# 计算机软件工程中人工智能代码辅助工具研究

秦波涛
西安翻译学院，陕西 西安 710105
DOI:10.61369/RSTD.2026030009

**摘　　要：** 人工智能代码辅助工具在软件工程中的部署日益广泛，其功能覆盖代码补全、缺陷检测与重构建议等多个维度。工具输出质量受训练数据分布与上下文解析深度共同影响，开发者对建议的采纳率随任务复杂度提高而下降。实践反馈显示，工具介入后开发周期有所缩短，但程序员对代码逻辑的主动思考频率相应降低。这种效率与认知投入之间的取舍构成工具应用的核心讨论点。

**关 键 词：** 人工智能代码辅助工具；软件工程；功能边界；实践效应；人机协作

## Research on Artificial Intelligence Code-Assistance Tools in Computer Software Engineering

Qin Botao
Xi'an FanYi University, Xi'an, Shaanxi 710105

**Abstract:** Artificial intelligence code-assistance tools are being increasingly deployed in software engineering, offering functionalities ranging from code completion to defect detection and refactoring suggestions. The quality of tool outputs is influenced by both the distribution of training data and the depth of context analysis; developers 'adoption rate of suggestions decreases with increasing task complexity. Practical experience shows that while tool integration shortens development cycles, programmers' proactive reflection on code logic frequency correspondingly declines. This trade-off between efficiency gains and cognitive effort represents the central issue in evaluating tool adoption.

**Keywords:** AI code-assistance tools; software engineering; functional boundaries; practical effects; human-computer collaboration

## 引言

软件项目规模的扩张与交付周期的压缩使得开发团队对自动化辅助手段的需求持续增强，人工智能代码辅助工具在此背景下获得了广泛的实践关注。当前工具以大规模开源代码为训练基础，能够在语法规范与程序结构层面为开发者提供即时参考，但其输出建议与具体业务场景之间的匹配精度并不稳定。开发者在借助这类工具完成编码任务时需持续保持自身判断力，对工具建议的无条件接受可能削弱其对系统整体架构的掌控能力。关于工具在何种条件下能够有效发挥作用、其使用边界又该如何划定，仍有待深入探讨。

本文从工具能力、使用后果与约束条件三个维度出发，构建一个递进式的分析框架。该框架先对代码生产、质量保障与软件演化三种场景下的工具功能进行分类考察，再转入开发流程中效率改进与认知损耗之间的实际权衡，最后从技术性能、组织管理与制度规范三个方向厘清工具的适用边界。这一分析路径力求呈现工具与开发者在具体实践中的互动关系，为工程团队制定合理的使用策略提供参照。

## 一、概念辨析与研究框架

### （一）人工智能代码辅助工具的概念界定

人工智能代码辅助工具在软件工程领域指向一类嵌入开发环境并基于算法模型提供编程支持的软件系统。该类工具区别于常规代码编辑器的核心特征在于其对程序上下文具备有限程度的语义理解，而非仅执行关键词匹配或模板填充操作。工具的训练依赖于海量开源代码仓库，从中提取语法规则与高频编程模式，生成结果在形式层面与人工代码高度相似。但工具对业务逻辑缺乏真实认知，其建议往往局限于局部语法层面，开发者需依据项目整体需求对输出进行甄别与调整。概念界定应明确工具的辅助定位，开发者始终保有编程决策的主导权，工具仅在编码实现环节

作者简介：秦波涛（1985-），男，汉族，陕西佳县人，高级工程师，硕士，软件工程、AI 智能、芯片开发。

Copyright © This Work is Licensed under A Commons Attribution-Non Commercial 4.0 International License. 

提供效率层面的支持。

**（二）人工智能代码辅助工具的核心技术路径**

人工智能代码辅助工具的实现基于大规模预训练语言模型的技术架构，其技术路线以对开源代码与自然语言文本的联合表征学习为核心。模型采用深层网络结构捕捉程序中的上下文依赖关系，从而根据前序代码推断后续内容。预训练阶段通过自监督学习从无标注数据中获取通用编程知识，微调阶段则利用特定任务数据调整模型参数以适配不同场景。部分工具引入检索增强机制以缓解纯参数化记忆的局限，在生成时从外部知识库提取相似代码片段并辅助输出，以此提升结果的准确性与可用性[1]。

**（三）本文的分析框架：从功能到影响再到限度**

本文构建的分析框架依照工具特性、实践后果与适用边界三个层次逐层递进。框架以功能类型分析为起点，分别考察代码补全、质量审查和重构迁移等场景下的工具表现，为后续效应评估建立分类基础。第二层面聚焦工具嵌入开发流程后引发的实践效应，从效率提升与认知损耗两个方向考察工具介入的实际得失。框架最终落脚于工具应用的约束条件，围绕技术可靠性、开发者能力维持与制度合规性三个维度划定工具的适用边界。三层结构形成从描述到评估再到反思的完整分析链条，各层面的讨论结论为后续论证提供支撑。

## 二、人工智能代码辅助工具的功能类型与技术实现

**（一）面向代码生产的功能：补全与生成**

代码补全与代码生成构成了人工智能辅助编码在代码生产层面的两项基本功能，二者在操作粒度与用户介入程度上呈现出显著分化。补全功能着眼于行内或行末的即时预测，模型依据光标前的代码上下文推断随后的 token 序列，输出范围通常限制在变量名、函数参数或单条语句之内，开发者的输入行为始终主导着补全的方向与推进节奏。生成功能则面向更高层级的开发诉求，开发者以自然语言描述函数目的或业务规则，模型据此产出完整的代码块乃至多个函数组成的逻辑单元，其生成过程对输入描述的完整性与清晰度高度敏感。描述模糊时输出结果可能出现结构偏差或逻辑缺陷，开发者需对生成内容进行功能性验证与结构性调整。两类功能在模型推理层面共享参数更新机制，但生成的输出长度与不确定性远高于补全，对上下文窗口的利用策略与解码参数配置提出了差异化要求。

**（二）面向代码质量保障的功能：审查与检测**

代码审查与缺陷检测是工具保障软件质量的两类主要手段。审查功能通常作用于代码提交阶段，模型通过比对历史版本与当前修改之间的差异来识别语法错误、风格偏离及潜在的逻辑漏洞，其输出多以审查意见的形式呈现给开发者。缺陷检测则针对静态代码库中的既存问题展开独立扫描，工具在全量代码中定位可能的空指针引用、资源泄漏或安全薄弱环节，其分析范围不依赖于版本变更事件。审查机制更注重变更上下文中捕捉到的差异信号，检测机制则需建立对代码结构与数据流向的全局性理解，两类功能在模型训练目标的设定上各有侧重。工具输出的问题标记仍需开发者逐一确认，误报与漏报的权衡在实际使用中始终难以达到完美平衡[2]。

**（三）面向软件演化的功能：重构与迁移**

重构与迁移是支撑软件长期演化的两项关键操作。重构工作集中于在保持外部行为不变的前提下调整代码内部结构，工具需识别可提取的重复逻辑、过长函数或过度耦合的类依赖关系，并给出具体的结构优化方案。迁移任务则涉及代码在不同编程语言或框架之间的转换，工具必须建立源语言与目标语言之间的语法映射关系，同时关照运行时环境的对应配置。重构方案对功能等价性的要求极为严格，任何自动化调整都需经过充分的回归验证方可纳入主分支。迁移过程中目标平台 API 语义的差异常常导致生成代码需要人工补充初始化逻辑或异常处理分支，纯粹的工具输出很少能够直接部署运行。两类功能共同面对的一个深层局限在于，工具对项目历史演进脉络缺乏实质性理解，其建议仅基于当前代码快照的静态分析，难以与团队既有的编码惯例和长期架构规划保持协调。

**（四）功能有效性的技术边界**

工具输出的可靠性在很大程度上取决于模型训练数据与目标应用场景之间的契合程度。开源代码主导的训练语料难以覆盖企业内部框架、特定业务规则或遗留系统中独有的代码模式，工具在陌生语境下的建议准确度难免下降。开发者对工具输出的过度信任可能掩盖潜在隐患，错误的补全或审查结论在代码库中长期积累后将显著推高后期的修复成本。模型对代码语义的处理本质上仍停留在统计关联层面，程序的实际运行状态与业务逻辑的深层关联并不在模型的感知范围之内，任何自动生成或审查的结果都需经过编译验证与功能测试方可投入使用。这些因素共同勾勒出工具能力的基本轮廓，自动化辅助的价值始终需要置于人工审核的框架内加以衡量，功能有效性的最终判定权无法让渡给算法本身[3]。

## 三、人工智能代码辅助工具嵌入软件工程实践的双重效应

**（一）积极效应：效率提升、质量改进与知识外显**

工具承担了语法纠错与样板代码生成等重复性事务，开发者得以将注意力转向模块划分与接口定义等设计层面的事务。审查环节中工具捕捉到的人工易漏项，诸如异常分支与边界条件，其提前暴露减少了后期测试阶段的返工成本。开发者频繁接触工具提供的多种实现路径，这种交互过程促成了编程经验的横向迁移，团队新成员借助补全与检索功能可以较快理解既有代码的命名惯例与结构特征。开发活动逐步从语法层面的机械操作向设计层面的主动思考偏移，开发者花费更多时间进行方案比较与逻辑验证，而不再仅仅着眼于代码能否通过编译。

**（二）消极效应：认知依赖、技能稀释与责任模糊**

开发者频繁采纳工具建议后，遇到不熟悉的问题时倾向直接依赖工具输出，独立检索文档与手动调试的行为相应减少。新手开发者在补全功能的辅助下绕过了查阅文档与手动调试的传统学

 | Copyright © This Work is Licensed under A Commons Attribution-Non Commercial 4.0 International License.

习路径，对语言特性和底层机制的掌握深度明显不足，这种技能基础的弱化在长期使用中难以逆转。团队中经验差异导致工具依赖程度分化，资深开发者能够识别建议中的逻辑偏差，初级开发者则难以察觉复杂场景下的输出缺陷，错误代码的传播风险随之上升。工具生成的代码源于大规模训练语料，其原始作者的追溯缺乏有效手段，审查阶段责任归属的模糊性增加了法律层面的不确定性。

**（三）效应差异的关键调节因素**

工具效应的正负方向取决于任务复杂度、开发者经验与组织规范三者之间的匹配程度。简单重复的编码场景下工具建议精准且直接可用，而涉及特殊算法逻辑或领域特定规则时输出偏差率显著上升，任务本身的复杂度决定了工具推理能力是否足以胜任。资深开发者倾向于将工具输出视为参考意见，遇到可疑建议能够及时识别并修正，新手则难以判断输出合理性，倾向于不加批判地采纳工具结论。团队内部的代码审查规范与工具使用指南同样影响工具的实际效用，完善的验证机制能够有效控制错误建议在项目内的扩散范围，而缺乏约束的环境则可能放大工具的负面效应。

## 四、人工智能代码辅助工具应用的约束条件与应对策略

多项实证研究揭示了 AI 代码辅助工具在实际应用中的复杂效应。邢万勇等（2025）的研究表明，AI 编程助手能有效提升编程效率与代码质量。然而，一项针对 EDA 软件开发教学的对照实验发现，AI 工具虽能使任务质量提升 16.9% ~ 21.2%、完成时间减少 47.3% ~ 71%，但其效益并非没有代价。研究明确指出，AI 工具存在约 20-22 次交互的“最佳使用区间”，超出该区间的过度依赖会显著削弱开发者的独立解决问题能力[4]。工具的效益受制于多重约束，需从以下层面制定针对性策略：

**（一）技术层面的准确性与可靠性约束**

模型训练所依赖的开源代码语料与企业内部框架、遗留系统及特定行业规范之间存在显著偏差，工具在陌生上下文中的输出准确度难以保证。模型对程序语义的捕获本质上属于统计关联而非逻辑推理，代码在复杂分支或并发条件下的实际运行状态不在其感知范围之内。为此，需将工具输出定位为参考素材而非最终结论，通过编译验证、单元测试与同行评审逐层过滤。核心业务模块还须在工具输出基础上补充压力测试与边界条件验证，以降低模型推理偏差引入缺陷的概率。团队应同步建立工具输出问题的反馈机制，将日常使用中的错误类型归类整理，使开发者在相似场景下对工具建议保持合理怀疑。

**（二）组织层面的开发者技能退化与管理对策**

工具频繁介入可能改变开发者的编码技能结构。部署初期即应划定工具的使用边界，核心算法与关键业务逻辑不得依赖工具生成，其他场景下工具输出可作为参考但不得直接复制。代码审查环节须强化开发者对工具输出的解释义务，提交者应在审查说明中标注生成来源并论证其合理性，审查者须将此类内容列为重点检查对象。定期组织脱离工具环境的编程练习，有助于维持开发者对语言特性与底层机制的掌握程度。团队还宜建立工具输出缺陷台账，定期统计高频错误类型，据此动态调整工具的使用范围与审查策略[5]。

**（三）制度层面的知识产权归属与合规风险**

工具生成代码的知识产权归属在现行法律框架内尚无明确界定，开源代码纳入训练语料后，原始许可证中的署名要求与传播约束在模型输出阶段难以追溯。企业应在工具选型阶段审查训练数据的构成与来源，优先采用基于合规语料训练的模型产品[6]。开发过程中须建立工具输出代码的归档与溯源机制，记录每次生成操作的时间、输入条件及对应输出，为后续可能的侵权纠纷保留使用轨迹。内部应制定 AI 工具使用规范，列明允许与禁止工具介入的具体场景。涉及核心业务逻辑或与开源组件交互的代码模块应完全绕开工具生成，由开发者独立编写并留存设计文档作为权利归属证据。

## 五、结语

代码辅助工具的智能化程度不断提升，但其实际效用取决于开发者的批判性使用方式。工具输出并非总是可靠，需结合具体项目背景进行验证。实践中应建立工具使用指南与代码审查制度，以平衡效率提升与技能维护。制度层面需明确生成代码的权责归属，避免法律风险。将工具纳入规范化开发流程，有助于实现短期提速与长期能力积累的协调。

*参考文献*


[1] 张钦，郑尚，邹海涛，等 . 一种融合注意力机制的形变 LSTM 智能代码补全方法 [J]. 小型微型计算机系统，2024,45(2):498-504.

[2] 徐一然，周宇 . 基于提示学习的轻量化代码生成方法 [J]. 计算机科学，2024,51(6):61-67.

[3] 邢万勇，廖伟军，杨智帆 . 基于 AI 编程助手的软件研发技术效能提升实践 [J]. 数字技术与应用，2025,43(1):28-30.

[4] 邢颖 . 基于可解释性人工智能的软件工程技术方法综述 [J]. 计算机科学，2023,50(5):9.

[5] 张鹏巍 . 人工智能算法在软件工程缺陷自动检测中的应用 [J]. 电子技术，2025(5):422-423.

[6] 那德尔・热西提 . 基于人工智能的软件测试自动化策略研究 [J]. 信息与电脑，2025(6):66-68.


Copyright © This Work is Licensed under A Commons Attribution-Non Commercial 4.0 International License. 