

多代码智能体研究与应用综述

张旭龙¹, 鲍习坤^{1,2}, 瞿晓阳¹, 倪晓俊³, 田晖⁴, 王健宗¹

- 平安科技(深圳)有限公司, 广东 深圳 518000;
- 中国科学技术大学, 安徽 合肥 230000;
- 中国科学院计算技术研究所, 北京 100190;
- 华侨大学, 福建 厦门 361021

摘要

随着大语言模型的快速发展, 基于其构建的代码智能体已成为自动化编程与软件工程智能化的重要方向。单智能体能够在代码生成、调试与测试等任务中展现出较强的自主性, 但在面对复杂、多阶段的软件开发流程时, 仍受限于上下文容量、任务分解与长期记忆等因素, 难以实现全流程协同与持续优化。为克服这一瓶颈, 研究者提出了多代码智能体体系, 通过引入角色分工、共享记忆与群体反思机制, 使多个智能体在同一开发任务中实现信息共享与协同决策, 从而显著提升代码生成的可靠性与工程可扩展性。系统梳理了从单智能体到多智能体的演化脉络, 总结了多代码智能体在任务规划、工具集成、记忆管理与协作优化方面的主要方法与代表性工作, 并分析了其在真实软件工程中的典型应用。最后, 讨论了当前多智能体体系在上下文一致性、协作调度与安全性方面面临的挑战, 展望了面向自我进化与工业落地的未来发展方向。

关键词

大语言模型; 代码智能体; 多智能体协作; 代码生成; 任务分解; 自我进化

中图分类号: TP18

文献标志码: A

doi: 10.11959/j.issn.2096-0271.2026062

A review of code agents based on large language models

Zhang Xulong¹, Bao Xikun^{1,2}, Qu Xiaoyang¹, Ni Xiaojun³, Tian Hui⁴, Wang Jianzong¹

- Ping An Technology (Shenzhen) Co., Ltd., Shenzhen 518000, China;
- University of Science and Technology of China, Hefei 230000, China;
- Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China;
- Huaqiao University, Xiamen 361021, China

Abstract

With the rapid advancement of large language models (LLMs), code agents have emerged as a key paradigm for intelligent and automated software development. While single-agent systems demonstrate strong capabilities in code generation, debugging, and testing, they struggle with complex, multi-stage software engineering workflows due to limitations in context length, task decomposition, and long-term memory. To overcome these bottlenecks, researchers have introduced multi-code agent systems, in which multiple agents with specialized roles collaborate through shared memory, role division, and collective reflection. This architecture enables more robust reasoning, coordinated planning, and scalable code generation in real-world projects. This paper provides a comprehensive review of the evolution from

single-agent to multi-agent frameworks, analyzing core advancements in task planning, tool integration, memory management, and collaborative optimization. It further summarizes representative studies and application scenarios in software development automation. Finally, the paper discusses ongoing challenges-such as maintaining contextual consistency, optimizing multi-agent coordination, and ensuring code safety-and explores future directions toward self-evolving and industry-level intelligent programming systems.

Key words

large language model, code agent, multi-agent collaboration, code generation, task decomposition, self-evolution

0 引言

人工智能与软件工程的深度融合，正在推动自动化编程从辅助式工具走向自主式智能体系统。大语言模型凭借其强大的自然语言理解与逻辑推理能力，成为代码生成的核心驱动力^[1-5]。以 GPT、Codex、CodeLlama 等模型为代表的研究，使机器能够根据自然语言描述自动生成结构化、可执行的代码，从而在程序设计、调试与测试中展现出前所未有的能力。

然而，随着任务复杂度的提升，单一模型难以应对完整软件开发流程的多阶段特性。开发任务通常包含需求分析、架构设计、模块实现、测试与维护等多个环节，每个阶段都需要不同的知识结构与上下文理解能力。单智能体虽在局部任务中表现优异，但其有限的上下文窗口、单向的推理链条与缺乏持续学习的特性，导致其在复杂系统开发中难以维持长期一致性与全局逻辑。

受人类协作式开发流程启发，研究者开始探索多智能体系统（multi-agent system, MAS），通过引入角色分工与任务协同，使多个智能体在同一任务空间中共享信息、互补能力并形成群体智能。多代码智能体系统正是这一思路在自动化编程领域的体现。与单智能体相比，多代码

智能体系统能够在复杂项目中实现并行处理、动态规划与自我进化，从而显著提升开发效率与代码质量。本文的目标在于梳理该方向的核心技术脉络，分析代表性研究与应用案例，并总结其未来发展趋势。

1 代码智能体理论方法基础

1.1 代码智能体的定义与核心特征

代码智能体是一种基于人工智能技术的自主系统，能够在整个软件开发生命周期中执行自动化任务。代码智能体不仅仅是一个被动工具，用于生成代码片段或完成单一任务，其核心特点在于具备自主规划、决策和执行的能力^[6-7]。通过结合预训练的大语言模型（LLM）及其微调机制，代码智能体能够在面对复杂的开发任务时，自主进行任务分解、跨模块协作、错误诊断和自我修正，从而显著提升开发效率和代码质量。

代码智能体的首要特征在于其自主性，即能够在无人工干预的情况下，自动管理开发任务的全过程。不同于传统的代码补全工具，代码智能体是一个在复杂开发环境中自我驱动的智能系统。其次，代码智能体的任务范围远超传统代码生成模型，能够涵盖软件开发的各个环节。除了生成简单的代码片段，代码智能体还能够参与

需求收集、项目架构设计、代码实现、持续集成与系统调试等多个重要阶段。最后，代码智能体相关研究不仅关注算法精度的优化，还面临系统可靠性提升、复杂开发流程管理和多工具及系统的有效集成等工程实践中的挑战。

1.2 代码智能体的演化历程

代码智能体的发展经历了从早期简单的代码生成工具到现今复杂多任务自动化系统的逐步演变。最初，代码生成方法主要依赖于程序合成或规则驱动的系统^[8]，这些方法的优点在于生成的代码具有严格的正确性保证（如形式验证），适用于安全关键系统（如航空软件）；缺点是过度依赖形式化输入描述，无法处理自然语言的模糊性，适用条件限于结构化任务。

随着深度学习的发展，研究者开始采用数据驱动的方法，通过训练大规模的代码库和自然语言数据，提升模型的泛化能力^[9-10]。早期的深度学习模型主要采用序列到序列的学习方法，通过逐步预测目标代码的每一个字符或标记来完成代码生成。这些方法在一定程度上缓解了传统代码生成方法对严格任务描述的依赖，但仍无法有效避免生成代码的功能不完整或错误问题。

LLM 的崛起为代码生成技术带来了突破性进展。LLM 通过海量开源代码数据的预训练，深入理解编程语言的语法和语义，从而生成高质量的代码。这些模型不仅能处理基本的代码生成任务，还能应对更复杂的任务，如算法设计、系统架构规划和跨文件代码生成等。LLM 在代码生成方面表现出色^[11-13]，优势在于零样本泛化强，适用于多语言代码生成，劣势是幻觉问题突出，在长序列任务中上下文丢失严重，仅适于辅助而非自主开发。

作为一种创新的系统架构，代码智能体结合 LLM 与其他智能技术，成功弥补了 LLM 在动态开发环境中的不足。它通过自主规划、决策、工具使用和反馈调整，使代码智能体不仅能够自动生成代码，还能够开发过程中进行自我调试与优化，从而实现从代码生成到项目管理的全自动化。

1.3 单智能体的能力与局限性

代码生成的核心是自回归预测：给定上下文 $x_{< t}$ ，模型预测下一个 token x_t 的概率 $P(x_t|x_{< t}) = \prod_{i=1}^t P(x_i|x_{< i})$ 。常用框架 ReAct 交替进行“思-行”循环（Thought $\rightarrow$ Action $\rightarrow$ Observation $\rightarrow$ Reflection），实现工具调用与自我修正。规划、工具集成、记忆管理、反思这 4 种基础能力源于 ReAct 的“思-行-观-反”循环，该框架形式化为 Agent = (π , τ , M)，其中 π 为规划策略（policy）， τ 为工具接口（tool），M 为记忆模块（memory）。基于此，规划定义为任务分解函数 Decompose(T) $\rightarrow$ {Sub T_i }，工具集成定义为 Action(a) = ToolCall(env, a)，记忆管理定义为 Update(M, obs) = Embed(obs) + Retrieve(M, query)，反思定义为 Critique(output) $\rightarrow$ Feedback。

单智能体系统是多智能体发展的起点，其目标是构建具备自主规划、代码生成、错误诊断与自我修复能力的独立模型。这类智能体通过结合大语言模型与工具接口，自动完成从代码合成到执行验证的完整过程。典型工作包括工具增强的代码生成（如 ToolCoder^[14]、CodeAgent^[15]）、基于规划的代码生成（如 CodeTree^[16]）以及自我改进框架（如 SelfRefine^[17]、Self-Debug^[18]、Self-Repair^[19]）等。

单智能体的核心优势在于其自主性和

可解释性。如图1所示,规划能力形式化为 ReAct 循环,工具调用对应 API。借助显式规划、工具调用和反思机制,单智能体可以独立完成小规模项目开发,并在一定程度上模拟人类程序员的思维过程。自反馈机制的引入使其能够通过执行结果不断修正生成策略,从而提升输出的逻辑性与稳定性。

然而,单智能体在扩展性与协作性方面仍存在显著不足。首先,其决策路径是封闭的,无法动态适应外部环境变化;其次,模型缺乏角色分化与并行处理能力,在大型项目中表现出计算瓶颈与上下文漂移;最后,单智能体的反思过程多依赖文本自评,缺乏跨任务的知识迁移与共享机制。这些限制使单智能体难以胜任完整的软件工程任务,也成为多智能体系统兴起的根本动力。

2 多代码智能体系统

多代码智能体系统可形式化为 MAS=

(A, E, C, M_s) , 其中 $A = a_1, \dots, a_n$ 为智能体集合, E 为环境(包括代码库), C 为通信协议(如共享 API), M_s 为共享记忆。协作过程: $\forall a_i \in A, \text{Execute}(a_i) = \text{Plan}(T_i) \rightarrow \text{Interact}(C, M_s) \rightarrow \text{Reflect}(\text{Feedback})$ 。

多代码智能体系统的核心思想在于将复杂任务分解为若干子任务,由具备不同角色与能力的智能体分别负责,从而形成一个可协作、可自组织的群体系统。每个智能体既能独立推理与执行,又能通过通信机制共享中间结果与全局知识,实现任务层级间的协同。

与单智能体不同,多智能体系统强调异构性与自治性共存。系统中的智能体可以使用不同模型结构或工具接口,以任务为驱动动态组成协作团队。这种架构极大地增强了系统的适应性与可扩展性,使其能够在不确定环境中完成复杂的软件开发任务。

此外,多智能体体系在认知机制上引入了共享记忆与群体反思两项关键能力。共享记忆为智能体提供了跨任务的知识通道,使其能够在长期项目中保持上下文连

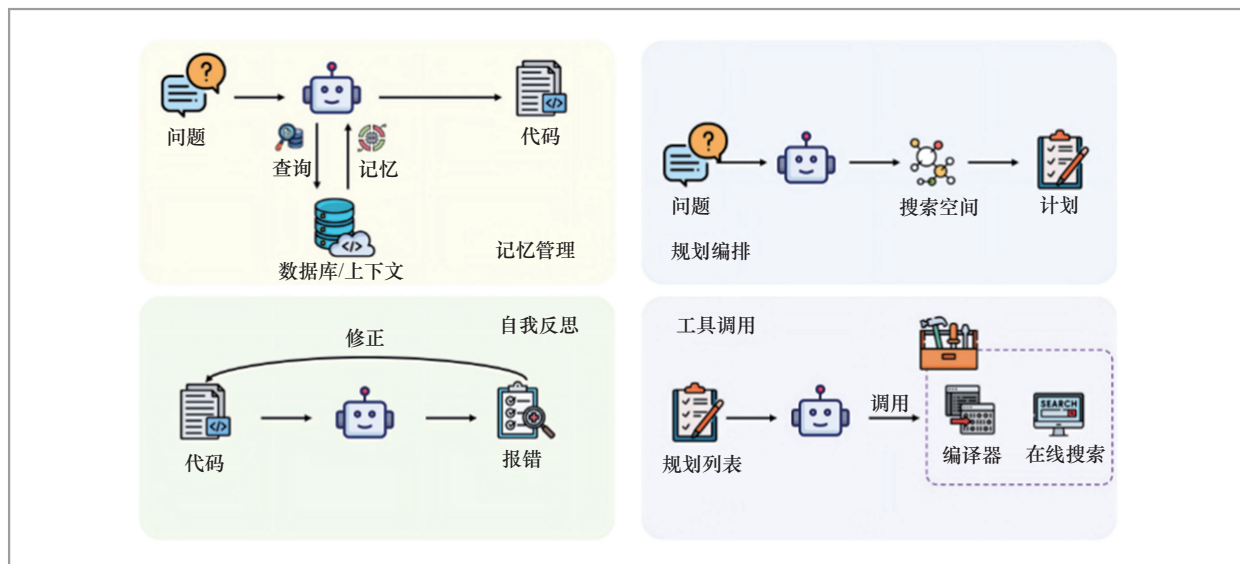


图1 代码智能体基础能力

续性；群体反思机制则通过集体评估与修订过程，实现多智能体间的共识形成与策略演化。这种结构已成为智能代码生成迈向自治与协同的重要基础。**表 1**总结了近期多代码智能体的相关工作。

2.1 角色分工与组织结构

在多智能体代码生成系统中，角色分工与组织结构是实现高效运作的核心基础。不同智能体根据任务需求与系统架构被分配特定的角色，这种角色化的组织方式不仅能够优化任务执行流程，还能促进智能体之间的协同合作。如图 2 所示，随着任务复杂度和协作需求的增加，多智能体系统逐渐演化出多样化的组织模式，包括流水线式分工机制、分层规划与执行机制、自我协商优化机制以及自我进化机制等。

流水线式分工机制将任务划分为若干阶段，使每个智能体专注于特定环节，形成清晰的执行链。该模式下，任务之间具有严格的依赖关系，必须按序完成。较早的代表性工作 Self-Collaboration[20]借鉴软件工程中的瀑布模型，将代码生成过程划分为需求分析、编码实现与测试 3 个阶

表 1 多代码智能体相关工作总结			
工作	组织结构	核心机制	协作模式
Self-Collaboration[20]	流水线式	共享记忆	顺序执行
AgentCoder[21]	流水线式	任务分解	顺序执行
CodePori[22]	流水线式	任务分解	并行处理
MAGIS[23]	流水线式	任务分解	顺序执行
Lingma SWE-GPT[34]	流水线式	任务分解	顺序执行
PairCoder[24]	层级式	任务分解	迭代优化
FlowGen[25]	层级式	任务分解	顺序执行
L2MAC[32]	层级式	共享记忆	顺序执行
Cogito[33]	层级式	共享记忆	迭代优化
MapCoder[27]	协商式	群体反思	迭代优化
AutoSafeCoder[28]	协商式	群体反思	迭代优化
QualityFlow[29]	协商式	群体反思	迭代优化
SyncMind[35]	协商式	群体反思	动态调整
SEW[30]	进化式	自我进化	动态调整
EvoMAC[31]	进化式	自我进化	动态调整

段，每个智能体在相应阶段完成独立任务后将结果交由下一阶段的智能体处理。其优点是结构清晰、便于调试，缺点包括阶段间依赖强，易受早期错误传播影响，适用于线性任务（如简单 API 开发），不宜用

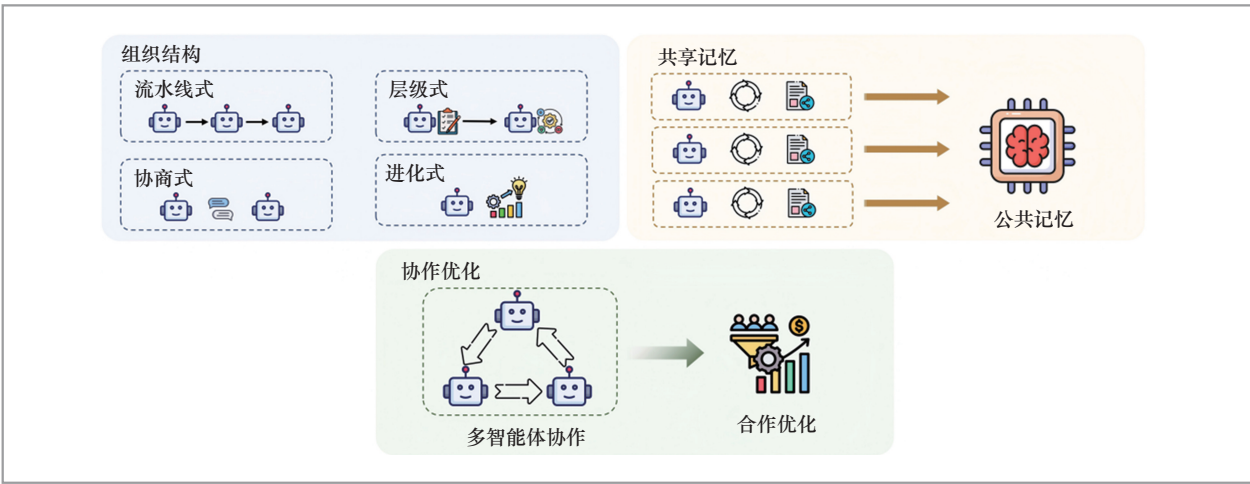


图2 多代码智能体框架

于迭代频繁的敏捷项目。在这一思路基础上，AgentCoder^[21]构建了由“程序员”“测试设计师”和“测试执行者”组成的三阶段流水线系统，通过结构化的任务分解与测试反馈显著提升了代码的正确性与可执行性。CodePori^[22]在此基础上进一步扩展，引入经理、开发者、终结者和验证者等角色：经理负责解析需求并分解任务，多个开发者并行完成模块化实现，终结者整合代码，最终由验证者执行集成测试，以确保整体功能的正确性与完整性。更进一步，MAGIS^[23]提出了一个更复杂的流水线系统，涵盖项目经理、维护人员、开发者与质量保障人员等多种角色，能够处理GitHub平台中的问题管理与修复任务，展示了流水线分工在真实开发场景中的可行性与实用性。

分层规划与执行机制强调层级化的任务组织，其中高层智能体负责整体任务规划与方向制定，低层智能体则执行具体实现。PairCoder^[24]是这一机制的典型代表，它模拟协作式编程模式，引入导航员与驱动者两个角色：导航员负责制定解决方案并指导操作步骤，驱动者则根据指导完成代码生成、测试与优化。FlowGen^[25]在此思路更上一步，借鉴了传统的软件工程开发方法（如瀑布模型、敏捷开发与测试驱动开发），构建了4层智能体架构，包括需求工程师、架构师、开发者和测试人员。该系统通过阶段性规划与目标验证推进任务，展现了良好的可扩展性与灵活性。SoA^[26]则引入动态调度机制，根据任务复杂度与资源需求，自适应地扩展或收缩智能体数量，实现对大规模代码库的高效管理与生成。

自我协商优化机制则强调通过智能体间的反复协商与反馈提升代码生成质量。此类方法通常设计多个并行智能体对候选

解进行评估、优化与修复，以不断提高最终输出的稳健性。MapCoder^[27]采用由4个专业化智能体组成的循环机制，这些智能体通过回忆示例、设计方案、生成代码与调试错误的迭代过程逐步优化生成结果，在多个步骤任务中表现稳定。AutoSafeCoder^[28]进一步结合编码器、静态分析器与模糊测试器的多角色架构，通过静态与动态检测反馈不断修正生成代码，强化安全性。QualityFlow^[29]则构建了一支LLM智能体团队，先行生成单元测试，再通过质量检查器评估代码与测试的一致性，最终执行测试以确认正确性，实现了从生成到验证的闭环优化。

随着系统复杂度的提升，自我进化机制逐渐成为提升多智能体系统适应性的关键途径。在该机制下，系统能够在运行过程中自我学习并动态调整组织结构与行为策略，以应对任务环境的变化。SEW^[30]提出了基于工作流的自我进化方法，系统能够依据协作效率与故障反馈实时重组通信路径与责任分配，从而在运行时实现自我调整与优化。EvoMAC^[31]则受到神经网络训练过程的启发，提出“文本反向传播”机制，通过比较目标要求与生成结果并利用环境反馈，自动调整智能体间的协作结构与策略，从而持续增强系统的自我进化能力。

2.2 共享记忆与知识管理

在代码生成任务中，多智能体系统往往需要共享和引用跨任务、跨时间的长期依赖信息，例如任务描述、历史修改记录与中间生成结果。因此，如何构建一个可写、可读且具备可扩展性的全局上下文空间，成为决定系统性能上限的关键因素之一。针对这一挑战，研究者提出了多种共

享记忆与知识管理机制，以确保智能体在协作过程中保持一致性、灵活性与可扩展性。

Self-Collaboration是最早将黑板模型 (blackboard model) 引入代码生成任务的系统。该方法通过建立显式共享内存空间来存储任务描述、中间结果与修改记录，允许所有智能体根据任务需求灵活地读写其中内容。黑板模型不仅提供了统一的共享视图，还突破了传统单向信息传递的局限，显著增强了智能体间的交互灵活性，从而有效提升了多智能体协作效率。黑板模型的优势在于实时协作能力强，缺点是内存开销大，在高并发场景下易冲突，适用条件为中小规模团队模拟。

L2MAC 系统^[32]则借鉴冯·诺伊曼架构，设计了分离式指令寄存器与文件存储模块，并引入显式控制单元精确调度上下文内容。其核心思想是将上下文按程序单元进行组织，从而突破语言模型上下文窗口的限制。在处理大型函数、多文件项目等复杂场景时，L2MAC 系统能够高效地保持跨阶段的上下文信息，避免了信息量过大导致的上下文丢失问题，展现了卓越的扩展性与稳健性。

Hintikka^[33]从神经生物学中的“认知—记忆—增长”模型汲取灵感，提出类脑上下文管理机制，将信息划分为短期记忆、长期知识库与进化增长单元。短期记忆用于维持任务执行的即时状态，长期知识库用于积累通用经验，而进化单元则不断提升系统的抽象与推理能力。该机制不仅赋予智能体层次化的记忆结构，还增强了自我进化与持续学习能力，使其能够在长期任务中逐步积累知识并实现自我优化。

SoA 则是一个自组织的多智能体框架，根据任务复杂度与资源需求动态扩展或收缩智能体池。在该框架中，每个智能体在

固定上下文窗口内独立运行，而中央控制器则负责保持全局子空间结构的一致性。通过这种方式，SoA 在避免单一智能体负载过重的同时，有效减少了信息稀疏与冗余问题，从而在大规模代码库的生成与管理中展现出良好的可扩展性与效率。

2.3 群体反思与协作进化

随着多智能体代码生成研究的逐步成熟，学术界开始将关注点转向如何进一步提升智能体之间的协作能力。这一方向通常被概括为群体反思与协作优化。其核心思想是在训练与推理过程中引入团队级的协作建模机制，通过整体优化多个智能体的行为，提升系统的综合性能。尽管这一方向仍处于早期探索阶段，相关方法与框架尚不完善，但其潜力与应用前景已引起学术界与工业界的广泛关注。

Lingma SWE-GPT^[34]通过模拟真实的软件开发流程，将任务分解为3个阶段——代码库理解、故障定位与补丁生成，每个阶段由专门的子任务智能体负责。在此框架下，各智能体独立执行任务，但最终目标是通过协作优化提升整体执行效率。具体而言，系统会收集各阶段智能体的行为数据，包括推理轨迹、工具调用与最终结果，并通过监督微调实现团队级优化。这种方法不仅提升了多阶段协作的质量与稳定性，还展示了任务分解与智能体协作有机结合的可能性。

协作优化的另一大挑战在于应对多智能体协作过程中的“不同步”问题，不同步问题指状态漂移，如 a1 更新 M_s 后 a2 未同步，导致冲突。SyncMind^[35]从系统层面提出了解决方案，其核心是通过三维评估机制帮助智能体识别并修复在更新共享代码库时出现的状态漂移。这一机制有效减

少了因信息不对称或同步延迟导致的性能损失，为多智能体协作编码 workflow 提供了新的理论视角与实践路径。

除了团队级的行为建模，群体反思与协作优化还强调在迭代中持续提升系统的稳定性与鲁棒性。MapCoder 采用循环式的协作机制，由多个专门智能体协同生成代码、检测缺陷并执行修复。通过多轮协作与反思，该系统能够不断提升输出质量，在处理复杂任务时展现出显著的稳定性与一致性。

协作优化机制并非依赖于固定的分工模式，而是通过持续学习与自我进化不断改进系统能力。EvoMAC 借鉴神经网络中的反向传播思想，提出“文本反向传播”机制，通过比较生成结果与目标要求的差异，自动调整智能体之间的协作结构和行为策略。这一方法不仅提升了系统的自适应性，还在代码生成质量与协作效率上取得了显著优化。

3 代码智能体的应用场景

3.1 自动化代码生成

代码智能体最核心的应用之一是自动化代码生成。传统编程方法依赖开发者手动编写代码，效率低且容易出现错误。通过结合自然语言理解和推理能力，代码智能体能够根据开发需求自动生成高质量的代码。代码智能体通常通过需求分析、功能描述、框架选择等步骤生成从函数级到模块级的代码。

Self-Planning 框架通过将复杂任务分解为多个高层次的步骤，使代码智能体能够在理解需求后逐步生成所需的代码。CodeChain 在生成的过程中引入了自我修正机制，可以根据之前的生成步骤和反馈

进一步调整代码，确保每一步都符合需求并优化结果。这使得代码智能体能够在多个开发场景中高效工作，自动生成符合功能要求的代码，并能在任务变动时灵活调整。

此外，HyperAgent^[36]通过跨语言生成代码的能力，能够根据自然语言描述自动生成不同编程语言下的代码，提高了跨语言编程的效率和灵活性。通过智能体的帮助，开发者能够更快速地实现系统功能，从而节省大量的开发时间。

3.2 自动化调试与程序修复

自动化调试和程序修复是提升代码质量和开发效率的重要任务。代码智能体能够通过分析错误信息、测试结果以及性能反馈来自动诊断代码中的缺陷，并提出修复方案。与传统的调试方法相比，代码智能体能够更加高效地定位问题并提出优化建议，减少人工干预。

RepairAgent 系统^[37]利用自动化推理与错误检测，在生成代码时会根据错误信息自动进行修复。MAGIS 通过多智能体协作，帮助开发者识别代码中的逻辑错误、性能瓶颈和潜在的安全漏洞。在 AutoSafeCoder 中，静态分析器和模糊测试器与生成器配合，能自动修复代码中的安全漏洞，使代码更加健壮和安全。

3.3 自动化测试代码生成

自动化测试是确保软件质量和可靠性的重要环节。代码智能体能够自动生成单元测试、集成测试和安全测试代码，通过全面的测试覆盖提高软件的质量和稳定性。与手动编写测试用例相比，代码智能体能够更加快速、全面地生成测试代码，并自动发现潜在的缺陷。

TestPilot 系统^[38]利用自然语言描述生成完整的测试用例，不仅能够针对单一功能进行验证，还能考虑到边界条件和潜在的异常情况。CANDOR 系统^[39]自动生成代码后，创建大量的测试用例以确保代码在不同场景下的鲁棒性。此外，QualityFlow 系统通过生成单元测试并对代码进行质量检查，确保生成的代码符合需求并经过严格的测试验证。

3.4 代码重构与优化

随着开发过程的推进，代码的可维护性和性能可能会受到影响，因此代码重构和优化成为提高系统质量的必要环节。代码智能体能够通过静态分析、性能分析等手段自动识别代码中的瓶颈、冗余和低效部分，并进行优化。

Baumgartner 等^[40]提出的基于 LLM 的模块化自动重构管道，能够自动检测代码中的代码异味（code smell），并提出相应的重构建议。类似地，EM-Assist^[41]和

iSMELL^[42]能够根据代码的结构和性能要求自动进行代码优化，并确保优化后的代码更加高效、可读且符合开发标准。

4 代码智能体相关评测基准

代码生成任务的评估高度依赖统一的基准数据集。早期研究多集中在独立、基础的函数或类级代码生成方面，而最新的基准更关注贴近实际的软件开发过程，考察智能体在复杂环境中进行交互、任务规划与执行的综合能力。基于协作粒度和优化机制进行划分，现有评测体系可分为两类（见表 2）：基础的代码级单一生成任务基准，以及面向真实开发情境的软件工程基准。

4.1 代码级生成任务基准

在代码生成智能体研究的初期，评测任务多集中于独立、明确的编程问题，

表 2 代码生成能力评测基准

评测基准	任务层级	数据来源与规模	编程语言	核心评测目标
HumanEval ^[43]	单一生成	164 个手工设计问题	Python	功能正确性
MBPP ^[44]	单一生成	974 个问题,每题含 3 个断言	Python	功能正确性
APPS ^[45]	单一生成	10 000 个题目	Python	算法推理
CodeContests ^[13]	单一生成	13 328 道真实竞赛题	C++,Python,Java	算法推理
LiveCodeBench ^[46]	单一生成	持续更新,1 055 道算法题	Python,C++	算法推理
SWE-Bench ^[47]	软件工程	12 个 Python 仓库,数千个 Issue	Python	项目执行、功能正确性
SWE-Bench-Lite	软件工程	300 个简化任务	Python	功能正确性
SWE-Bench-Verified	软件工程	500 个人工验证样本	Python	功能正确性
CodeAgentBench ^[16]	软件工程	5 个真实项目、101 个任务	Python	项目执行、上下文理解
Web-Bench ^[48]	软件工程	Web 项目	JS,Python	长程规划、上下文理解
EvoCodeBench ^[49]	软件工程	真实项目版本数据	Python,Java	项目执行、代码演化
DevEval ^[50]	软件工程	真实项目场景	多语言	项目执行、上下文理解

旨在检验模型能否根据自然语言描述生成语法和功能均正确的代码。这一阶段的工作通常被称为函数级或类级代码生成，即模型根据高层功能需求生成完整的方法实现。这类基准不仅为后续复杂任务研究提供了方法论基础，还确立了以功能正确性为核心的评估标准，并被广泛沿用至今。

2021年，OpenAI提出HumanEval基准^[43]，该基准具有划时代的意义，它借鉴OpenJudge代码评测平台的思路，为每个任务配置单元测试，从而将评估方式从主观的文本匹配转向可自动执行、量化的代码正确性测试。同样，由Google推出的MBPP^[44]面向初级Python编程任务设计，每个题目均提供自然语言描述、标准答案及3条断言测试语句，涵盖从数学运算到基础数据处理等多种任务类型，为入门级代码生成提供了系统化的评测框架。

随着大语言模型能力的持续提升，代码生成评测的重心逐渐从基础函数实现转向更具挑战性的编程竞赛类任务。这类任务要求模型不仅能正确编写代码，还需展现深入的逻辑推理与算法设计能力。学术界基于此提出了一系列编程竞赛型评测基准，其中APPS^[45]、CodeContests^[13]和LiveCodeBench^[46]最具代表性。APPS汇集了来自Codeforces等平台的1万道题目，并按难度分级；CodeContests涵盖了13 000余道来自真实竞赛的题目，支持多种编程语言；LiveCodeBench为防止数据泄漏问题，定期从主流在线平台更新最新题目，为代码生成模型提供了更加动态且严谨的评测环境。

4.2 真实软件开发环境基准

随着代码生成智能体的能力不断提升，

研究重心从单一生成任务转向能在完整软件开发环境中独立执行复杂任务的自主智能体。因此，学术界陆续提出了一系列模拟真实软件工程流程的基准测试。这类基准的共同特点是构建了可交互的开发环境，使智能体像人类程序员一样，通过访问代码库、命令行和调试工具等资源完成任务。

SWE-Bench^[47]是最具代表性的基准之一，它收录了来自12个热门Python项目的数千个真实的GitHub Issue，覆盖缺陷修复与功能扩展等任务。其衍生版本SWE-Bench-Lite聚焦更小规模的快速评测，而SWE-Bench-Verified则经过人工验证来确保样本质量。CodeAgentBench^[16]提供了5个真实软件项目和101个任务，要求模型基于文档与现有代码实现新功能。Web-Bench^[48]则通过项目式的连续任务设计，考察智能体在长期规划与跨步骤记忆方面的表现。此外，EvoCodeBench^[49]与DevEval^[50]等数据集进一步扩展至完整项目层面，要求智能体在真实软件工程环境中生成可运行、可维护的目标代码，标志着代码生成评测正全面迈向真实开发场景阶段。

5 代码智能体产品和应用

随着模型性能的持续提升与基准测试体系的不断完善，研究重心逐渐从算法层面的能力验证转向真实开发环境中的可用性与可靠性。这一趋势推动了代码智能体从学术研究阶段迈向产品化落地阶段。近年来，业界围绕代码智能体构建出一系列面向开发者的智能编程产品^[51-53]。这些系统不再局限于单次代码生成，而是通过与集成开发环境（integrated development environment，IDE）、命令行工具和项目

上下文的深度结合，展现出持续交互、任务规划与自主执行的特征。它们在实际工作流程中扮演了从辅助编写到项目的多层角色，标志着人工智能在软件工程领域的应用进入新阶段。

由 GitHub 与 OpenAI 联合推出的 GitHub Copilot 是最早的大规模代码生成应用之一。基于 Codex 及后续的 GPT-4 模型，GitHub Copilot 可直接集成于 Visual Studio Code 等主流 IDE，实现代码自动补全、注释理解与行内生成建议。GitHub Copilot 的重要意义在于其普适性与可用性，它首次将 AI 编程助手引入主流开发工作流，使语言模型从研究原型真正成为开发者的日常工具。

Cursor 代表了新一代的智能体 IDE。与传统自动补全不同，Cursor 支持用户与开发环境进行交互式对话。智能体能够修改代码、执行命令、分析运行错误并迭代改进方案。通过结合 GPT-4 或 Claude 等推理引擎与本地项目上下文嵌入，Cursor 有效地弥合了静态代码生成与动态任务执行之间的差距。这一设计理念与 SWE-Bench 等真实项目类基准高度契合，强调在真实开发约束下的推理与操作能力。

阿里云推出的通义灵码（Tongyi Lingma）是基于 Qwen2 与 CodeQwen 模型系列构建的企业级智能编程助手。与通用型助手不同，通义灵码深度集成企业内部的知识库、代码仓库与 API，能够进行领域专属的代码生成与安全部署，并创新性地使用蒙特卡罗树搜索进行系统导航和故障定位，实现对复杂软件存储库的深入理解。

由 Exafunction 开发的 Codeium 定位为注重隐私保护的 AI 编程助手。它支持超过 100 种编程语言，并可与主流 IDE 无缝集成。该系统针对时延与数据隐私进行了

优化，能够在不上传用户数据的情况下提供实时建议。Codeium 体现了业界在模型能力与数据安全之间寻找平衡的趋势，对于企业级落地尤为重要。

6 结束语

本文系统综述了基于大语言模型的代码智能体的研究进展，围绕理论基础、智能体基础能力建设、多智能体组织能力建设以及典型应用场景进行了全面梳理。代码智能体不仅能够在代码生成、调试、测试和优化等环节展现出强大的自动化能力，还在需求澄清与团队协作方面展现出巨大的潜力。单智能体技术的不断完善使其逐步接近人类程序员的工作模式，而多智能体框架则通过角色分工与协同优化，推动了代码智能体在复杂工程环境中的应用拓展。

然而，现阶段的研究仍面临诸多挑战，包括上下文信息管理的有效性、多智能体间高效协作的机制设计，以及生成代码的安全性与稳定性等问题。未来，代码智能体有望在以下两方面取得进展：一方面提升自我学习与自适应能力，以更好地应对动态开发环境；另一方面探索更高效的多智能体进化机制，实现大规模软件工程任务的智能协作。

参考文献：

[1] Touvron H, Lavril T, Izacard G, et al. LLaMA: open and efficient foundation language models[PP]. V1. arXiv (2023-02-27)[2025-11-10]. arXiv:2302.13971.

[2] Touvron H, Martin L, Stone K, et al. Llama 2: open foundation and fine-

- tuned chat models[PP]. V2. arXiv (2023-07-19)[2025-11-10]. arXiv:2307.09288.
- [3] Agarwal S, Almeida D, Askeel A, et al. Training language models to follow in - instructions with human feedback[C]// Proceedings of the Advances in Neural Information Processing Systems 35. Neural Information Processing Systems Foundation, Inc., 2022: 27730-27744.
- [4] Grattafiori A, Dubey A, Jauhri A, et al. The llama 3 herd of models[PP]. V3. arXiv (2024-11-23)[2025-11-10]. arXiv: 2407.21783.
- [5] Guo D Y, Yang D J, Zhang H W, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning[J]. Nature, 2025, 645(8081): 633-638.
- [6] Xi Z H, Chen W X, Guo X, et al. The rise and potential of large language model based agents: a survey[J]. Science China Information Sciences, 2025, 68 (2): 121101.
- [7] Wang L, Ma C, Feng X Y, et al. A sur - vey on large language model based au - tonomous agents[J]. Frontiers of Com - puter Science, 2024, 18(6): 186345.
- [8] Kitzelmann E. Inductive programming: a survey of program synthesis techniques[M]// Approaches and Applications of Induc - tive Programming. Heidelberg: Springer, 2010: 50-73.
- [9] Ling W, Grefenstette E, Hermann K M, et al. Latent predictor networks for code generation[PP]. V2. arXiv (2016-06-08) [2025-11-10]. arXiv:1603.06744.
- [10] Yin P C, Neubig G. A syntactic neural model for general-purpose code genera - tion[PP]. V1. arXiv (2017-04-06) [2025- 11-10]. arXiv:1704.01696.
- [11] Rozi è re B, Gehring J, Gloeckle F, et al. Code llama: open foundation models for code[PP]. V3. arXiv (2024-01-31)[2025- 11-10]. arXiv:2308.12950.
- [12] Wang Y, Wang W S, Joty S, et al. Co - deT5: identifier-aware unified pre- trained encoder-decoder models for code understanding and generation[PP]. V1. arXiv (2021-09-02) [2025-11-10]. arXiv:2109.00859.
- [13] Li Y J, Choi D, Chung J, et al. Competition-level code generation with AlphaCode[J]. Science, 2022, 378(6624): 1092-1097.
- [14] Zhang K C, Zhang H Z, Li G, et al. ToolCoder: Teach Code Generation Models to use API search tools[PP]. V5. arXiv (2023-09-11)[2025-11-10]. arXiv: 2305.04032.
- [15] Zhang K C, Li J, Li G, et al. Co - deAgent: enhancing code generation with tool-integrated agent systems for real-world repo-level coding challenges [PP]. V2. arXiv (2024-08-09) [2025-11- 10]. arXiv:2401.07339.
- [16] Li J R, Le H, Zhou Y B, et al. Code - Tree: agent-guided tree search for code generation with large language models[PP]. V2. arXiv (2024-11-12) [2025-11-10]. arXiv:2411.04329.
- [17] Alon U, Clark P, Dziri N, et al. Self- refine: iterative refinement with self- feedback[C]//Proceedings of the Ad - vances in Neural Information Processing Systems 36. Neural Information Pro - cessing Systems Foundation, Inc., 2023: 46534-46594.
- [18] Chen X Y, Lin M, Schärli N, et al. Teaching large language models to self- debug[PP]. V2. arXiv (2023-10-05) [2025-11-10]. arXiv:2304.05128.
- [19] Olausson T X, Inala J P, Wang C L, et al. Is self-repair a silver bullet for code generation? [PP]. V5. arXiv (2024-02-02) [2025-11-10]. arXiv:2306.09896.

- [20] Dong Y H, Jiang X, Jin Z, et al. Self-collaboration code generation *via* Chat-GPT[J]. ACM Transactions on Software Engineering and Methodology, 2024, 33 (7): 1–38.
- [21] Huang D, Zhang J M, Luck M, et al. AgentCoder: multi-agent-based code generation with iterative testing and optimisation[PP]. V3. arXiv (2024–05–24) [2025–11–10]. arXiv:2312.13010.
- [22] Rasheed Z, Waseem M, Saari M, et al. Codepori: Large scale model for autonomous software development by using multi-agents[PP]. V2. arXiv (2024–09–18)[2025–11–10]. arXiv:2402.01411.
- [23] Cheng Y, Tao W, Wang Y L, et al. MAGIS: LLM-based multi-agent framework for GitHub issue resolution[C]// Proceedings of the Advances in Neural Information Processing Systems 37. Neural Information Processing Systems Foundation, Inc., 2024: 51963–51993.
- [24] Zhang H, Cheng W, Wu Y H, et al. A pair programming framework for code generation *via* multi-plan exploration and feedback-driven refinement[C]// Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering. New York: ACM, 2024: 1319–1331.
- [25] Lin F, Kim D J, Tse-Husn, et al. SOEN-101: code generation by emulating software process models using large language model agents[PP]. V2. arXiv (2024–10–31) [2025–11–10]. arXiv: 2403.15852.
- [26] Ishibashi Y, Nishimura Y. Self-organized agents: a LLM multi-agent framework toward ultra large-scale code generation and optimization[PP]. V1. arXiv (2024–04–02) [2025–11–10]. arXiv: 2404.02183.
- [27] Islam M A, Ali M E, Parvez M R. Map-Coder: multi-agent code generation for competitive problem solving[PP]. V1. arXiv (2024–05–18)[2025–11–10]. arXiv: 2405.11403.
- [28] Nunez A, Islam N T, Jha S K, et al. AutoSafeCoder: a multi-agent framework for securing LLM code generation through static analysis and fuzz testing [PP]. V2. arXiv (2024–11–05) [2025–11–10]. arXiv:2409.10737.
- [29] Hu Y J, Zhou Q, Chen Q H, et al. QualityFlow: an agentic workflow for program synthesis controlled by LLM quality checks[PP]. V2. arXiv (2025–03–24) [2025–11–10]. arXiv:2501.17167.
- [30] Liu S W, Fang J Y, Zhou H, et al. SEW: self-evolving agentic workflows for automated code generation[PP]. V2. arXiv (2026–04–14) [2025–11–10]. arXiv: 2505.18646.
- [31] Hu Y, Cai Y Z, Du Y X, et al. Self-evolving multi-agent collaboration networks for software development[PP]. V1. arXiv (2024–10–22) [2025–11–10]. arXiv:2410.16946.
- [32] Holt S, Luyten M R, van der Schaar M. L2MAC: large language model automatic computer for extensive code generation [PP]. V6. arXiv (2025–06–27) [2025–11–10]. arXiv:2310.02003.
- [33] Hintikka J. Cogito, ergo sum: inference or performance? [J]. The Philosophical Review, 1962, 71(1): 3.
- [34] Ma Y W, Cao R Y, Cao Y C, et al. Lingma SWE-GPT: an open development-process-centric language model for automated software improvement[PP]. V1. arXiv (2024–11–01) [2025–11–10]. arXiv:2411.00622.
- [35] Guo X H, Wang X Y, Chen Y Y, et al. SyncMind: measuring agent out-of-

- sync recovery in collaborative software engineering[PP]. V2. arXiv (2025-06-09) [2025-11-10]. arXiv:2502.06994.
- [36] Phan H N, Nguyen T N, Nguyen P X, et al. HyperAgent: generalist software engineering agents to solve coding tasks at scale[PP]. V3. arXiv (2025-09-05) [2025-11-10]. arXiv:2409.16299.
- [37] Bouzenia I, Devanbu P, Pradel M. RepairAgent: an autonomous, LLM-based agent for program repair[PP]. V2. arXiv (2024-10-28) [2025-11-10]. arXiv: 2403.17134.
- [38] Schäfer M, Nadi S, Eghbali A, et al. An empirical evaluation of using large language models for automated unit test generation[PP]. V4. arXiv (2023-12-11) [2025-11-10]. arXiv:2302.06527.
- [39] Xu Q, Wang G, Briand L, et al. A multi-agent LLM-based JUnit test generation with strong oracles[PP]. V7. arXiv (2026-03-25) [2026-04-20]. arXiv: 2506.02943, 2025.
- [40] Baumgartner N, Iyengar P, Schoe-maker T, et al. AI-driven refactoring: a pipeline for identifying and correcting data clumps in git repositories[J]. Electronics, 2024, 13(9): 1644.
- [41] Pomian D, Bellur A, Dilhara M, et al. EM-assist: safe automated Extract-Method refactoring with LLMs[C]//Proceedings of the Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering. New York: ACM, 2024: 582-586.
- [42] Wu D, Mu F W, Shi L, et al. iSMELL: assembling LLMs with expert toolsets for code smell detection and refactoring[C]//Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering. New York: ACM, 2024: 1345-1357.
- [43] Chen M, Tworek J, Jun H, et al. Evaluating large language models trained on code[PP]. V2. arXiv (2021-07-14) [2025-11-10]. arXiv:2107.03374.
- [44] Austin J, Odena A, Nye M, et al. Program synthesis with large language models[PP]. V1. arXiv (2021-08-16) [2025-11-10]. arXiv:2108.07732.
- [45] Hendrycks D, Basart S, Kadavath S, et al. Measuring coding challenge competence with APPS[PP]. V3. arXiv (2021-11-08) [2025-11-10]. arXiv:2105.09938.
- [46] Jain N, Han K, Gu A, et al. LiveCode-Bench: holistic and contamination free evaluation of large language models for code[PP]. V2. arXiv (2024-06-06) [2025-11-10]. arXiv:2403.07974.
- [47] Jimenez C E, Yang J, Wettig A, et al. SWE-bench: can language models resolve real-world GitHub issues? [PP]. V3. arXiv (2024-11-11) [2025-11-10]. arXiv:2310.06770.
- [48] Xu K, Mao Y W, Guan X Y, et al. Web-bench: a LLM code benchmark based on web standards and frameworks[PP]. V1. arXiv (2025-05-12) [2025-11-10]. arXiv: 2505.07473.
- [49] Li J, Li G, Zhang X M, et al. EvoCode-Bench: an evolving code generation benchmark aligned with real-world code repositories[PP]. V1. arXiv (2024-03-31) [2025-11-10]. arXiv:2404.00599.
- [50] Li J, Li G, Zhao Y F, et al. DevEval: a manually-annotated code generation benchmark aligned with real-world code repositories[PP]. V1. arXiv (2024-05-30) [2025-11-10]. arXiv:2405.19856.
- [51] 姜春宇, 韩晓璐, 王超伦. 智能体技术演进: 场景分析、关键技术与发展趋势[J]. 大数据,

2026, 12(2): 129-138.

Jiang C Y, Han X L, Wang C L. Evolution of agent technology: scenario analysis, key technologies and development trends[J]. Big Data Research, 2026, 12(2): 129-138.

[52] 王涵, 俞扬, 姜远. 基于动态自选择参数共享的合作多智能体强化学习算法[J]. 智能科学与技术学报, 2022, 4(1): 75-83.

Wang H, Yu Y, Jiang Y. A cooperative multi-agent reinforcement learning algorithm based on dynamic self-selection parameters sharing[J]. Chinese Journal of Intelligent Science and Technology, 2022, 4(1): 75-83.

[53] 项凤涛, 罗俊仁, 谷学强, 等. 群视角下的多智能体强化学习方法综述[J]. 智能科学与技术学报, 2023, 5(3): 313-329.

Xiang F T, Luo J R, Gu X Q, et al. Survey on multi-agent reinforcement learning methods from the perspective of population[J]. Chinese Journal of Intelligent Science and Technology, 2023, 5(3): 313-329.

作者简介



张旭龙（1988-），男，博士，平安科技（深圳）有限公司算法研究员，2023年入选上海市东方英才计划青年项目。兼任清华大学深圳国际研究生院及中国科学技术大学先进技术研究院校外导师，以及中国指挥与控制学会具身智能专业委员会委员、中国自动化学会联邦数据与联邦智能委员会委员、中国电子音响行业协会声音与音乐技术专委会常务委员。目前是IEEE、中国自动化学会以及中国计算机学会会员。主要研究方向为大模型、具身智能、跨模态智能计算等。发表学术论文80余篇，获得国家发明专利授权20余项。



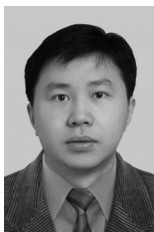
鲍习坤（2001-），男，平安科技（深圳）有限公司算法工程师，中国科学技术大学硕士生，主要研究方向为智能体强化学习和多模态大模型。



瞿晓阳（1988-），男，博士，平安科技（深圳）有限公司前沿机器学习算法分组负责人，清华大学深圳国际研究生院校外导师，中国科技大学校先进技术研究院校外导师，中佛罗里达大学访问学者，主要从事机器学习、大数据、体系结构方面的研究工作，在语音语义分析、自动化机器学习、零样本和小样本学习、高性能计算与存储等方面经验丰富。近几年，在体系结构方向（如INFOCOM、DAC、IPDPS、TPDS）和人工智能方向（如NeurIPS、IJCAI、ICASSP、Interspeech）的国际顶级会议和顶级期刊发表过近50篇文章，1篇论文荣获会议最佳学生论文奖提名，担任过多个国际顶级期刊的评委，已授权专利70篇，已出版专著两本。



倪晓俊（1973-），男，中国科学院计算技术研究所助理研究员，主要从事数据库安全、大数据分析、隐私计算等方面的研究工作，曾获得北京市科技进步奖一等奖等奖励。



田晖（1982-），男，博士，博士生导师，华侨大学教授，计算机科学与技术学院院长、网络与信息安全产业学院院长，厦门市数据安全与区块链技术重点实验室主任；入选福建省青年拔尖创新人才、福建省百千万人才工程省级人选、福建省高等学校新世纪优秀人才、福建省高校杰出青年科研人才培育计划。现为IEEE 高级会员、中国计算机学会（CCF）杰出会员、CCF互联网专委会执行委员、CCF区块链技术专委会执行委员、中国中文信息学会大数据安全与隐私保护专业委员会常务委员、福建省计算机学会副理事长、福建省网络空间安全学会理事长。主要从事网络与信息安全、人工智能安全，数据安全、多媒体内容安全等方面的研究工作。近年来，主持各类科技项目20余项，其中包括国家自然科学基金项目4项，国家重点研发课题和子课题各1项；在IEEE TDSC、IEEE TIFS、IEEE TASLP、ACM MM等国内外知名期刊和学术会议上发表论文120余篇；获得授权发明专利40余项，软件著作权8项；部分研究成果曾荣获中国电子学会科学技术奖二等奖、海南省自然科学奖二等奖、福建省科学技术进步奖三等奖和福建省自然科学奖三等奖等奖励。



王健宗（1983-），男，博士，正高级工程师，平安科技（深圳）有限公司智能金融前沿技术研究院院长。美国佛罗里达大学人工智能博士后，美国莱斯大学和华中科技大学联合培养博士，CCF杰出会员，CCF大数据专家委员会委员，中国自动化学会联邦数据和联邦智能专业委员会副主任。主要研究方向为具身智能、大模型、联邦学习和深度学习等。荣获中国专利优秀奖、广东省科学技术奖二等奖、吴文俊人工智能科学技术奖一等奖、中国人民银行金融科技发展奖一等奖、深圳高层次人才领军人才、福田英才等奖项。

收稿日期: 2026-02-09

通信作者: 鲍习坤, xikun0625@mail.ustc.edu.cn

基金项目: 深港联合资助项目(A类)(No. SGDX20240115103359001)

Foundation Items: Shenzhen-Hong Kong Joint Funding Project (Category A) (No. SGDX20240115103359001)