识别错误或非最佳的解决方案（Dakhel et al.，2023）。每名测试人员详细记录了与 Copilot 交互的过程，包括每次输入注释或代码得到的反馈、Copilot 是否能够理解用例的需求、生成的代码是否达到预期、代码未达到预期的可能技术原因以及使用 Copilot 的操作体验等。与此同时，本文也建立了统一的评测指标体系来观测和评估每个编程用例的代码生成效果，测评指标包括代码推荐质量、自动生成代码占比、效率提升度和操作体验四个方面，指标体系如表 2 所示。各组的测试人员根据三级指标设定的标准对每个编程用例进行评分，如果小组成员打分差距在 2 分以内，取 3 人打分的平均值为最终评分；如果小组成员打分差距在 2 分以上，3 人讨论后进行二次测试，对评分进行验证和校准。

针对不同应用场景，本文通过计算每种场景所包含的用例评分的算术平均值，得到 9 类应用场景的三级指标评分。二级指标和一级指标则以等权重的方式计算加权平均值。早期相关研究主要关注生成式 AI 工具在软件开发应用中的自动生成代码占比（即代码数量），而对代码质量、效率提升、操作体验的关注不足。因此，本文选择了等权重的加权平均计算方式，以求更加全面地反映生成式 AI 工具对软件开发工作的影

响。具体计算方式如式（1）至式（3）所示，最终计算得到的9类应用场景的各项指标得分如表3所示。

$$\text{代码提升效应评分} = \text{代码有效性评分} \times 0.5 + \text{自动生成代码评分} \times 0.5 \quad (1)$$

$$\text{效率提升效应评分} = \text{效率提升度评分} \times 0.5 + \text{操作体验评分} \times 0.5 \quad (2)$$

$$\text{综合效应评分} = \text{代码提升效应评分} \times 0.5 + \text{效率提升效应评分} \times 0.5 \quad (3)$$

本文后续内容将从应用场景、软件开发任务、软件编程用例三个层面对Copilot评分结果进行深入分析和讨论。首先，基于对A公司9类应用场景的评分结果，对Copilot在软件编程实际应用中的整体表现作出判断，并分析Copilot在不同应用场景下的影响效果与评分特征。然后，从软件开发任务层面分析Copilot在业务型、通用型、技术型场景下的评分差异，探讨生成式AI工具的局限性和应用场景的异质性。最后，结合112个编程用例的评分数据和软件开发人员使用Copilot的记录，对生成式AI工具在辅助软件开发工作中的制约因素和影响机制展开分析。

表2　Copilot使用情况测评指标体系

一级	二级	三级	指标说明
综合效应	代码提升效应	代码有效性	根据工具给出推荐代码的有效性（与预期符合）和质量打分： 推荐与预期匹配，且代码质量好（7~10分）； 推荐与预期基本匹配，但代码质量一般（4~6.9分）； 推荐与预期有些匹配，代码质量一般（2~3.9分）； 推荐与预期很少或不匹配，代码质量一般或较差（0~1.9分）。
		自动生成代码	工具能在所在场景自动生成的代码占全部代码的比例： 能生成60%以上（7~10分）； 能生成40%以上（5~6.9分）； 能生成20%以上（2~4.9分）； 只能生成不到20%（0~1.9分）。
	效率提升效应	效率提升度	工具能在所在场景提高开发效率的比例： 能提升40%以上（7~10分）； 能提升20%以上（5~6.9分）； 能提升10%以上（2~4.9分）； 提升不到10%（0~1.9分）。
		操作体验	根据实际操作体验顺畅性程度评分： 操作顺畅，无被干扰感（8~10分）； 体验良好，基本顺畅（6~7.9分）； 操作不顺畅，有语法语义报错、代码内容混乱等问题，体验有待改进（0~5.9分）。

表 3　Copilot 在不同应用场景中的评分结果汇总

编号	应用场景	代码有效性评分	自动生成代码评分	效率提升度评分	操作体验评分	代码提升效应评分	效率提升效应评分	综合效应评分
1	云服务	4. 46	3. 06	1. 90	5. 20	3. 76	3. 55	3. 65
2	数据智能	2. 98	2. 83	1. 58	2. 03	2. 90	1. 80	2. 35
3	企业应用	4. 23	3. 81	2. 44	3. 66	4. 02	3. 13	3. 58
4	企业服务	5. 15	6. 40	5. 50	6. 45	5. 78	5. 98	5. 88
5	金融应用	4. 40	4. 00	4. 60	5. 50	4. 20	5. 05	4. 63
6	智能车	4. 70	4. 10	3. 70	4. 20	4. 40	3. 95	4. 18
7	智能物联网	4. 50	4. 40	3. 10	7. 00	4. 45	5. 05	4. 75
8	智能终端	6. 00	5. 80	4. 70	5. 20	5. 90	4. 95	5. 43
9	通用工具	5. 30	4. 90	5. 00	5. 40	5. 10	5. 20	5. 15

资料来源：根据评分数据计算得到。

四　生成式 AI 工具对软件开发工作的影响与制约因素

（一）生成式 AI 工具对软件开发工作的整体影响

本文从代码提升效应、效率提升效应和综合效应三个角度，对 A 公司软件开发人员使用 Copilot 的实际获益情况进行了分析，以此呈现生成式 AI 工具对软件开发工作的影响。代码提升效应主要从代码推荐质量（推荐代码的有效性）和数量（自动生成代码占比）两个方面进行综合评估，考察 Copilot 推动软件开发自动化的能力。效率提升效应从效率提升度和操作体验两个方面进行评估，用于测度 Copilot 对编程人员整体工作效率和工作体验的影响程度，作为除代码提升之外的测评角度。

从代码提升效应来看（图 1a），在 9 项应用场景中，智能终端、企业服务和通用工具的评分最高，场景平均分分别为 5. 90、5. 78 和 5. 10，整体代码提升效应超过 50%。上述场景主要包含智能终端交互开发、企业通用类服务开发、通用技术和通用数据开发，即可跨行业、跨企业使用的通用设备、软件、技术、数据类开发。相关场景所涉及的代码逻辑通用性强、历史代码积累多且质量高，Copilot 可以直接调用或学习历史代码，由此带来显著的代码提升效应。其余 6 个应用场景（云服务、数据智能、企业应用、金融应用、智能车、智能物联网）的代码提升效应评分均未超过 50%，Copilot 工具的使用还未对这些场景的代码编写工作产生较好的提升作用。上述 6 个场景与企业具体的业务流程、细分行业的业务逻辑等结合紧密，一方面对 AI 模型和从业人员的行业知识、经验积累、技能水平等提出了更高要求；另一方面，由于代

码可能涉及企业商业机密，也将限制企业分享其高质量历史代码，阻碍模型训练数据集的构建。

从代码提升效应所涉及的推荐代码质量和自动生成代码占比来看，现阶段 Copilot 还无法直接替代代码编写任务，自动生成代码的质量和数量评分都不高。以数量评分（自动生成代码评分）为例，9 项应用场景评分分布在 2.8 ~ 6.4 之间，即 Copilot 自动生成代码仅占全部代码的 20% ~ 40%。这与人类在自然语言相关应用场景中的表现相比，还存在较大差距。已有研究表明，在专业写作、电话客服等领域，生成式 AI 工具已经可以提供完整的、大篇幅的自然语言文本生成，从业人员可以直接使用且改动较少（Brynjolfsson et al.，2023；Noy & Zhang，2023），即基本相当于提供 100% 的文本自动生成。而在软件编程行业的应用场景下，Copilot 只能逐行提供代码片段，无法直接实现完整的代码任务。同时，用于表征代码质量的推荐有效评分结果分布在 3.0 ~ 6.0 之间，即 Copilot 自动生成代码质量一般，有效性仅能部分满足预期。

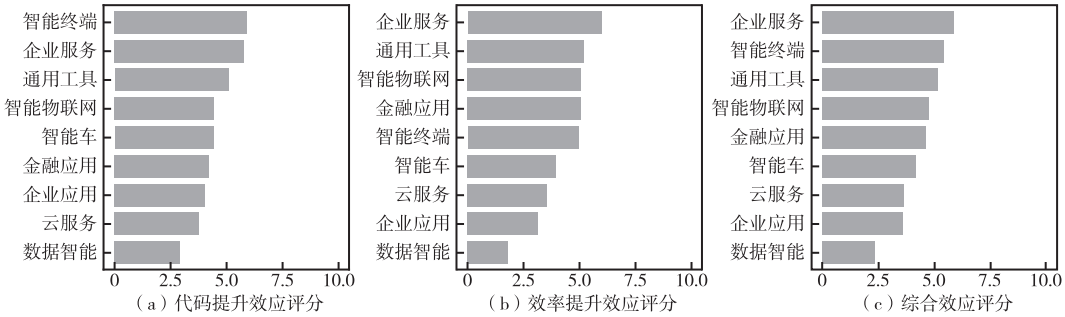

图 1　Copilot 在不同应用场景中的评分

资料来源：根据评分数据绘制得到。

从效率提升效应来看（图 1b），9 项应用场景中 Copilot 效率提升效应评分的分布更为平均，且对场景通用属性的区分度不高。企业服务、通用工具、智能物联网、金融应用 4 项场景的效率提升效应评分均超过 5.0，Copilot 对通用属性较强场景的效率提升效应没有展现出明显优势。这表明历史代码积累、行业知识等因素对 Copilot 工具在效率提升方面的影响有限。值得注意的是，云服务和数据智能两个场景的代码提升效应和效率提升效应评分均排名靠后，其原因可能在于相关场景所涉及的技术领域较为前沿、对编程人员的技能水平要求较高，现有 AI 工具还无法实现良好的辅助作用。本文将在后续分析中，对上述场景进行更加深入的分析探讨。

综合效应评分反映了不同应用场景下 Copilot 在代码自动化和效率提升方面的总体

情况。结合表 3 和图 1c，本文发现 Copilot 综合效应评分展现出了如下特征。第一，Copilot 在通用属性较高的应用场景中提升效应较高。企业服务、智能终端、通用工具 3 项场景属于跨行业、跨企业的通用软件开发，其综合效应评分分别为 5.88、5.43 和 5.12，明显优于其他 6 个应用场景。第二，Copilot 在行业属性较高的应用场景中，提升效应有限。智能物联网、金融应用、智能车、企业应用 4 项场景的行业属性较高，其工作任务包含较多业务逻辑设计、行业知识和机理的个性化编程任务，Copilot 综合评分偏低，反映出目前生成式 AI 工具尚未实现在软件编程领域的行业知识编码化突破。第三，当应用场景涉及云计算、机器学习、数据建模等前沿技术能力时，Copilot 表现最弱。

（二）生成式 AI 工具在不同应用场景下的异质性影响

在对生成式 AI 工具应用场景分析的过程中，本文发现应用场景是否具有业务属性、是否涉及前沿技术对评分的影响很大。因此，本文将 A 公司的 9 项应用场景划分为通用型、业务型、技术型 3 个类别。本文将企业服务、智能终端、通用工具划分为通用型应用场景，因其不具备明显的行业属性，可以实现跨企业、跨行业的应用范围；将企业应用、金融应用、智能车、智能物联网划分为业务型场景；将云服务和数据智能划分为技术型应用场景，因其任务涉及云原生、机器学习等前沿技术，完成难度较大。图 2 展示了 3 类应用场景的代码提升效应和效率提升效应评分，本文对评分进行了线性拟合。整体来看，3 类场景下代码提升效应和效率提升效应都是同向变动的，即代码提升效应越高，效率提升效应也越强。通用型场景下 Copilot 的代码提升和效率提升效应高于业务型和技术型，业务型场景的评分波动最大，通用型场景的评分波动最小。本文对三类应用场景下所涉及的具体任务进行更加深入的分析，以解释 Copilot 在不同类型应用场景下的影响效果和异质性特征，并尝试分析 Copilot 在实际应用中面临的制约因素，作为后续机制分析的基础。

1. 通用型应用场景

通用型应用场景共包含 5 项任务，图 3 展示了各项任务的代码提升效应评分和效率提升效应评分。在通用型应用场景下，各项细分任务之间的评分相对稳定，评分波动明显小于业务型任务。首先，企业服务场景下的“结构维修”任务评分表现出色，代码提升效应评分（7.85）和效率提升效应评分（7.45）均为全样本范围内得分最高的任务。原因在于该任务主要是对已有代码的查询搜索类用例，例如“生成（创建维修单）代码”或“生成（验收）代码”等。一方面可以充分利用已有训练数据和代码库；另一方面该任务类型也非常符合生成式 AI 的对话式工作逻辑，代码搜寻和提取需

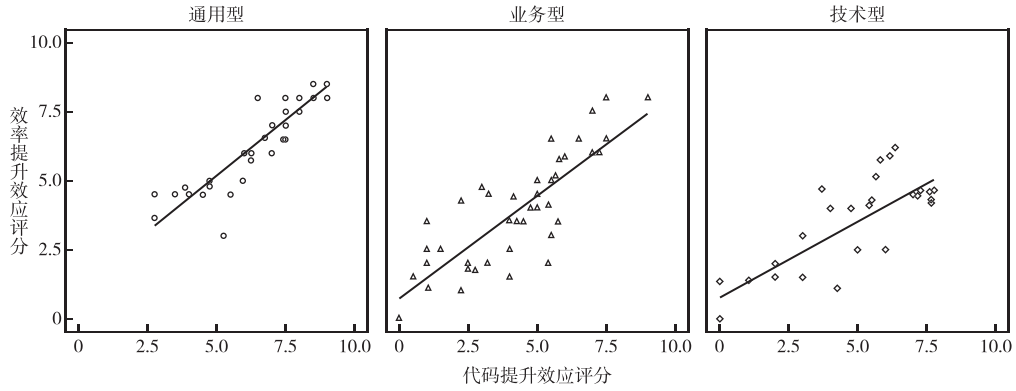

图 2　三类应用场景下 Copilot 的代码提升效应评分和效率提升效应评分

资料来源：根据评分数据绘制得到。

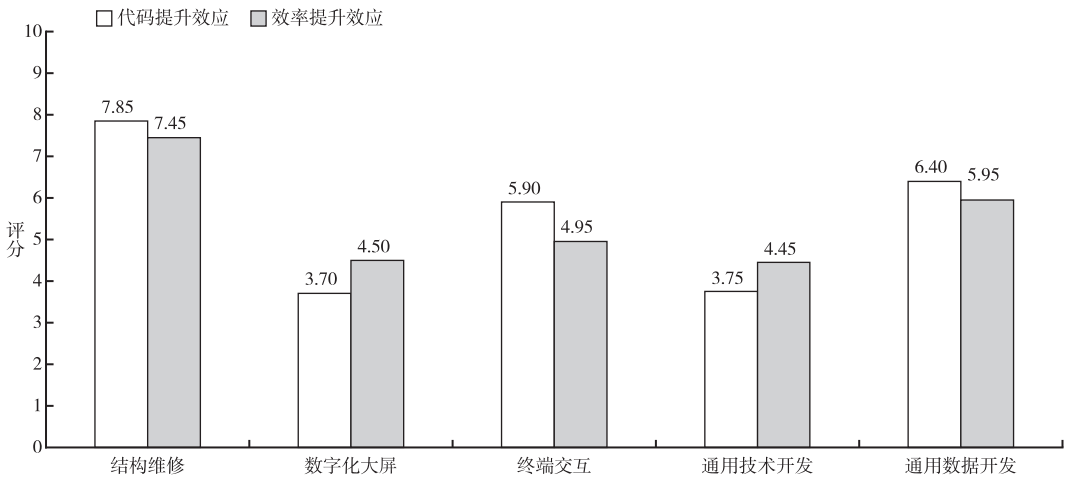

图 3　Copilot 在通用型应用场景中的任务评分

资料来源：根据评分数据绘制得到。

要软件开发人员与 AI 工具之间进行多轮对话。其次，在通用型任务场景下，“通用数据开发”任务评分较高，而“通用技术开发”任务评分较低。本文对两类任务的具体用例进行了对比分析，发现“通用数据开发”任务主要涉及数据的采集、分析、展示等标准化代码生成，而“通用技术开发”所包含的用户管理、权限管理等用例，涉及模型创建、校验及格式化等非标准化代码编写工作。本文推断，Copilot 在通用型场景中的评分结果很大程度上取决于训练数据的可得性和编程逻辑的复杂度。从训练数据

的角度，当训练代码库较为成熟或代码标准化程度较高时，Copilot 的评分更高；从任务逻辑的角度，Copilot 在“查询回复”类逻辑简单的工作任务中评分较高，而在模型创建、校验、格式化等逻辑较为复杂的工作任务中评分较低。

2. 业务型应用场景

业务型应用场景主要涉及 12 项垂直业务领域的细分任务，从图 4 中可以看出，不同任务之间展现出了较大的评分差异。本文选取了评分较低的几项任务，对其相关用例进行了详细分析。首先，Copilot 在涉及专业知识的任务中评分较低。以金融应用场景为例，“金融基础”任务评分明显高于“金融业务”。本文进一步分析两类任务的具体用例后发现，“金融基础”包含“返回页面和简单数据”“用户登录视图”“随机数字字符串”等经典代码生成任务，Copilot 可以很好地利用已有训练数据，提供令人满意的代码。而“金融业务”则包含了汇率换算、信用评级、实时汇率获取等金融具体业务实践。尽管只是非常基础的金融业务，但 Copilot 完成相关任务所获得的评分非常低，特别是代码生成类评分只有 1 ~ 2 分，反映出生成式 AI 工具在专业知识编程方面还存在较大短板。其次，Copilot 在涉及机构知识的任务中评分较低。以企业应用场景为例，在具体的任务描述中，本文发现 Copilot 在完成业务型编程任务的过程中，需要理解和掌握企业内部的业务逻辑、合同类别、产品特征、客户需求等较敏感信息。此类机构知识涉及企业的具体业务和商业机密，通常不会作为训练数据提供给第三方的生成式 AI 模型和工具。因此，Copilot 工具在涉及产品查询、需求分析等方面的任务中表现较差。

3. 技术型应用场景

技术型应用场景主要涉及通用云服务、云管服务、大数据和机器学习四项任务，具体评分如图 5 所示。其中，“云管服务”和“机器学习”的评分最低，甚至在个别用例中出现了 0 分，即 Copilot 完全无法理解指令，没有给出任何推荐代码。其原因可能在于生成云资源相关代码、实现机器学习等数据建模具有较高的逻辑复杂性和技术难度，Copilot 底层的 AI 模型训练和学习还没有覆盖相关技术知识，因此无法生成相应代码。

（三）生成式 AI 工具在软件开发应用中的制约因素

基于编程任务层面评分结果的分析，本文认为生成式 AI 工具在不同类型场景下的评分差异主要受到训练数据、复杂逻辑、隐性知识、交互能力等因素的影响。这些因素也是生成式 AI 工具在软件开发行业实际应用场景中所面临的主要制约和挑战。

第一，现阶段生成式 AI 工具的表现主要受限于训练数据的可获得性。通用型任务

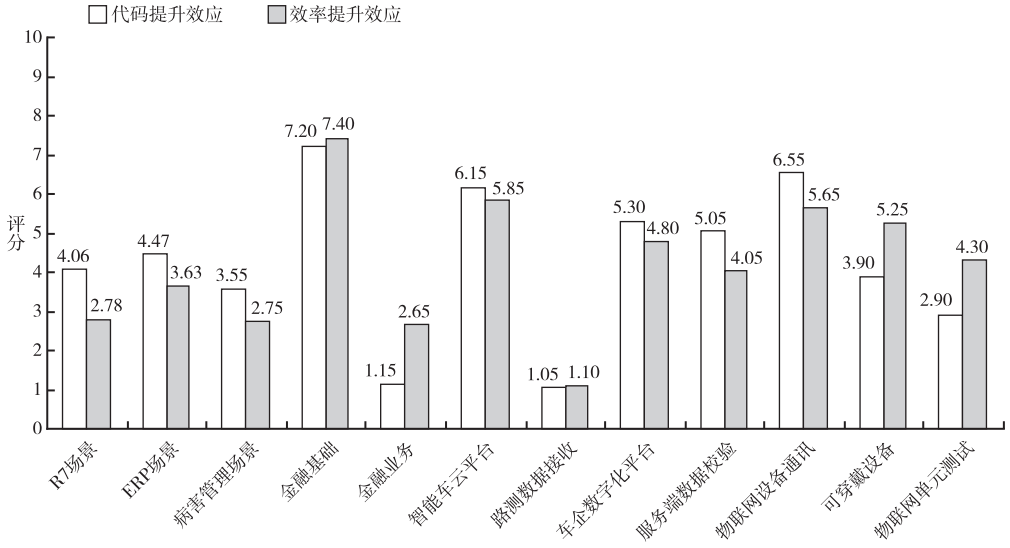

图 4　Copilot 在业务型应用场景中的任务评分

资料来源：根据评分数据绘制得到。

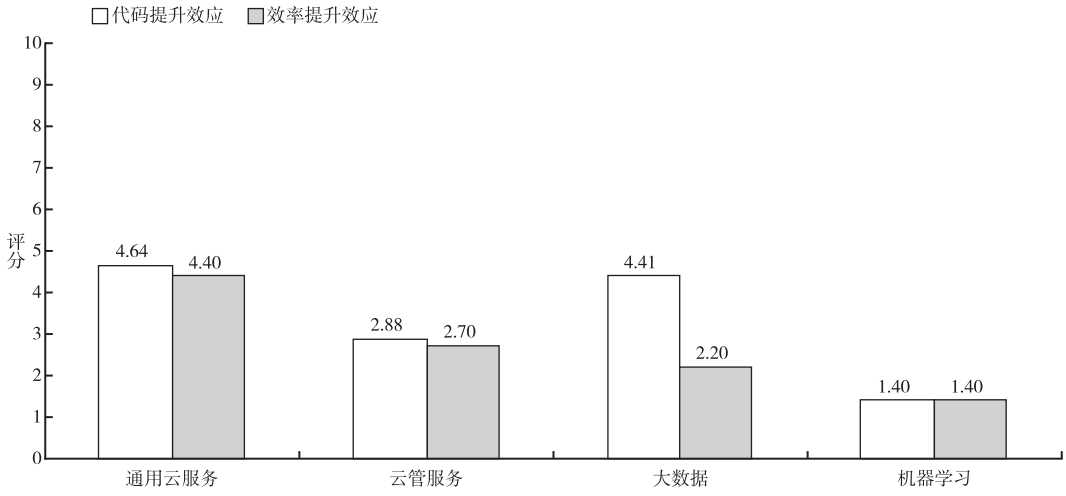

图 5　Copilot 在技术型应用场景中的任务评分

资料来源：根据评分数据绘制得到。

所涉及的代码更为标准化，相关训练数据的可得性更高，因此模型在充分利用更高质量数据的基础上，能够在通用型任务中取得较高得分。而业务型和技术型任务涉及企业内部信息或前沿技术，相关训练数据难以获得或尚未形成，导致生成式 AI 工具难以

理解任务意图或实现任务目标，因此 Copilot 在相关场景中的评分明显低于通用型任务。

第二，生成式 AI 工具在逻辑复杂度较高的任务中评分较低。以 ChatGPT 为代表的大语言模型实现了对人类语言和上下文的良好理解和生成能力，在与人类的“问答交互”式场景中表现突出。类似地，Copilot 在“查询回复”类简单逻辑的编程任务中能够取得较高评分，而在模型创建、校验、格式化等逻辑较为复杂的工作任务中评分较低。

第三，生成式 AI 工具在隐性知识的自动化编程能力方面有限。Copilot 工具在业务型、技术型场景中的得分较低，主要受到专业知识、机构知识、技术知识的限制，而这些知识代表了软件开发行业中除编程技术能力以外的人类隐性知识。在这些领域，Copilot 尚未展现出较强的自动化编程能力，与人类语言的自动化生成和创造能力相比还存在较大差距。

第四，生成式 AI 工具的最终表现还取决于软件工程师与工具之间的交互能力。生成式 AI 工具在训练数据、复杂逻辑、隐性知识等方面的制约，可以通过软件工程师与 AI 工具之间的问答交互予以补偿。本文在任务和用例评分分析过程中发现，在软件工程师多轮提示词的样本中，AI 工具能够更好地理解编程任务，获得更高评分。

五　生成式 AI 工具对软件开发工作的影响机制

基于 A 公司 9 项应用场景的任务评分结果分析，本文发现生成式 AI 工具目前受到训练数据、复杂逻辑、隐性知识、交互能力等因素的制约和影响。下面，本文将使用每项编程任务下的用例评分，分析生成式 AI 工具在软件开发流程各环节所发挥的作用，以及上述因素在各环节如何影响生成式 AI 工具的表现，并对生成式 AI 技术引发的软件开发工作范式变革做出长期展望。

（一）生成式 AI 工具对软件开发工作的影响机制

按照软件开发流程，本文将软件工程师的工作内容大致划分为理解需求、解决问题、生成代码三个环节，明确 Copilot 在各环节所发挥的作用，识别训练数据、复杂逻辑、隐性知识、人机交互等因素对 Copilot 表现的影响，分析生成式 AI 工具对软件开发工作的具体影响机制，以期更好地理解生成式 AI 技术对专业型工作的影响和长期变革趋势。

1. 生成式 AI 工具理解需求的能力

对用户需求的理解和分析是软件开发工作的起始环节，编程辅助工具的需求理解能力是其实现编程任务的重要基础。在对 Copilot 进行评分的过程中，软件工程师对

Copilot 理解需求的能力和表现做了详细记录。由于研究选取的开发用例都是不同业务场景下比较基础、简单的任务，112 个编程用例中仅有 20 个用例（17.9%）无法理解任务需求。提示词或注释的使用是影响生成式 AI 工具需求理解能力的重要因素，112 个编程用例中有 69 个用例（61.6%）Copilot 能准确理解其业务需求；在软件开发人员反复提示后，这一数字增加到 92 个用例（82.1%），表明软件开发人员与 Copilot 的交互能力有助于提升生成式 AI 工具的表现。

软件开发工程师的记录还显示，尽管 Copilot 在需求理解方面表现不俗，但其对任务的理解能力与人类的理解能力仍存在很大差距，特别是无法理解任务描述中的一些细节以及较长的任务描述。这种情况在业务型和技术型场景下更为突出。这与任务评分的分析结果一致，一定程度上体现了复杂逻辑对生成式 AI 工具的限制。现阶段，生成式 AI 工具在“查询回复”类简单逻辑编程应用中表现优异，而当编程逻辑复杂度提高、编程需求较长或细节较多时，生成式 AI 工具的表现会受到较大限制。

对用例评分数据进行分析显示，能否理解需求对最终代码推荐的有效性和编程效率提升有重要影响①。如图 6 所示，Copilot 在理解需求的情况下（包括经提示后理解），代码有效性的平均分约为 5.8，而在不理解需求的情况下平均分只有 1.7；Copilot 在未经提示就能理解需求的情况下，软件开发人员对编程效率提升的评分也显著高于其他两种情形。

本文对不同应用场景下 Copilot 理解需求的表现进行了分析。图 7 中，圆形标记代表 Copilot 不需要提示词就可以直接理解编程需求的用例，三角形标记代表 Copilot 无法理解编程需求的用例，菱形标记表示 Copilot 在软件工程师的提示词或注释帮助下能够理解编程需求的用例。评分分布结果表明：第一，通用型用例基本不需要提示或注释，Copilot 就能够理解编程需求，自动生成质量较高的代码；第二，提示对业务型用例的提升作用最大，表现为使用提示的用例更多集中在业务型类别，且使用提示的业务型用例评分较高；第三，技术型用例受提示的影响较少，用例评分也没有展示出显著的提升。

分场景用例评分结果表明，Copilot 的需求理解能力还受到训练数据的制约。生成式 AI 工具对任务需求的理解能力在很大程度上取决于训练数据的可得性，通用型任务相关训练数据的可得性更高。例如，在互联网开源社区中，有大量经验丰富的软件开发人员贡献高质量代码数据，大语言模型在充分利用这些高质量数据的基础上，能够

① 为了更好地展示 Copilot 需求理解能力的影响，本文直接使用了三级指标中的代码有效性评分和效率提升度评分，而不是加权后的代码提升效应评分和效率提升效应评分。

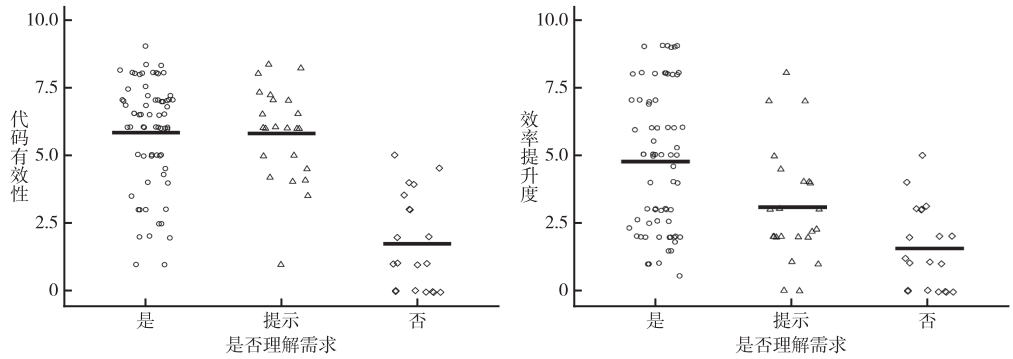

图 6　Copilot 是否能够理解需求对代码有效性与效率提升度的影响

资料来源：根据评分数据绘制得到。

在通用型任务中更好地理解需求，取得较高得分。而在业务型和技术型应用场景中，由于相对封闭，代码的可得性较低，大语言模型难以通过训练高质量数据提升性能，在理解任务需求上面临较大难度。

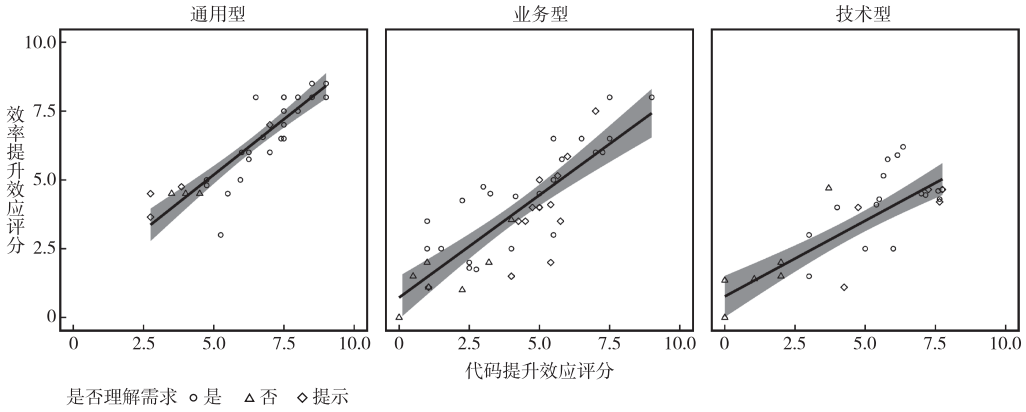

图 7　三类应用场景下 Copilot 的代码有效性和效率提升度评分

资料来源：根据评分数据绘制得到。

2. 生成式 AI 工具解决问题的能力

在完成对编程任务的需求理解后，生成式 AI 工具需要通过搜索内部代码库或网络公开资源，提供相关的代码片段或提示，协助软件开发人员解决编程问题、实现编程任务目标。生成式 AI 工具的问题解决能力是其核心能力所在。前文针对场景和任务的分析显示，生成式 AI 工具在业务型、技术型场景中的得分较低，表明生成式 AI 工具

在解决问题环节可能受到了专业、机构和技术等隐性知识的限制。本文从编程用例层面对此进行了进一步分析和验证。

首先，本文从是否需要行业知识和机构知识两个角度，分析编程任务的知识依赖度对生成式 AI 工具解决问题能力的影响。根据参与测试的软件开发人员对每项编程用例是否涉及行业知识和机构知识进行的评估，图 8 展示了是否需要行业知识对 Copilot 的代码有效性与效率提升度的影响。整体来看，在不需要行业知识的编程用例中，使用 Copilot 的代码有效性和效率提升度平均分都高于需要行业知识的编程用例，并且是否依赖行业知识对代码有效性的影响更大。本文进一步分析了不同应用场景下的情况。由于技术型应用场景较少有编程用例需要行业知识，因此只对比了通用型和业务型两类场景。如图 9 所示，是否需要行业知识对业务型应用场景的影响明显大于通用型场景；业务型场景对行业知识的依赖度更高，在需要行业知识的情况下，Copilot 推荐代码的有效性和效率提升度低于不需要行业知识的情况，不过效率提升度受行业知识的影响更小。不论是代码有效性还是效率提升度，业务型场景下需要和不需要行业知识两类用例所表现出的差异均大于通用型场景。

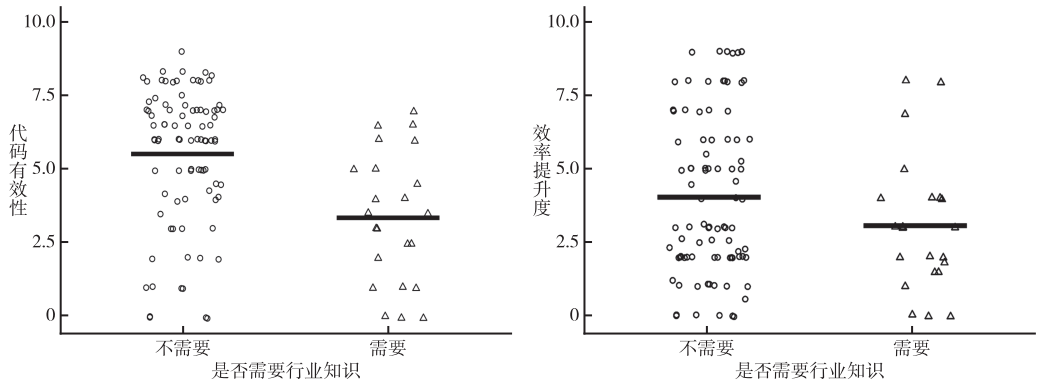

图 8　是否需要行业知识对 Copilot 的代码有效性与效率提升度的影响

资料来源：根据评分数据绘制得到。

对机构知识的分析表现出类似的特征。图 10 显示，编程任务是否需要机构知识对业务型、技术型应用场景的影响明显大于通用型场景，尤其表现在 Copilot 推荐代码的有效性方面。在技术型场景下，需要和不需要机构知识两类用例的平均分相差 6 分以上，这个差距是通用型场景的两倍。是否需要机构知识对 Copilot 带来的效率提升度影响不大，在三类应用场景中，反而是对通用型场景的效率提升度影响最大。

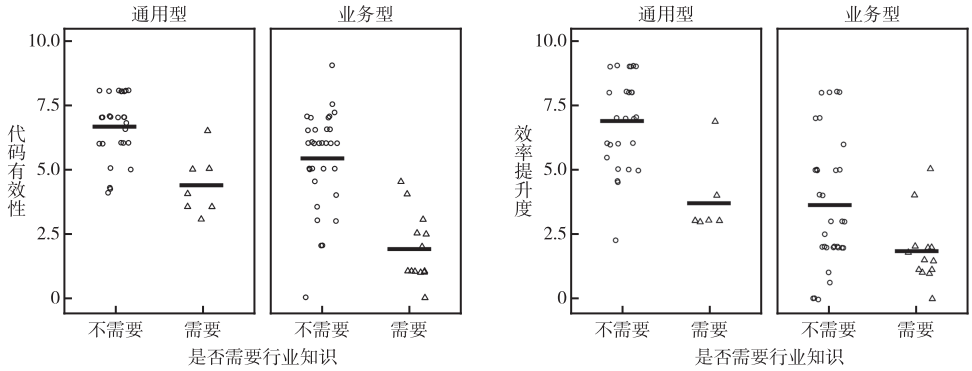

图 9　不同应用场景下行业知识对 Copilot 代码有效性和效率提升度的影响

资料来源：根据评分数据绘制得到。

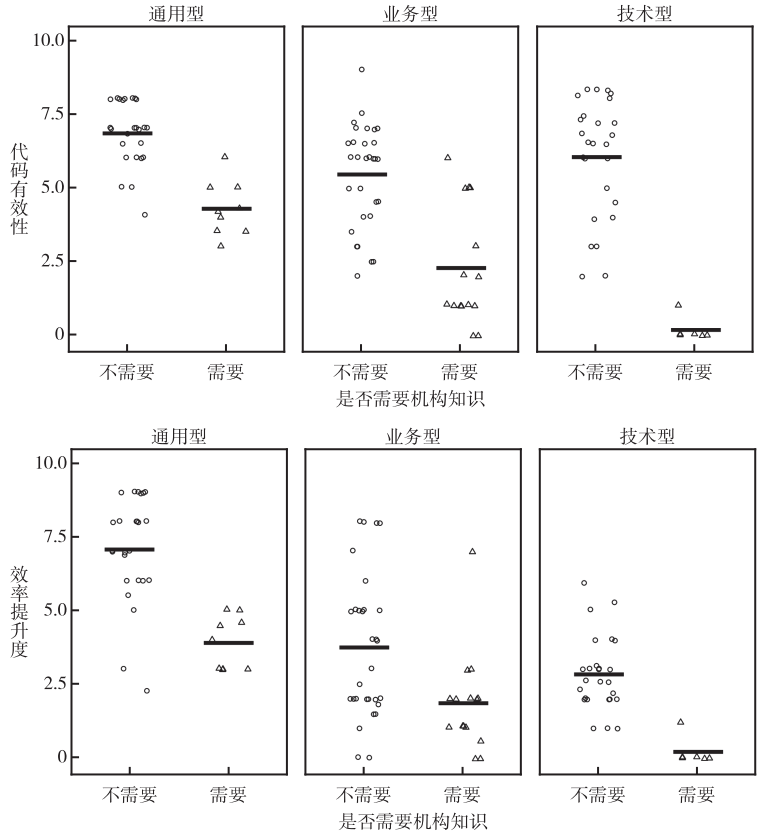

图 10　不同应用场景下机构知识对 Copilot 代码有效性和效率提升度的影响

资料来源：根据评分数据绘制得到。

结合前文关于提示词的分析，提示词或注释的使用效果在很大程度上可以反映出行业知识、机构知识的“隐性知识”属性。专业知识和机构知识能够通过软件编程人员的提示词或注释，转化为生成式 AI 模型的训练数据，即专业知识和机构知识相对“显性”，能够通过自然语言传递给 AI 模型，用于提升模型效能。与之相比，云计算、数据建模、机器学习等前沿技术类知识则更为“隐性”，软件工程师很难通过自然语言将其转化为高质量的训练数据，提升模型的理解能力和表现。因此，在解决问题环节，软件开发人员的交互能力能够在一定程度上弥补行业和机构类隐性知识对生成式 AI 工具的限制，而技术类隐性知识更难通过提示词或注释形式输入给 AI 工具。

其次，本文进一步考察了技术复杂度对生成式 AI 工具解决问题能力的影响。根据参与测试的软件开发人员对每项编程用例是否涉及逻辑问题的评估，图 11 展示了不同应用场景下是否涉及逻辑问题对 Copilot 代码有效性和效率提升度的影响。从图中可以看出，在三类应用场景中，当编程任务涉及逻辑问题时，Copilot 推荐代码的有效性更低，这一趋势在业务型和技术型场景下更加显著，且在技术型场景下两者的差距最大；编程任务是否涉及复杂逻辑对效率提升度的负面影响在整体层面不显著，但在业务型场景下表现出明显的不利影响。

3. 生成式 AI 工具生成代码的能力

为软件开发人员实现全部或部分代码的自动化生成，是 Copilot 等生成式 AI 工具在软件编程领域的主要贡献和目标。本文对软件开发工程师使用 Copilot 进行自动化编码的记录进行了详细分析，发现 Copilot 与许多软件自动化编程工具的不同之处在于，其与软件开发工程师有更强的交互属性。软件开发工程师通过输入问题让 Copilot 理解其需求，Copilot 给出代码建议，通常是较小的代码片段；软件工程师根据生成的内容给出反馈，修改注释让 Copilot 更好地理解需求完成任务，或者自行修改代码。在完成编程任务的过程中，这一交互过程可能持续数轮。在 Copilot 协助软件开发人员完成编程任务的过程中存在非常高频的互动，在所有样本中，超过 92% 的用例需要软件开发人员对 Copilot 推荐的代码进行修改，其中超过一半需要进行逻辑补齐，即代码具有明显的逻辑问题（如表 4 所示）。这一特征与其他生成式 AI 应用场景（客服、写作、广告创意等）存在明显差异，其原因可能在于编程对正确性、逻辑性的要求更高。程序代码作为一种文字表现形式，与专业写作、翻译等内容生成任务相比，具有更高的价值密度，代码中的文字和符号都有特定的意义，多一个词或者符号就可能完全改变代码的逻辑。因此，从外部约束来看，生成式 AI 技术对专业型工作的替代/赋能能力很大程度上受制于工作任务对正确性的要求，像软

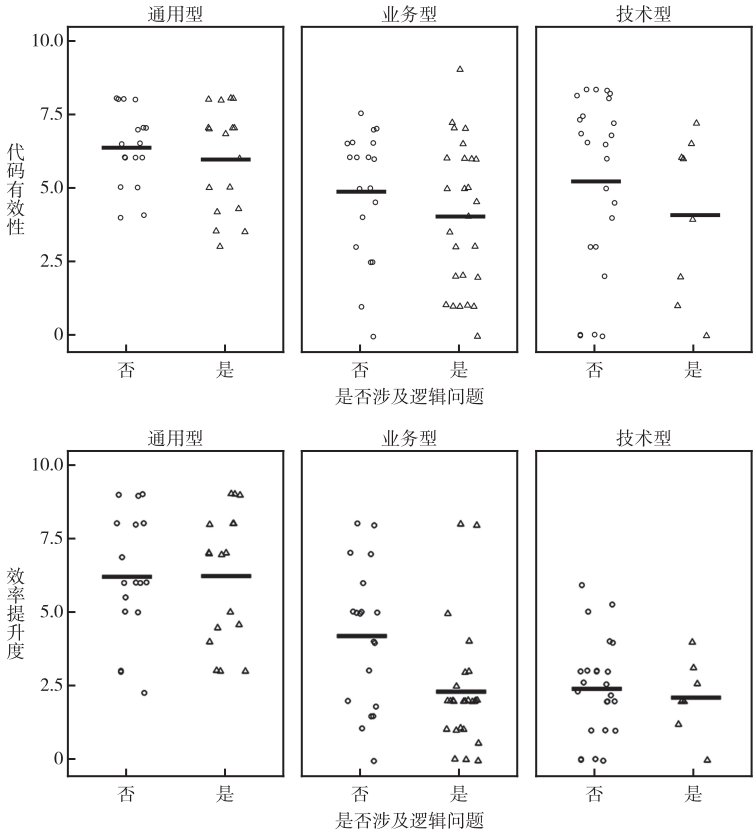

图 11　不同应用场景下逻辑问题对 Copilot 代码有效性和效率提升度的影响

资料来源：根据评分数据绘制得到。

件编程这种对正确性要求极高的工作，通过生成式 AI 技术实现完全自动化面临很大挑战。

表 4　使用 Copilot 自动生成代码需人工修改用例的情况

任务类型	用例数量（个）	自动生成代码需修改用例占比（%）	修改用例中需提升逻辑用例占比（%）
通用型	52	96. 15	66. 00
业务型	27	92. 59	40. 00
技术型	33	87. 88	48. 28
总样本	112	92. 86	54. 81

资料来源：根据评分数据计算得到。

总体上看，在软件开发领域，Copilot 等生成式 AI 工具的作用主要体现在理解需求、解决问题、生成代码等阶段。目前，训练数据、复杂逻辑、隐性知识、人机交互等因素在软件开发工作流程中的各个环节，影响着生成式 AI 工具在实际应用场景中的表现。具体而言，在理解需求阶段，软件开发人员与 Copilot 的交互问答有助于提升 AI 工具的需求理解能力，而复杂逻辑和训练数据的缺失则对 AI 工具的表现形成限制。在解决问题阶段，隐性知识和复杂逻辑是生成式 AI 工具需要克服的制约因素，软件开发人员通过问答交互能够在一定程度上弥补行业和机构知识的限制，而技术类隐性知识更难通过提示词或注释形式输入给 AI 工具。在生成代码阶段，生成式 AI 工具与软件开发人员之间具有更强的交互属性，绝大多数的代码生成过程需要人工参与，以满足编程应用场景下对正确性、逻辑性的高要求（编写的代码有一点逻辑错误就会影响编译和运行）。因此，中短期在软件开发领域引入生成式 AI 工具可以重点从上述因素入手，提升生成式 AI 工具的技术能力和配套条件。

（二）生成式 AI 技术对软件开发工作范式的长期影响展望

尽管生成式 AI 工具 Copilot 在辅助编程方面的实际效果还有很大的提升空间，但它给软件开发的工作范式带来了一些新变化，这种变化有可能重塑 AI 工具与软件工程师之间的辅助关系。在以往自动化工具不断迭代的过程中，软件工程师始终处于主导地位，自动化的目标是将开发人员的注意力转移到计算机不擅长的概念任务上，并在计算机可以提供辅助的那些任务中减少开发人员的错误。然而，生成式 AI 技术似乎带来了更大的不确定性。与传统的软件自动化工具不同，生成式 AI 工具（例如本文所分析的 Copilot）带来的效率提升是基于“双向互动”的。在本研究的一些场景中，软件工程师向工具提出需求（问题），工具给出解决方案——生成所需要的代码或部分代码；在另一些场景中，工具给出的方案并不能直接解决问题，但能够提供启发式的建议，打开软件工程师的思路，或者通过不断地双向互动找到合适的解决方案。这一过程实际上改变了软件自动化的工作范式，从单向的任务自动化转变为团队合作型的效率提升。生成式 AI 工具不只是扮演一个工具的角色，而更像一个“合作伙伴”，软件开发人员通过对工具的提问和指导，不断提高生成式 AI 工具的能力，并对其提升后的能力进行监督，让其更好地提供辅助作用。

基于这种新的工作方式，生成式 AI 在自动生成代码方面表现出比前几代技术更大的潜力。随着生成式 AI 技术的发展和工具能力的提升，软件工程师是否依然能够处于主导地位值得商榷。短期来看，生成式 AI 技术对软件开发范式的影响正在带来两方面的变革：一方面，在软件工程经历的多次范式变化中，软件开发人员始终依赖

高级编程语言生成代码，而此次人工智能技术突破能够支持从问题的自然语言描述中自动合成代码，这将极大地提高软件开发人员的生产力，并可能大大降低软件编程的门槛；另一方面，生成式 AI 技术将加速软件开发职业内部的任务再分配，软件开发人员与生成式 AI 工具的“结对编程”将更加普遍，软件开发工作流程中会有越来越多依赖非常规认知能力的任务被生成式 AI 工具替代。为了保持竞争优势，软件开发人员需要不断提高其专业技能，或者致力于提高那些不容易被技术替代的非常规认知能力。

六　总结与建议

生成式人工智能技术及相关工具的快速发展正在对劳动力市场产生广泛影响。从工作类型和任务来看，生成式 AI 技术的影响范围已经从重复型、程式化的工作扩大到知识型、专业型工作，例如专业写作、翻译、数据分析、广告传媒、软件编程等，同时在一些领域实现了对人类隐性知识和认知能力的自动化。本文选取软件工程行业为研究对象，分析生成式 AI 技术在应用早期阶段对专业型工作的实际影响。本文通过考察一家信息技术服务公司在实际工作场景中使用生成式 AI 工具（Github Copilot）的情况，评估生成式 AI 工具在实际工作场景下解决基本编程和算法问题的能力，并深入分析生成式 AI 工具在不同应用场景下对软件开发工作的辅助效果、局限性及其背后的原因。本文的研究显示，在技术应用的早期阶段，生成式 AI 技术对软件开发工作的影响主要表现出以下特征和趋势。

第一，现阶段生成式 AI 工具对软件开发工作的冲击有限，自动生成代码的质量难以达到预期，对软件工程师的影响主要体现在效率提升，岗位替代风险较低。以 Copilot 为代表的生成式 AI 工具对训练数据和历史代码的依赖较强，尚未实现高质量、完整代码的自动化生成能力。因此，基于经典代码的查询回复类任务更符合现阶段生成式 AI 的技术能力，可以实现较好的提升效应。涉及企业业务流程、行业逻辑、专业知识的原始代码通常属于高价值资产，企业出于保护商业机密的考虑，不会将其作为模型训练数据对外分享。由于缺少相关场景和业务逻辑的训练，生成式 AI 工具目前很难完成业务逻辑复杂或包含专业知识的编程任务。

第二，生成式 AI 工具对通用型场景的编程任务提升效果显著，但对业务型和技术型场景的影响有限。生成式 AI 工具 Copilot 在不同类型场景下的得分差异主要受到以下因素的影响：通用型任务所涉及的代码更为标准化，相关训练数据的可得性更

高，因此模型在充分利用更高质量数据的基础上，能够在通用型任务中取得较高得分；而在业务型和技术型场景中，Copilot 工具的得分较低，主要受到专业知识、机构知识和技术知识的限制，这些知识尚未形成训练数据，导致生成式 AI 工具难以理解任务意图。

第三，现阶段使用生成式 AI 工具辅助软件开发仍面临训练数据、复杂逻辑、隐性知识、人机交互等制约因素，影响着生成式 AI 工具在软件开发工作场景中的表现。具体而言，在理解需求阶段，软件开发人员与 Copilot 的交互问答有助于提升 AI 工具的需求理解能力，而复杂逻辑和训练数据缺失则对 AI 工具的表现形成限制。在解决问题阶段，隐性知识和复杂逻辑是生成式 AI 工具需要克服的制约因素，软件开发人员通过问答交互能够在一定程度上弥补行业和机构知识的限制，而技术类隐性知识更难通过提示词或注释形式输入给 AI 工具。在生成代码阶段，生成式 AI 工具与软件工程师之间具有更强的交互属性，绝大多数的代码生成过程需要人工参与，以满足编程应用场景下对正确性和逻辑性的高要求。

第四，生成式 AI 对软件开发行业的工作范式带来较大冲击，软件工程师与人工智能工具之间的关系、软件开发工作岗位的技能需求很有可能发生较大变化。与传统的软件自动化工具有所不同，生成式 AI 工具使软件开发从单向的任务自动化转变为团队合作型的效率提升。在 A 公司软件开发工作的多个实际用例中，Copilot 都展现出了较强的学习能力及区别于传统 AI 的技术潜力。Copilot 能够理解并学习上下文编程逻辑，并将其作为训练数据，快速提升模型表现；在部分用例中，Copilot 能够通过与从业人员的多轮“提示词”对话，逐渐理解任务意图，提高推荐代码的有效性。这一变化有可能重塑 AI 工具与软件工程师之间的辅助关系，软件开发工作中更多的任务将被生成式 AI 工具取代，推动软件开发职业内部的任务再分配。

总体来看，生成式 AI 技术引领的新一波自动化浪潮确实对知识型、专业型工作产生了重大影响，即使是像软件开发这样依赖高水平认知能力的专业型工作，也受到了较大的冲击。不过，短期内这种冲击尚未显露出消极的倾向，新技术的发展仍主要致力于为软件开发工作提供辅助和赋能作用，未来的经济社会影响将在很大程度上取决于如何管理技术的扩散。鉴于全球范围内软件开发人员实际上仍处于供不应求的状态（Taulli，2023），本文认为短期内生成式 AI 工具将成为解决软件开发人才需求不足的重要手段，其商业化应用也将快速发展。因此，当前应该关注的是如何提升生成式 AI 工具在软件工程领域的性能和表现，以及更好地建立从业人员与人工智能技术进步之间的良性关系。

首先，从生成式 AI 工具本身来看，从通用型到业务型、技术型应用场景的辅助编程效果还有很大的提升空间。未来一方面需要加强相关场景的训练数据覆盖和模型能力的提升，另一方面也应当完善生成式 AI 技术的开源系统建设，进一步推动生成式 AI 软件创新。其次，现阶段生成式 AI 工具的替代倾向主要表现在对工作任务而不是人，因此提高软件开发人员和工具“结对编程”的有效性将是短期内的重要趋势。例如，生成式 AI 工具在完全理解工作任务需求方面仍存在较大障碍，这是软件开发人员可以弥补的地方。同时，软件开发人员还需要对工具提供的解决方案进行检测和评估，修改其错误或非最优方案。最后，软件开发人员需要对技能发展做出长远的规划，从而在人工智能技术进步中保持竞争力。随着生成式 AI 工具在软件工程领域被更广泛地采用，软件开发工作所需的任务类型也将发生变化，留给软件开发人员的往往是更高阶的工作。软件开发人员应当提早规划转型，一是学会如何与工具更密切地配合，二是如何在生成式 AI 工具能力之外的工作任务中提升自己的竞争力，例如软件开发的框架设计和决策等。

本文分析了生成式 AI 工具在赋能软件开发工作过程中面临的局限性和制约因素，这些因素更多是从技术角度的分析。实际应用过程中还有很多来自环境的约束，例如基础设施是否能够支持技术的采用、劳动力与技术的相对成本、数字技能水平、市场动态和人们对技术创新的态度等。这些约束将影响生成式 AI 技术在不同行业、不同领域的扩散速度和方式，并对专业型工作产生不同的影响。职业分布中的一些岗位可能面临更大的替代风险，另一些岗位将受益于技术的赋能作用。因此，政策应该同时兼顾两种发展趋势可能带来的影响，一方面提早应对生成式 AI 技术带来的就业替代风险，另一方面也应当重视生成式 AI 技术在发挥赋能作用时的持续性和公平性，确保在就业增强效应下工作质量的提高。

具体来看，本文提出以下几点建议。一是加强对人工智能技术发展的规划和管理，引导人工智能技术向劳动力辅助或增强方向发展，避免发生大规模的技术性失业。应加强对人工智能技术发展方向的监督管理，使其成为提升劳动生产率的有力工具，培育人机协同的工作模式。二是加快教育和人才培养模式改革，推动传统的标准化、输入式教育以及批量培养的人才发展模式转向“智慧教育”，更加关注个体成长，重视个体创新思维和数字技能的培养，从教育内容和模式、考核标准、人才选拔和评价体系方面进行系统性改革，提升人才对未来技术、产业和社会发展的适应能力。

参考文献：

陈永伟（2023），《超越 ChatGPT：生成式 AI 的机遇、风险与挑战》，《山东大学学报（哲学社会科学版）》第 3 期，第 127－143 页。

徐国庆、蔡金芳、姜蓓佳、李政、杨惠、郑杰（2023），《ChatGPT/生成式人工智能与未来职业教育》，《华东师范大学学报（教育科学版）》第 7 期，第 64－77 页。

郑世林、姚守宇、王春峰（2023），《ChatGPT 新一代人工智能技术发展的经济和社会影响》，《产业经济评论》第 3 期，第 5－21 页。

Acemoglu, Daron & Pascual Restrepo (2019a). Artificial Intelligence, Automation, and Work. In Ajay Agrawal, Joshua Gans & Avi Goldfarb (eds.), *The Economics of Artificial Intelligence: An Agenda*. Chicago: The University of Chicago Press, pp. 197－236.

Acemoglu, Daron & Pascual Restrepo (2019b). Automation and New Tasks: How Technology Displaces and Reinstates Labor. *Journal of Economic Perspectives*, 33 (2), 3－30.

Autor, David (2014). Polanyi's Paradox and the Shape of Employment Growth. *NBER Working Paper*, No. 20485.

Autor, David (2015). Why Are There Still So Many Jobs? The History and Future of Workplace Automation. *Journal of Economic Perspectives*, 29 (3), 3－30.

Autor, David (2022). The Labor Market Impacts of Technological Change: From Unbridled Enthusiasm to Qualified Optimism to Vast Uncertainty. *NBER Working Paper*, No. 30074.

Autor, David & Anna Salomons (2017). Robocalypse Now: Does Productivity Growth Threaten Employment. *Proceedings of the ECB Forum on Central Banking: Investment and Growth in Advanced Economie*, 45－118.

Brynjolfsson, Erik, Danielle Li & Lindsey Raymond (2023). Generative AI at Work. *NBER Working Paper*, No. 31161.

Cockburn, Iain, Rebecca Henderson & Scott Stern (2019). The Impact of Artificial Intelligence on Innovation: An Exploratory Analysis. In Ajay Agrawal, Joshua Gans & Avi Goldfarb (eds.), *The Economics of Artificial Intelligence: An Agenda*. Chicago: The University of Chicago Press, pp. 115－146.

Dakhel, Arghavan, Vahid Majdinasab, Amin Nikanjam, Foutse Khomh, Michel Desmarais & Zhen Ming Jiang (2023). Github Copilot AI Pair Programmer: Asset or Liability?

Journal of Systems and Software, 203, 111734.

Eisfeldt, Andrea, Gregor Schubert & Miao Ben Zhang (2023). Generative AI and Firm Values. *NBER Working Paper*, No. 31222.

Eloundou, Tyna, Sam Manning, Pamela Mishkin & Daniel Rock (2023). GPTs Are GPTs: An Early Look at the Labor Market Impact Potential of Large Language Models. *arXiv Working Paper*, No. 2303. 10130.

Frank, Morgan, David Autor, James Bessen, Erik Brynjolfsson, Manuel Cebrian, David Deming, Maryann Feldman, Matthew Groh, José Lobo, Esteban Moro, Dashun Wang, Hyejin Youn & Iyad Rahwan (2019). Toward Understanding the Impact of Artificial Intelligence on Labor. *Proceedings of the National Academy of Sciences*, 116 (14), 6531 – 6539.

Goldfarb, Avi, Bledi Taska & Florenta Teodoridis (2023). Could Machine Learning Be a General Purpose Technology? A Comparison of Emerging Technologies Using Data from Online Job Postings. *Research Policy*, 52 (1), 104653.

Kabir, Samia, David Udo-Imeh, Bonan Kou, Tianyi Zhang (2023). Is Stack Overflow Obsolete? An Empirical Study of the Characteristics of ChatGPT Answers to Stack Overflow Questions. *arXiv Working Paper*, No. 2308. 02312.

Korinek, Anton (2023). Language Models and Cognitive Automation for Economic Research. *NBER Working Paper*, No. 30957.

Li, Yujia, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Lago & Others (2022). Competition-Level Code Generation with Alphacode. *Science*, 378 (6624), 1092 – 1097.

Lou, Bowen, Hongshen Sun & Tianshu Sun (2023). GPTs and Labor Markets in the Developing Economy: Evidence from China. *SSRN Working Paper*, No. 4426461.

Nguyen, Nhan & Sarah Nadi (2022). An Empirical Evaluation of Github Copilot's Code Suggestions. *Proceedings of the 19th International Conference on Mining Software Repositories*, 1 – 5.

Noy, Shakked & Whitney Zhang (2023). Experimental Evidence on the Productivity Effects of Generative Artificial Intelligence. *Science*, 381 (6654), 187 – 192.

Ozkaya, Ipek (2022). A Paradigm Shift in Automating Software Engineering Tasks: Bots. *IEEE Software*, 39 (5), 4 – 8.

Peng, Sida, Eirini Kalliamvakou, Peter Cihon & Mert Demirer (2023). The Impact of AI on Developer Productivity: Evidence from Github Copilot. *arXiv Working Paper*, No. 2302. 06590.

Russo, Daniel (2023). Navigating the Complexity of Generative AI Adoption in Software Engineering. *arXiv Working Paper*, No. 2307. 06081.

Surameery, Nigar & Mohammed Shakor (2023). Use Chat GPT to Solve Programming Bugs. *International Journal of Information Technology and Computer Engineering*, 3 (1), 17 – 22.

Taddy, Matt (2019). The Technological Elements of Artificial Intelligence. In Ajay Agrawal, Joshua Gans & Avi Goldfarb (eds.), *The Economics of Artificial Intelligence: An Agenda*. Chicago: The University of Chicago Press, pp. 61 – 87.

Taulli, Tom (2023). Auto Code Generation: How Generative AI Will Revolutionize Development. In Tom Taulli (ed.), *Generative AI: How ChatGPT and Other AI Tools Will Revolutionize Business*. Berkeley: Apress, pp. 127 – 143.

Tian, Haoye, Weiqi Lu, Tsz Li, Xunzhu Tang, Shing-Chi Cheung, Jacques Klein & Tegawendé Bissyandé (2023). Is ChatGPT the Ultimate Programming Assistant - How Far Is It? *arXiv Working Paper*, No. 2304. 11938.

Tolan, Songül, Annarosa Pesole, Fernando Martínez-Plumed, Enrique Fernúndez-Macías, José Hernúndez-Orallo & Emilia Gómez (2021). Measuring the Occupational Impact of AI: Tasks, Cognitive Abilities and AI Benchmarks. *Journal of Artificial Intelligence Research*, 71, 191 – 236.

Zarifhonarvar, Ali (2024). Economics of ChatGPT: A Labor Market View on the Occupational Impact of Artificial Intelligence. *Journal of Electronic Business & Digital Economics*, 3 (2), 100 – 116.

How Does Generative AI Affect Professional Work? Early Evidence from the Software Engineering Industry

Ma Yefeng[1,2], Chen Nan[1,2] & Cui Xuebin[3]

(Institute of Quantitative & Technological Economics, Chinese Academy of Social Sciences[1]; Laboratory for Economic Big Data and Policy Evaluation, Chinese Academy of Social Sciences[2]; School of Digital Economy and Management, Nanjing University[3])

Abstract: The development of generative artificial intelligence (AI) technology has significantly narrowed the gap between AI and human cognitive abilities, profoundly affecting knowledge-based and professional work. This study examines the software engineering industry to analyze generative AI's early-stage impacts and mechanisms on professional tasks. Our findings indicate that, at the current stage, generative AI has a limited impact on professional work, primarily enhancing efficiency in software development. There remains a significant gap before generative AI can broadly replace human roles. AI tools show the highest efficiency improvements in general business scenarios but are less effective in industry-specific applications. Notably, when application scenarios involve advanced technical capabilities such as cloud computing and machine learning, the performance of generative AI tools is suboptimal. The limitations arise from several constraints in using generative AI tools for software development, including issues with training data, complex logic, tacit knowledge, and human-computer interaction. These factors limit the technology's effectiveness in understanding problems, solving them, and generating code. In the long term, generative AI will shift software development from one-way task automation to a human-machine collaboration model, thereby enhancing efficiency through two-way interaction. The insights from this study are significant for establishing a positive relationship between professional workers and future AI technology advancements.

Keywords: generative AI, professional work, software development, application scenarios

JEL Classification: J24, J21, O32

(责任编辑：封永刚)