CAICT 中国信通院 HUAWEI

集智 专题报告

智能化软件开发落地实践指南

(2024 年)

中国信息通信研究院人工智能研究所

华为云计算技术有限公司

2024年9月

版权声明

前　言

2024 年《政府工作报告》首次提出“人工智能+”战略行动，旨在推动人工智能赋能各行各业。以大模型为代表的新一代人工智能技术迅猛发展，已成为软件工程领域智能化转型的关键驱动力，为软件开发、测试等环节注入新智力。智能化软件开发工具凭借其强大的代码理解和生成能力，有效降低了开发人员的技术门槛，并提高开发效率和质量，进一步推动软件开发领域的进步。

然而，在软件开发的智能化能力建设和应用过程中，仍面临诸多挑战，如代码大模型选择困难、开发工具的工程化建设复杂、智能化能力建设无参考、开发场景选择和落地难，以及与现有开发工具或流水线的集成难等问题。为此，本指南旨在为正在进行软件开发智能化转型的企业提供一份实用指南，为企业制定落地策略、建设智能开发能力体系提供有力参考。本指南系统梳理了智能化软件开发的发展历程、现状和面临的挑战，结合大模型和软件开发特点，提出了具体的落地方向、路径和框架，详细阐述了落地所需的核心能力和使能能力，并对多个行业的落地案例进行了深入剖析，最后对智能化软件开发的发展趋势进行了展望。

由于大模型等人工智能技术产业仍处于快速发展阶段，智能开发相关的技术产品、服务和应用也在不断演变，我们的认识将随着产业实践而不断深化，报告存在的不足之处，恳请大家批评指正。

目 录

一、智能开发发展概述 1
（一） 软件工程步入 3.0 时代 1
（二） 智能开发发展现状 5
（三） 智能开发价值显现 7
（四） 智能开发落地挑战 9
二、智能开发落地策略 11
（一） 智能开发落地原则 11
（二） 智能开发落地步骤 12
（三） 智能开发落地框架 15
三、智能开发核心能力建设 17
（一） 代码生成与补全 17
（二） 单元测试用例生成 19
（三） 代码转换与优化 21
（四） 代码解释与注释 22
（五） 代码检查与修复 23
（六） 研发问答 24
四、智能开发使能能力建设 26
（一） 代码数据处理能力 26
（二） 代码大模型优化能力 28
（三） 智能开发能力评估 30
（四） 智能开发安全能力 32
（五） 其他工程化能力 36
五、智能开发落地案例分析 41
（一） 云服务行业案例 41
（二） 软件服务业案例 42
（三） 电力行业案例 44
（四） 金融行业案例 46
（五） 制造行业案例 47
六、总结与展望 50

图 目 录

图 1　软件工程发展历程图 1
图 2　软件工程 3.0 示意图 3
图 3　智能开发落地步骤图 12
图 4　企业智能化能力自我诊断等级图 13
图 5　智能开发能力实施的多阶方案图 14
图 6　智能开发落地框架示意图 15
图 7　三“码”合一方案图 19
图 8　单测生成实战方案示意图 21
图 9　基于 RAG 的代码理解流程图 23
图 10　数据处理流程图 26
图 11　代码大模型优化过程示意图 29
图 12　智能开发能力评估架构图 31
图 13　安全能力体系架构图 33
图 14　数据安全治理流程示意图 33
图 15　模型安全治理架构图 35
图 16　应用安全治理框架图 35
图 17　RAG 落地流程示意图 39
图 18　某云服务企业案例落地方案示意图 42
图 19　某软件服务企业案例落地方案示意图 44
图 20　某电力行业企业案例落地方案示意图 45
图 21　某银行案例落地方案示意图 47
图 22　某家电制造企业案例落地方案示意图 49

表 目 录

表 1　准确性评估场景与指标参考清单 32
表 2　提示词示例 1 38
表 3　提示词示例 2 38

一、智能开发发展概述

随着人工智能技术的不断进步，特别是大模型能力的持续提升，软件工程领域正迎来前所未有的变革，软件开发的流程和模式正被重新定义，智能化进程显著加速。本章将简要回顾软件工程的发展历程，重点介绍软件工程 3.0 的发展特点，以及智能化软件开发（以下简称：智能开发）领域发展现状，并深入探讨智能化能力为软件开发带来的价值提升，最后梳理落地挑战。

（一）软件工程步入 3.0 时代

1.软件工程发展历程

软件工程自 1968 年诞生以来，其发展历程可分成三个阶段，包括软件工程 1.0、软件工程 2.0 和软件工程 3.0，如图 1 所示。

图 1 软件工程发展历程图

软件工程 1.0，即第一代软件工程，亦称为“传统软件工程”。本阶段的发展起点可追溯至上个世纪六十年代，随着大容量、高速度计算机的问世，软件开发需求急剧增长，软件系统的规模越来越大，复杂程度越来越高，可靠性问题更加突出。为解决该类软件危机，

1968 年北大西洋公约组织（NATO）在国际会议上首次提出“软件工程”概念，期望将软件开发带出混乱的局面，走向有纪律、有流程的规范化之路。软件工程 1.0 推崇结构化方法，重视流程和文档规范化等工程实践，对提升软件产品交付成功率和质量有重大意义。但在软件工程 1.0 时期，由于软件被当作传统的工业制造产品进行交付，无法及时满足业务需求的变化和更新，使得交付周期长。

软件工程 2.0，亦称“敏捷软件工程”。本阶段的发展自 2001 年“敏捷软件开发宣言[1]”发布起，通过践行敏捷开发最佳实践，实现持续集成（CI）、持续交付（CD）和快速迭代以更好地满足日益变化的业务需求。软件工程 2.0 时期，软件形态从“产品”发展为“服务”，而服务的性能、效率、可靠性、可持续性也更加重要，这推动人们将软件开发和运维融合起来，DevOps 模式应运而生，使得软件交付效率更加高效。但本时期的开发、测试等核心工作，多以人工为主，技术门槛和研发成本居高不下，对技术人员的依赖性非常高，虽然陆续有自动化技术的出现和落地，对效能提升方面未有显著成效，“软件工程没有银弹[2]”这一观点仍然有效。

软件工程 3.0，亦称“智能化软件工程”。2022 年底以来，随着 ChatGPT 等大模型相继发布，软件工程迎来新一轮变革。大模型凭借其强大的理解和生成能力，通过代码生成、代码续写、测试用例生成、智能问答等能力，为软件工程带来了智能化能力升级，为提

[1] https://agilemanifesto.org/
[2] 《没有银弹：软件工程的本质与偶然》

升软件质量和效率带来了新动力，全新的软件开发范式正在诞生。软件工程中到底有没有“银弹”，在大模型时代成为热议的话题。

2.软件工程 3.0 发展特点

软件工程 3.0 围绕“智能化”理念以构建智能化助手为起点，通过使用大模型为核心的 AI 技术驱动软件全生命周期能力升级（如图 2 所示）。本阶段以构建支持软件开发、测试或运维等环节的 AI 模型为基础，将智能化技术逐步运用至软件工程各阶段，促进软件研发和运营效率的提升、质量的跃进、成本的降低。本阶段的发展特性较为明显，核心特点包括**智能化、数据驱动性、交互性、自适应和持续优化**。

图 2 软件工程 3.0 示意图

智能化，软件工程中各工具都将逐步实现对大模型等 AI 能力的调取和应用，以实现工具自身能力的提高，为智能化软件工程打下坚实基础。例如，将大模型与测试平台对接，提供测试用例生成、测试结果分析等能力，提高测试平台效能；将大模型与用户界面

（UI）测试工具对接，提供界面要素和组件等识别能力，提高 UI 测试准确率[3]。

数据驱动性，高质量数据是大模型成功的关键，“Garbage In，Garbage Out”仍然适用。在软件工程中，代码数据集、需求数据集、规范类文档数据集、测试用例数据集、日志数据集等，均将应用于大模型的训练、调优和推理过程，其质量高低直接影响推理结果，从而影响软件工程智能化能力带来的效果。例如，使用了精调代码数据集训练后的模型，相比基座模型在 MultiPL-E 上的评测结果绝对值提升了 8.2%，而使用了未精调的普通的代码加注释的数据集训练后的模型，评测结果反而降低了 5.5%[4]。

交互性，包括两个方面，一是大模型与人之间的人机交互，二是大模型与工具间的交互。**人机交互**将不断相互启发、相互促进，大模型可学习特定项目的上下文和研发人员的偏好，并根据更优的提示词，激发出更符合期望的推理结果，提高协作效率；**大模型与工具间交互**，将通过智能体（AI Agent）、编排等方式，实现大模型对各工具的调取和执行，以解决更加复杂的工程级问题，推动全生命周期智能化能力的提升。例如，2024 年 3 月 Cognition 公司发布的全球首个 AI 程序员 Devin，以及 8 月 Cosine 公司发布的全球最强 AI 程序员 Genie，对复杂的软件工程问题解决能力不断得到刷新。

自适应性，根据对工程级代码的更优理解能力，以及检索增强

[3] 中国信息通信研究院调研
[4] Magicoder: Empowering Code Generation with OSS-INSTRUCT

生成（RAG）等技术的辅助能力，大模型的自学习能力越发强大，这使得智能化软件工程对场景化的业务和数据具备更好的理解和适应能力，从而实现自主提升其推理性能和准确性。例如，代码大模型可在不同角色设定下展现良好的自适应性，MetaGPT 为大模型赋予了多个人设（产品经理、架构师、开发工程师等）后，其 HumanEval 评测结果为 85.9%，相比未设定人设的大模型评测结果绝对值提升了 18.9%[5]。

持续优化，通过建立数据飞轮和反馈闭环，根据用户反馈、场景化数据和监控数据对大模型进行持续改进，从而使软件工程智能化能力持续提升。例如，某银行的智能开发助手上线后，每周由研发部门的各团队提交优秀代码数据，对代码大模型进行定期的优化训练、测试和部署，从而持续保持和提升开发助手的能力[6]。

（二）智能开发发展现状

开发是软件工程全生命周期中的核心环节，开发人员大约有三分之二的工作时间与开发代码直接相关[7]，应用大模型助力软件开发具有得天独厚的优势。一方面开发人员对新技术的接纳度较高，能够更快地适应开发新范式；另一方面各类编程语言对业务的依赖度不是非常高，使得代码生成能力的普适性较强。因此软件开发是大模型率先应用落地的领域之一，Gartner 已将“AI 增强软件开发”列

[5] MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework
[6] 中国信息通信研究院调研
[7] https://thenewstack.io/how-much-time-do-developers-spend-actually-writing-code/

入 2024 年十大战略技术趋势之一[8]。

智能开发工具市场迅速发展，国内外百花齐放。国外，截止 2024 年 4 月微软的 GitHub Copilot 拥有 180 万付费订阅用户，近 60%财富 500 强公司使用 Copilot AI 工具[9]，GitHub Copilot 的市场占有率达到 64.5%[10]。亚马逊的 CodeWhisperer、谷歌的 Project IDX 等产品虽难以并肩，但依靠其强大的用户规模基础，仍具备较强竞争力。**国内**，智能化软件开发工具已发布近 40 余款[11]，如华为、阿里、百度等提供的智能开发工具产品，其核心功能较为同质化，但工具的性能、工程化能力、用户体验度、整体准确度等方面存在差异。

智能开发工具能力持续提升，应用行业更加多元化。智能开发工具重点聚焦于代码生成、代码补全、代码注释、代码检查、智能单测等能力。一方面代码大模型能力的持续提升为工具提供了更强的 AI 底座，根据智源大模型排行榜显示，榜首大模型在 HumanEval 数据集上的 Pass@1[12]从 2023 年 8 月的 24.4%提升至 2024 年 8 月的 81.1%[13]；另一方面通过 RAG、自学习、AI Agent 等技术的加持，为工具提供了更强的工程化能力，如微软的 Copilot WorkSpace 作为 AI 辅助的下一代开发工具，为工程级代码提供了更优的理解和生成能力。随着能力的持续提升，智能开发工具被越来越多行业所接受和应用，包括如互联网等科技行业的成熟落地，金融、电信、软件服

[8] Gartner《2024 十大战略技术趋势》
[9] https://github.com/features/copilot
[10]《中国软件技术发展洞察和趋势预测研究报告 2024》
[11] https://ai-bot.cn/favorites/ai-programming-tools/
[12] Pass@1 是指代码大模型为每个问题生成 1 个答案，其答案通过测试的比例。
[13] https://flageval.baai.ac.cn/#/leaderboard/nlp-capability?kind=CHAT

务等行业的逐步落地，能源、零售、教育、制造、物流等领域的试点试行，未来智能开发工具将为各行各业的软件研发提供强有力支撑。

应用需求快速增长，编码阶段提效显著。无论个人还是企业开发者，**使用智能开发工具的人数和频次越来越多。**根据 Stack Overflow 2024 年的全球开发者调研显示，76%的受访者正在或计划使用 AI 工具进行软件开发，使用 AI 工具的开发者数量占比从去年 44%提升至 62%[14]。根据 CSDN 2024 年的调查显示，48.6%开发者每天使用智能开发工具[15]。根据 Deloitte 报告指出，超过 60%的金融机构正在探索或已经实施了智能开发工具以加速创新。**智能开发工具的使用助力开发人员编码效率提升明显。**根据 GitHub 报告显示，使用 GitHub Copilot 的开发者编码速度提升 55%，编写代码量增加 46%[16]。根据中国信通院 2024 年初调研问卷数据显示，受访者在开发阶段使用 AI 工具的提效最为明显，人效提升达到约 40%。

（三）智能开发价值显现

在大模型应用浪潮中，软件开发的智能化转型正成为企业提升软件产品竞争力的关键因素，通过智能化能力的注入，为软件开发的价值提升带来巨大动力。

提升开发效率，降低项目风险。通过代码生成、代码补全和问答等能力，开发人员能够更迅速地编写高质量代码，显著减少手动

[14] 《the 2024 results from Stack Overflow's Annual Developer Survey》
[15] 《2024 中国开发者调查报告》
[16] The economic impact of the AI-powered developer lifecycle and lessons from GitHub Copilot

调试和错误修复时间。同时可降低项目对开发人员的依赖，通过智能开发工具的学习和记忆能力，一方面可辅助新的开发人员快速开发出符合项目需求和规范的代码，另一方面可帮助开发人员快速学习新的编程语言，从而减少人员流动带来的项目风险。根据 BIS 年度经济报告显示，智能开发工具对程序员的生产力提升超过 50%，且其中仅有小部分来自于代码的直接生成，而更多是通过与机器交互的过程激发了程序员的创造力[17]。

改善代码质量，提高产品稳定性。通过代码质量检查和智能单测等能力，开发人员能够快速进行代码验证和测试，及时发现并修复潜在的问题和漏洞，如代码缺陷、代码异常、代码风险等，从而帮助开发人员编写出更高质量的代码，降低软件发布后的故障率，提升软件的稳定性和性能。例如，GitHub Copilot 可帮助开发人员在编码过程中解决超过三分之二的漏洞[18]。

加速产品创新迭代，增强企业竞争力。通过提升开发效率和改善产品质量，企业能够更快地推出创新产品，抢占市场先机，从而在激烈的市场竞争中脱颖而出。一方面，智能化的软件开发流程使开发周期缩短，企业能够更灵活地适应市场变化，加快产品迭代速率；另一方面，智能开发工具的辅助使得开发人员能够从重复性的繁琐的低端编码工作中释放出来，从而拥有更多时间投入创新相关工作，提升企业创新能力。例如，2023 年 5 月 GitHub 首席执行官

[17] 国际清算银行（BIS）《2024 年度经济报告》

[18] https://www.synopsys.com/software-integrity/resources/analyst-reports/open-source-security-risk-analysis.html#UXexecutiveSummary

在全球网络峰会（Web Summit）上，演示 GitHub Copilot X 实现简单的贪吃蛇小游戏用时为 18 分钟，而 2024 年 8 月通过真实操作，使用 Deepseek 开发助手（V2）仅需 2 分钟左右，开发时长大幅减少，开发人员将有更多时间思考产品创新。

（四）智能开发落地挑战

随着智能化技术在软件开发领域的广泛应用，其带来的质效提升日益显著。然而，企业在推进软件开发的智能化能力建设和落地时，仍面临诸多挑战，涵盖组织、技术、应用落地及安全等多个层面。

组织变革和转型的挑战。软件开发的智能化转型首先带来的是组织变革，一是组织文化的重塑，其落地要求组织具备开放合作、持续学习和创新的文化氛围，促进跨部门沟通与协作，打破信息孤岛；二是全员认知水平的提升，自顶向下每位员工都应深刻理解智能化的潜力与挑战，客观且正确地了解大模型等人工智能技术及其作用；三是人才结构的调整与补充，由于数据治理及模型训练调优等过程需要专业技术人员参与，因此补充人才，或调整或融合 AI 团队和软件工程团队的结构，方能匹配能力建设和应用的需求。

模型技术迭代及与工具融合的挑战。当前智能开发能力建设的核心技术是大模型，因此围绕大模型面临着技术迭代的挑战。一是大模型的技术能力仍以较快速度迭代更新，企业需配备良好的维护更新机制，以保证智能开发工具的底座 AI 能力；二是智能开发能力

如何与现有软件研发工具集有效融合或改造，从而提升软件工程全流程能力水平。

产品选型与应用场景落地的挑战。企业需根据自身行业特点、业务需求及已有 AI 能力，选择适合的业务场景，合理规划落地路径和方案。然而，一方面现有代码大模型和智能开发工具数量繁多，如何建设或选择出一款合适的模型或工具，如何评价模型或工具的能力，均是落地时考虑的重点；另一方面不同业务场景的编程语言可能不尽相同，或代码数据集或规范不一致，如何选择最优场景试点落地，如何优化模型或工具使其满足多场景需求，同样面临挑战。

代码数据、模型和工具的安全性挑战。生成式代码可能会带来更多不可控的风险，因此企业应从数据、模型和工具多维度构筑风险防线，以应对安全挑战。第一道数据防线，面向模型训练和调优阶段所需的代码数据集，可能面临敏感代码数据的泄露、未经许可代码的训练、非法代码的推理等问题；第二道模型防线，面向代码大模型在推理和管理阶段，可能面临暴力攻击、非法套取、非法提问、敏感内容推理、被恶意使用生成恶意软件代码或攻击脚本等问题；第三道工具防线，对智能开发工具的输入输出进行检查和处理，并关注关联代码库、第三方软件开发工具的数据传输和集成安全，构筑最后一层安全围栏减少应用风险。

二、智能开发落地策略

随着代码大模型和智能开发工具如雨后春笋般出现，各行业陆续规划和试点将智能化能力有效地落地应用于开发环节。本章将首先总结智能开发落地遵守的主要原则，其次就落地关键步骤进行深入剖析，最后梳理当前行业的通用落地框架。

（一）智能开发落地原则

智能化软件开发落地是指应用方企业通过采购或自研等方式，构建智能化能力，包括**代码大模型能力**和**智能开发工具**能力，并将其应用落地于软件开发过程。作为大模型应用的重要场景，根据中国信息通信研究院对金融、软件、电信等行业关于智能开发落地实践的调查显示，智能开发能力的落地通常考虑以下主要原则。

目标导向原则，以企业战略定位为首要目标，落地策略和方案应在该目标指导下完成。如是否落地智能开发，落地范围预计多大，预期目标是什么，都应遵循企业组织级的人工智能战略目标。

因地制宜原则，从企业实际情况出发，清晰了解已有 AI 基础及成本预算，从而制定合适的策略。如企业自身的科技实力、人才储备、AI 基础设施能力、资金支持等情况，应作为落地基础进行考虑。

应用优先原则，从业务实际需求出发，明确亟需解决的问题，以及为业务赋能的目标。如软件开发过程中面临的最大问题是什么，最耗成本和资源的环节是哪些等。

标准化原则，参考行业标准开展能力构建，以保证数据质量、模型性能及工具可塑性。如代码数据如何处理如何配比，模型如何

调优，工程化能力如何建设，推理准确性应达到何等水平，均可以行业标准为依据，少走弯路，提高建设质量。

安全性原则，围绕能力建设全过程构建安全保障机制，降低新工具引入的风险，提升安全治理能力。如代码大模型推理安全、数据安全、安全围栏建设等。

持续改进原则，构建持续反馈机制和数据驱动流程，通过效能指标数据推动工具和流程的持续改进。如面临模型退化、新项目新代码数据等情况，应通过持续改进从而保证代码大模型的能力稳定性。

（二）智能开发落地步骤

落地智能化能力是一个全面且系统化的过程，尤其面向已成体系的软件开发过程，构建清晰明确的落地步骤，将为有效落地提供良好指引。智能开发落地过程可划分为自我诊断、方案设计、部署实施、持续优化四个关键步骤，如图 3 所示。

图 3 智能开发落地步骤图

开展多维度自我诊断，科学客观定位自身能力。诊断维度上，通过**应用场景诊断**，明确智能开发能力在企业内部的落地场景，帮助梳理当前需求和未来规划的需求；通过**技术能力诊断**，明晰企业在人工智能领域的优势和不足，明确是否具备足够技术储备开发或

维护代码大模型或智能开发工具的能力；通过**基础设施诊断**，深入了解企业现有算力资源、存储资源、数据资源等，帮助企业做好基础性准备；通过**安全可信诊断**，明确企业自身对安全可信的要求，如代码数据安全性要求、用户安全性要求、研发场景安全性要求等，从而便于制定安全保障机制。**能力定位上**，企业根据战略规划、基础设施、数据资产、人才储备、经费预算等方面的能力现状，并结合安全可信要求，综合研判自身已有智能化能力等级，可划分五个不同等级（如图 4 所示）。

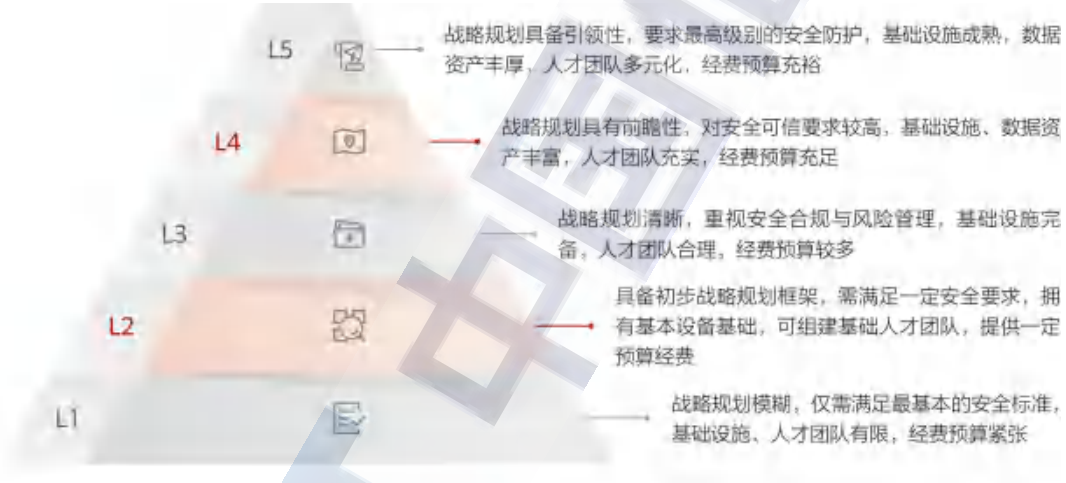


图 4 企业智能化能力自我诊断等级图

根据自我诊断及能力定位结果，选择合适的实施方案。L1 企业适宜采用低成本的智能开发工具 SaaS 服务，以便快速达到应用效果；L2 企业可采购软硬件集成的智能开发工具（如智能编码一体机），以实现智能开发能力的本地化部署，形成一体化解决方案；L3 企业可采购包含代码大模型的智能开发工具进行私有化部署，保障企业级代码资产的安全，并通过采购模型调优和升级等服务，保证大模型能力的稳定；L4 企业可采购解耦的代码大模型和智能开发工具，

或者选用高性能大模型训练微调成自有代码大模型，构建定制化的智能开发能力；L5 企业依托丰富的算力、数据、人才等基础，可考虑自主研发代码大模型，并依此构建软件工程领域的工程化能力，全面赋能软件开发流程，引领行业创新。

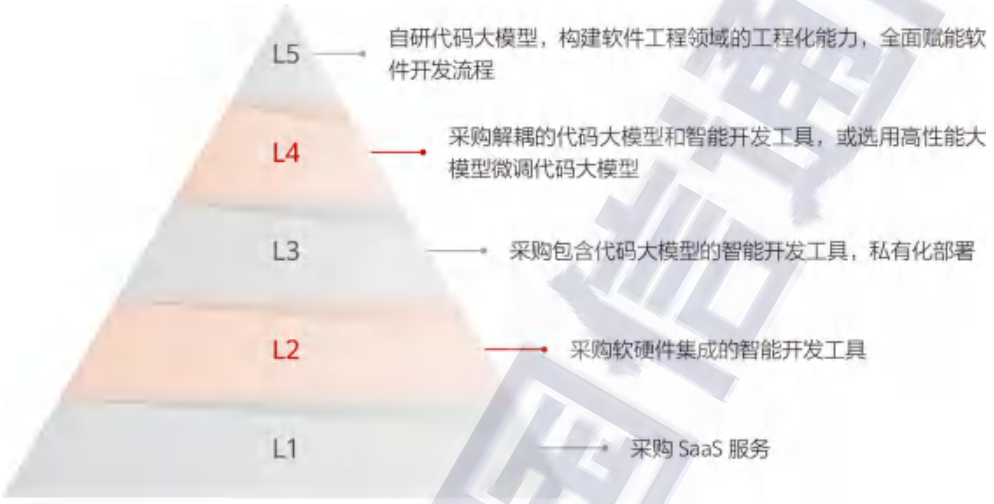


图 5 智能开发能力实施的多阶方案图

根据实施方案按计划部署和实施。一是明确项目的时间节点、资源配置、任务分工等，并制定风险管理计划；二是根据能力等级开展必要的软硬件采购与资源配置，确保相关工具和平台能够满足落地要求；三是开展代码大模型和智能开发工具的部署、评估和测试，确保其准确性、稳定性、安全性和可靠性；四是对使用人员进行培训并提供技术支持，选定试点项目或团队；五是根据试点结果进行优化调整，并逐步宣贯和拓展至更多团队。

根据实施效果持续优化和改进。建立有效的指标体系和监控机制，一方面持续地自动化监测大模型推理效果和应用成效，如推理准确率、代码采纳率、代码修复率、代码生成占比等指标；另一方

面定期收集用户反馈，根据数据分析结果明确问题和优化方向。构建大模型维护更新机制，通过获取和生成高质量代码数据集，开展模型的定期训练或调优，持续优化代码大模型质量。

（三）智能开发落地框架

智能化软件开发的能力落地框架（如图 6 所示）由三部分组成，分别包括模型层、服务层和应用层，应用层的能力是模型层和服务层能力的组合体现。

图 6 智能开发落地框架示意图

模型层以各类 AI 模型为主体，为智能开发提供 AI 底座能力。本层建设目标是在基座大模型的基础上训练调优生成代码大模型，而代码大模型普遍由 2 个大模型组成，分别是代码补全大模型和研发问答大模型。代码补全大模型的任务是代码生成与补全，由于其使用频率最高，通常设计为参数规模较小但推理速度较快的如 3B、7B、13B 等较小型模型，以匹配编码速度；研发问答大模型的任务是应对更为复杂的代码解释、代码检查和研发问答等各类任务，通常设计为参数量更庞大的如 65B、72B、128B 等较大规模模型，以满足复杂的软件开发需求。

服务层依托 AI 底座能力，运用更多技术手段增强或调度大模型能力。通过**模型调度**，针对工具层不同的操作精确匹配调度相应模型，确保模型资源的合理分配与高效利用；通过**提示词封装**，对用户常用功能设计提示词模板，丰富用户问题的表述，使模型能够更全面、准确地理解用户意图，提供更精确的推理结果；通过 RAG 将内部数据形成知识库注入推理过程，提升推理结果更符合项目特性；通过 AI Agent **辅助**，以感知和记忆能力将复杂研发问题进行拆解，以决策和执行能力调用各类工具或组件，提升问题解决能效。

应用层以用户为核心提供各类智能开发功能，主要包括智能编码和开发者辅助两大类功能，并将其集成于 IDE（集成开发环境）等工具中为用户提供直接服务。应用层能力一方面基于服务层对大模型能力进行了加强，如对工程级别的代码理解和生成能力更强，能够智能识别 IDE 当前打开的多文件内容，并将其作为大模型输入的上下文，提升跨文件感知能力；另一方面应用层为工具打造了更优异的用户体验，如兼容适配主流 IDE、操作系统以及常见自然语言等，建设自定义配置相关功能，优化输入输出性能，建立用户反馈机制等，满足开发者需求并提升体验度。

三、智能开发核心能力建设

在软件开发过程中，无论是企业开发者还是个人开发者，都面临重复书写代码、缺少编写单元测试用例和代码注释的习惯等问题，从而导致编码效率低下或存在质量隐患。因此，在智能开发能力建设过程中，可优先构建核心能力，如代码生成与补全、单元测试用例生成、代码转换与优化等能力可提升编码效率，代码解释与注释生成等能力可增强代码可读性，代码检查与修复等能力可提高代码质量。本章重点对各核心能力的构建进行剖析[19]。

（一）代码生成与补全

代码生成与补全是指根据自然语言描述或代码上下文，自动生成或补全代码的能力。通过代码生成与补全，可帮助开发人员提高编程效率、降低开发门槛。

1.关键能力

● 代码生成能力：基于自然语言描述，生成有效的片段级或文件级或工程级代码；

● 代码补全能力：根据代码上下文，补全有效的引用、类/方法/函数名称、行级或片段级或文件级或工程级代码；

● 多维兼容能力：支持多种主流（或所需）编程语言和开发框架；

● 代码质量能力：生成和补全的代码满足基本的质量规范要求

[19] 本章核心能力的阐述以中国信息通信研究院联合发布的《智能化软件工程技术和应用要求：第 1 部分 代码大模型》和《智能化软件工程技术和应用要求：第 2 部分 智能开发》为参考依据。

和安全性要求。

2.扩充能力

● 根据多模态输入生成代码能力;

● 知识库接入能力，可提供更符合企业内部业务逻辑或规范的代码或用例;

● 第三方工具对接能力，如与版本管理工具的对接，支持代码合并冲突时提出解决和修复建议。

3.重点和难点

● 推理效率提升：代码生成与补全是使用频率较高的功能，且流式生成对推理速度有较高要求，因此优化模型推理效率是本能力的建设重点;

● 高质量数据集构建：在代码大模型训练或调优过程中，高质量代码数据集的构建成为重点，其中包括对各类代码的清洗、处理、安全过滤、标注等;

● 提示词工程优化：基于意图识别对提示词模板进行调优，辅助推理效果的提升;

● 上下文理解能力：准确理解代码上下文，如变量的作用域、函数的调用关系、跨文件间代码逻辑等，对提升推理准确度有明显作用;

● 模型持续改进：通过代码采纳与拒绝等历史数据构建数据飞轮，对持续改进和维持推理质量有明显作用。

实战经验参考：

采用三“码”合一方案（将训练态 SFT 代码语料、检索态 RAG 代码语料、推理态补充代码语料组装）优化 Prompt 模板，形成统一的 Prompt 模板，构建成一个一致的训练和推理逻辑，通过将训练阶段与推理阶段的格式保持一致，可明显提升代码模型在垂直领域上的适应性和鲁棒性，使其在不同场景下保持较一致的性能。

图 7 三“码”合一方案图

（二）单元测试用例生成

单元测试用例生成是指根据输入的代码，自动生成用于测试该代码的单元测试用例。通过单元测试用例生成，可帮助开发人员减少编写单元测试用例的时间，并提高单元测试覆盖率，提升代码质量。

1.关键能力

- 用例生成能力：根据函数级或文件级代码，生成单元测试用例；
- 用例质量能力：生成的用例满足基本的质量规范要求，且具

备所需覆盖度;

● 多维兼容能力：支持多种主流（或所需）单元测试框架。

2.扩充能力

● 生成的单元测试用例包含较为清晰的注释，且执行性能较好;

● 生成的单元测试用例可自动对部分外部依赖进行打桩，且打桩用法合理。

3.重点和难点

● 结合传统代码分析技术：加强大模型对代码结构和逻辑的理解，如对函数、类和模块级的接口，以及各类控制流和数据流进行分析和识别，可帮助提升单元测试用例生成的质量;

● 结合多样性测试技术：支持随机测试、符号执行、模糊测试等，通过组合不同测试技术，提高单元测试质量和覆盖度;

● 单元测试用例生成能力的评估：通过编译正确性、覆盖充分性、错误检出率等指标对推理结果进行综合评价，是本能力建设难点。

实战经验参考：

采用大模型与传统软件分析等技术相组合的方式提升单元测试用例生成效果，传统分析工具通过整体分析被测工程，获取被测函数的依赖、测试场景等，生成用例初始化部分，而大模型生成用例的函数部分可结合软件分析的后处理算法，大幅度提高生成用例的编译通过率和覆盖率。

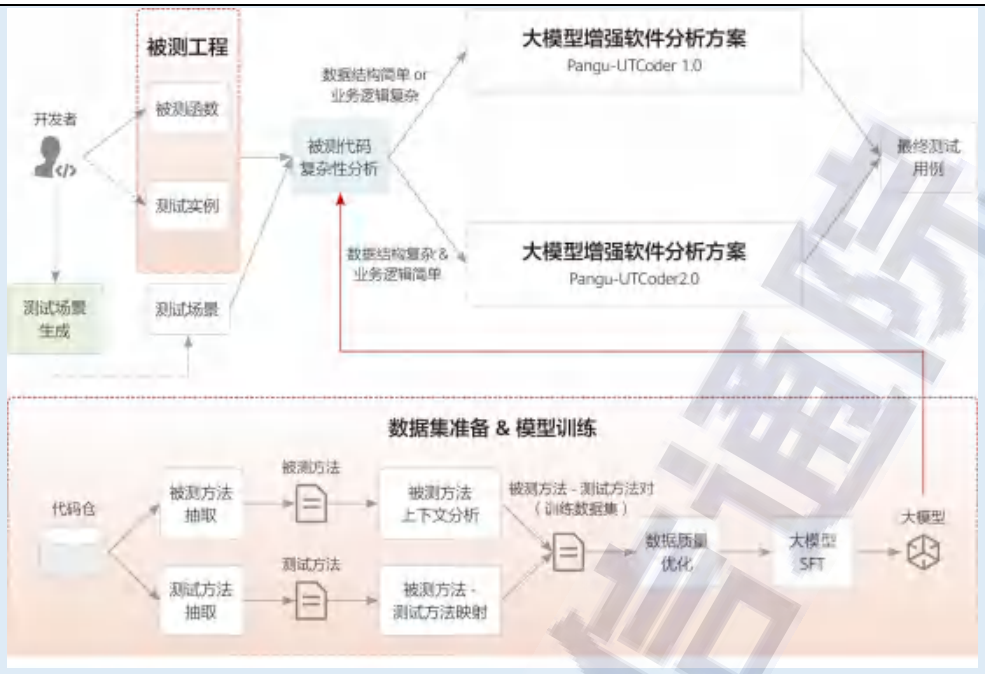


图 8 单测生成实战方案示意图

（三）代码转换与优化

代码转换与优化是指在保持代码逻辑和功能不变的基础上，对输入的代码进行不同语言或不同框架之间的转换，或特定方向的优化。通过代码转换可帮助快速理解代码，通过代码优化可帮助解决代码中存在的规范性、性能等问题，降低开发人员的技术门槛。

1.关键能力

● 代码转换能力：根据给定的代码，转换成不同编程语言、不同编程范式或不同框架等维度的代码；

● 代码优化能力：根据给定的代码，提供规范性、性能和复杂度等维度的优化建议；

● 转换和优化质量能力：转换和优化后的代码满足基本的质量规范要求，且保留了原代码功能。

2.扩充能力

● 知识库的接入能力：使转换和优化后的代码更符合企业内部业务逻辑或规范；

● 对多文件的工程级代码进行转换和优化。

3.重点和难点

● 集成优化算法：集成各种优化算法，如循环优化、内存访问优化和并发执行策略，提高代码优化的性能和质量；

● 结合 RAG 的代码转换：检索历史代码以及外部依赖信息等，使转换后代码更符合项目需求。

（四）代码解释与注释

代码解释与注释是指对输入的代码进行解释或注释的生成。通过代码解释，可帮助开发人员快速理解大量代码或不熟悉的代码，通过代码注释，可提高代码的可读性和可维护性，帮助开发人员生产和维护更符合规范的代码资产。

1.关键能力

● 代码解释能力：根据给定的代码，提供基本准确的解释内容；

● 代码注释能力：根据给定的代码，提供基本准确的注释内容；

● 注释质量能力：代码的注释内容满足基本的注释规范要求。

2.扩充能力

● 知识库的接入能力：使代码的解释和注释内容更符合企业内部规范；

● 对变更的代码进行解释。

3.重点和难点

● 语义理解及关键词识别：准确理解代码语法、语义，并识别出关键代码元素和结构，是本环节的难点。

实战经验参考：

设计基于 RAG 的上下文感知的代码理解流程，协同模型能力与业务领域知识（如设计文档、开发规范等）、工程上下文调用信息等，并根据开发者实时行为，精确检索上下文，提高代码解释与注释的生成准确率。

图 9 基于 RAG 的代码理解流程图

（五）代码检查与修复

代码检查与修复是指对代码进行相关问题的检查，并根据检查出的问题进行自动修复。通过代码检查，可帮助开发人员发现代码中存在的问题，如静态缺陷、运行时错误、安全漏洞、架构问题等，通过代码修复，可帮助开发人员解决问题并提高代码质量。

1.关键能力

● 代码检查能力：根据给定的代码，进行代码规范性问题、代码异味、代码语法错误、代码逻辑错误、代码安全漏洞等问题的检查；

● 代码修复能力：根据检查出的问题，提供代码修复建议及修复后的代码；

● 检查和修复的质量能力：代码检查的错误检出率和误报率达到要求，修复后的代码需满足基本的质量规范要求，且保留了原代码功能。

2.扩充能力

● 对多文件的工程级代码进行问题检查和修复；

● 代码检查规则的自定义能力；

● 对污点类问题进行检查，可跟踪和分析污点数据（或输入）在代码中的流动情况，定位可能污染的关键位置。

3.重点和难点

● 提示词优化：制定代码检查与修复提示词模板，支持检查风格类问题、质量类问题、安全漏洞、性能缺陷、逻辑缺陷、行业规范等多种问题的提示词；

● 降低误报率：传统漏洞扫描工具误报率较高，基于大模型的理解生成能力降低漏扫误报率成为重点和难点，

（六）研发问答

研发问答是指利用自然语言处理、信息检索等技术进行研发相关问题的解答，帮助开发人员提供问答辅助能力，提高研发和创新

能力。

1.关键能力

● 问答能力：根据对话可理解研发相关问题，并提供相关解释、建议、教程或示例代码，较为准确地回答用户提问；

● 代码搜索能力：根据关键词分析或语义理解等，快速、准确地查找和定位所需代码；

● 多轮对话能力：可通过多轮对话，不断优化答案以更加准确地满足用户需求。

2.扩充能力

● 可追溯能力：可明确推理答案的参考来源；

● 对工程级代码库的理解和分析能力，并基于其进行问答。

3.重点和难点

● 多轮对话感知：理解和保存用户多轮问答交互结果，作为下次提问上下文，以便提供更加准确的答案；

● 交互式反馈机制：提供答案反馈机制，以便不断优化答案的准确性和相关性。

四、智能开发使能能力建设

智能开发落地过程中，通过核心能力建设可实现各项关键需求，但在持续改进迭代环节，为持续维持或提升代码大模型推理准确性，构建代码数据集、调优代码大模型、评估代码大模型或智能开发工具等使能能力同样重要。本章将围绕数据、模型、评估、安全和相关工程化能力展开分析其建设要点。

（一）代码数据处理能力

代码大模型所需的代码数据集包括纯代码数据，以及代码和文本的混合数据。通过有效地管理和处理代码数据集，将其处理成可直接被用于模型训练调优的高质量代码数据集，为代码大模型提供基础支持，**应用于企业在建设和维护优化代码大模型的过程**。

图 10 数据处理流程图

数据清洗

数据清洗用于过滤低质代码数据，提升数据质量。代码数据清洗主要包括数据过滤、数据去重、敏感数据处理、许可协议（License）过滤等环节。不同数据可采用不同处理方式，**开源代码数据**可进行所有方式的清洗，**私有代码数据**可进行数据过滤、数据去重、敏感信息过滤等方式的清洗。

数据过滤是筛选、排除或提取数据集中特定部分，过滤方法主要包含三种。**基于规则过滤**是采用基于正则表达式、专家规则、点

赞数量等规则的过滤器，识别并删除空白文件、自动生成的代码数据、有效信息过少的数据等；**基于程序分析过滤**是采用语法解析器和抽象语法树过滤存在语法错误的代码数据，采用规范检查规则或工具过滤掉不符合规范和不安全的代码数据；**基于模型过滤**是采用标注的高质量和低质量数据，训练质量模型分类器，对代码数据质量进行评估和过滤。

数据去重是从数据集中消除冗余的重复记录，仅保留具有唯一性的数据条目，提升数据质量和存储效率。**去重方式**包含精确去重和近似去重，**去重策略**包含代码的文件级去重和仓库级去重，有效压缩数据规模。

敏感信息处理是指识别并移除或替换数据中的敏感信息，包含代码中的密钥、令牌、个人身份信息、URL、IP 地址、机密算法等，可采用正则表达式、深度学习与机器学习、人工审查等技术实现敏感信息过滤。

License **过滤**是检验并去除开源代码中不允许使用的数据。开源代码 License 包含允许使用、不允许使用和无规定，为保证开源数据使用的合规性，可使用 ScanCode-Toolkit 等许可检测工具，过滤掉非法数据，将允许使用和无规定的开源数据用于模型训练调优。

数据增强

数据增强是通过对现有代码数据执行有意义的变换和扩充，生成新代码样本，增加数据集的多样性和丰富度，提高模型泛化能力和鲁棒性。**注释增强**是通过人工或自动化方式在代码的方法定义、

循环、条件语句等位置新增注释，增强代码可读性，例如开源代码可采用 AI 自动化生成注释，私有代码可采用人工注释以得到更好效果；**上下文增强**是在准备微调数据时，将代码数据扩充增加相关联的上下文、依赖、外部对象和方法等信息；**数量及多样性增强**用于提升代码数据覆盖度减少过拟合，常用方法如 Self-Instruct、Evol-Instruct、OSS-Instruct 和 Self-OSS-Instruct 等。

数据检查

数据集的质量决定了大模型的性能，代码数据集在处理过程中应满足一定要求。**数据集质量方面**，应满足语法正确、编码规范、无安全漏洞、无缺陷、无敏感信息、无重复代码、包含注释等要求；**数据场景覆盖方面**，应选择高质量代码，只包含代码文件，无需其他配置文件，同时只包含核心代码，弃用边缘代码等要求。

（二）代码大模型优化能力

对代码大模型进行优化的目标是维持或增强其推理能力，主要包括模型调优及调优后模型部署，**应用于企业使用专有数据集对代码大模型进行专项调优的过程。**

模型调优

模型调优是在特定任务或领域上进一步训练或调优模型，以优化其对这一任务或领域的理解和推理能力。代码大模型调优通常采用有监督微调和强化学习等方法。

图 11 代码大模型优化过程示意图

有监督微调亦称指令微调，是指通过构建指令格式的实例，以有监督方式对大模型进行微调。指令微调可辅助代码大模型提升推理能力，从而展现出泛化到未见过的任务的卓越能力。指令微调的前提是构建问答对数据集，同时根据实际任务（如代码生成、代码解释等任务）配置数据配比不断完善数据质量。有监督微调包含全量参数指令微调和高效微调，当算力充足且数据集较多的情况下，通常采用全量参数指令微调的方式，从而达到更好微调效果，但算力不足时，可采用更高效的微调方法，如 LoRa、Adapter、Prefix-tunning 等。

强化学习的目标是强化代码大模型在某些准确率不高的任务下的能力，可通过人类反馈对推理质量进行评估和排序，并依据其反馈结果开展学习优化，从而持续提升推理生成的质量。强化学习通常选择基于人类反馈的强化学习算法框架，并利用近端策略优化算

法进行优化，或进行直接偏好优化。强化学习需要特殊构建数据集，一方面可以使用大模型生成或人工构建数据对，针对不同开发场景进行增强，从而提升模型的通用研发能力，如针对某类代码框架或前端开发、后端开发等不同场景构建正负数据对；另一方面则需要结合测试反馈或用户反馈来针对模型能力较弱的场景进行增强，如在服务运行阶段，通过后台监控白名单用户行为，如点赞点踩、代码采纳情况等，收集好回答示例和坏回答示例进行数据构造，从而修正模型偏好。

模型部署

模型部署是指将调优好的模型部署至服务器上供用户使用的过程。模型部署时普遍考虑三点，一方面开展模型转换，根据部署资源将模型转换成所需形态，另一方面通过优化算法提升大模型推理速度和吞吐量，提高计算资源的利用率，最后根据需求、使用场景选用并生成合理的模型调用方式，如接口（API）或插件方式。

但在代码大模型部署时，还应特别考虑**前后处理**工作，当用户提交请求时，需对用户所在项目工程进行软件分析，并将分析信息和用户输入一并输入给代码大模型；当代码大模型推理完成后，需对推理结果进行后处理，如在函数结束位置提前停止、复读检测、幻觉检测等，以增强代码大模型推理生成的准确性。

（三）智能开发能力评估

为验证代码大模型及智能开发工具是否满足需求，通过构建评估数据集和指标对其开展全面评估，**应用于企业在采购、建设及维**

护优化智能开发能力的过程。

评估对象

智能开发能力评估包括**代码大模型能力评估和智能开发工具能力评估**。代码大模型能力评估聚焦于大模型的理解和推理能力，通过采用专业评估数据集对代码生成的准确性、单元测试的覆盖度、代码检查的错误检出率等指标进行评估。智能开发工具能力评估聚焦于平台能力、功能丰富度和专项化能力等多个维度，综合衡量呈现给用户的工具的性能和工程化能力，其框架如图 12 所示。

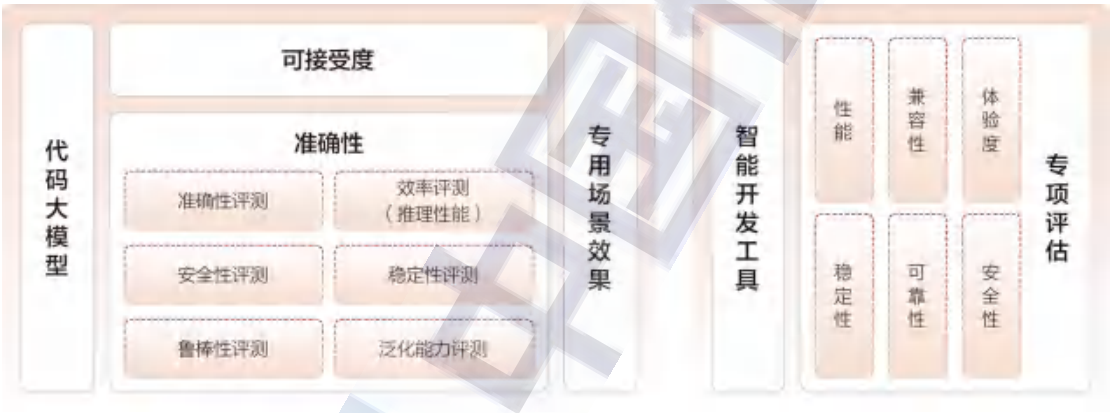


图 12 智能开发能力评估架构图

评估内容

代码大模型评估旨在衡量准确性和可接受度，智能开发工具评估旨在衡量准确性和可接受度，以及工具的功能丰富度、工程化能力以及体验度。**准确性评估**是通过预设的关键指标对工具在特定场景下的输出进行客观分析（如表 1 所示），考察输出的准确性和逻辑合理性；**可接受度评估**是通过专家评审等主观分析，考察输出是否达到行业标准，主要包括输出代码的符合性、可读性、规范性、完整性、合规性和安全性等；**工具专项评估**是通过对工具的功能和

性能的评估，考察其应用层面的灵活性、可扩展性、兼容性、安全性、稳定性等，以及功能丰富度和使用过程的体验度等。

表 1 准确性评估场景与指标参考清单

评估场景		关键指标
代码生成与补全		Pass@k、相似度指标 CodeBLEU
单元测试用例生成		编译通过率、分支覆盖度、行覆盖度、UT 通过率
代码转换与优化		行命中率、命中 50%比率
代码解释与注释	代码解释	关键字命中率
	代码注释	指令遵从度（中文回答比率、修改代码比率）、注释比例、格式正确率、语义相似度
代码检查与修复	代码检查	错误检出率、误报率
	代码修复	错误修复率、错误引入率、Pass@k
研发问答		语义相似度、关键词命中率、上下文一致性、稳定性、非复读率、问答语言一致性、多轮对话成功率

评估方法

智能化开发能力的评估方法涉及多种方法。一是**自动化评估方法**，面向准确性评估，通过构建评估框架、指标体系和自动化工具，使用数据集开展相关客观指标的评估；二是**裁判模型评估方法**，面向可接受度评估，通过选取具有高精度和相关领域知识的大模型作为“裁判”，设置好详细的提示词，对输出进行评估；三是**领域专家评估方法**，面向可接受度和工具专项评估，通过行业专家对照预期结果评估工具的实际性能及工程化能力，提供主观判断。

实际评估过程中，通常采用多种方法可形成互补的评估机制，同时考虑评估效率和准确性，从而为持续提升智能开发能力提供有效方法。

（四）智能开发安全能力

智能开发的安全能力是指从代码数据、代码大模型、智能开发工具等层面构建的安全治理体系，目的是减少使用智能开发能力时

带来的风险，主要包括数据安全、模型安全和应用安全，**应用于企业落地智能开发能力的全过程**。

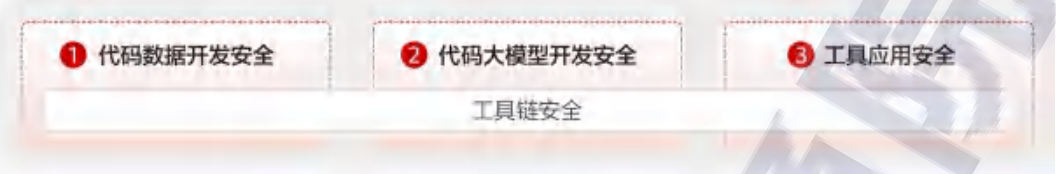


图 13 安全能力体系架构图

1.数据安全治理

数据安全治理应考虑从数据源头到应用全过程的安全可控，以保证用于代码大模型训练和调优的代码数据集安全。**数据采集阶段**确保从可信代码仓库获取源数据，防止版权风险；**数据处理阶段**通过清洗与过滤等技术，精准剔除敏感代码，并对代码数据进行分类与标注，便于后续的差异化管理和使用；**数据安全评估阶段**作为质量门禁，对拟入库并共享发布的代码数据集进行审查；**数据管理阶段**可通过存储层加密等手段确保代码数据存储安全，以版本控制和访问控制等手段确保代码数据可追溯，并通过定期安全评估开展数据健康状况监控，及时处理安全风险。同时，承载数据采集、处理等全过程的工具链，需具备任务隔离和严格的访问控制机制，确保其自身的安全，保证代码数据开发过程的安全可控。

图 14 数据安全治理流程示意图

2.模型安全治理

模型安全治理应考虑代码大模型在开发、管理及运行阶段安全可控。**开发阶段**，即代码大模型调优阶段，首先应保证数据和基础模型的来源可信，同时对模型进行安全标识和分类分级；其次通过相关数据集对代码大模型开展安全可信评估，确保模型推理结果满足基础安全要求。

管理阶段，应对代码大模型进行安全性存储和加密传输，并加强模型版本管理实现模型溯源，确保合理的权限约束。同时，对于承载代码大模型开发全过程的工具链，需具备任务隔离和严格的访问控制机制，确保其自身的安全，保证代码大模型开发过程的安全可控。

运行阶段，面向代码大模型推理服务，一是确保推理接口安全，通过通信加密协议、安全访问机制等方式，防止未授权访问及 API 滥用等情况；二是确保提示词数据安全（如提示词不被篡改等）、知识库数据安全，并对 RAG 检索的知识附带来源，降低运行时推理风险；三是对推理资源做合理分配，防止推理资源的过度消耗；四是确保运行环境安全，防止模型泄露或被篡改，通过实施沙箱隔离及强化访问控制等措施，确保只有经过认证的用户和应用程序才能访问代码大模型。

图 15 模型安全治理架构图

3.应用安全治理

应用安全治理是对基于代码大模型的智能开发工具的安全保障，一是关注应用安全防护，如部署 DDoS、WAF 等网络安全防护技术；二是强化敏感信息检测机制，对用户输入及模型输出内容实施敏感信息检测和内容安全防控，以防范敏感信息及关键资产的泄露，并保障内容合法性和适宜性；三是保证用户数据隐私安全，用户数据应严格按照授权内容存储，或者不存储，确保私有数据安全；四是可考虑模型领域隔离，仅限定于研发领域的应用，不提供其他无关领域的内容生成等功能；五是健全安全审计机制，通过系统请求和操作等审计日志的管理，确保智能开发工具使用安全性。

图 16 应用安全治理框架图

（五）其他工程化能力

为进一步提升代码大模型及智能开发工具对问题的理解能力，以及对企业私有化知识的习得能力，提示词、RAG、Agent 等各工程化能力扮演着重要角色，可在不改变代码大模型的前提下达到更好推理效果，本节重点围绕提示工程和 RAG 展开分析。

1.提示工程能力

提示工程是用户与代码大模型交互的主要桥梁，通过设计和使用提示词可优化代码大模型能力，对提升模型性能、增强模型可控性、扩展模型能力等方面具有直接作用。面向智能开发领域，提示工程主要被用于两个阶段，**一是代码大模型训练调优阶段**，使用提示词构造训练样本，通过预训练、指令微调、反馈对齐等方式训练大模型，从而提高其对特定指令的遵从和理解能力，或者赋予模型连续对话、工具调用等能力；**二是代码大模型推理阶段**，通过提示词为大模型提供解决问题所需的必要信息，包括但不限于任务背景知识、必要的上下文信息、期望输出格式等，从而提升大模型正确生成答案的概率。提示工程的典型应用包括可感知上下文的代码补全、围绕代码仓的问答、智能解决问题单等场景，需通过提示词将待生成代码中可能用到或可能参考的代码片段、相关文档、项目结构、错误堆栈等信息告知代码大模型。

（1）提示词要素

提示词内容通常可包含以下元素。**系统设定**，涵盖人设/角色、基本原则、安全边界等重要内容；**任务提示**，明确指出大模型需要

完成的具体任务或目标、输入格式、输出要求等；**用户指令**，包括当前要回答的具体问题的背景、描述、详细要求等；**示例样本**，提供若干具体的输入-输出示例；**上下文输入**，提供当前任务所需要的隐式上下文。除此之外，还可以根据具体场景附加其他元素，如检索增强内容、可调用工具列表、行动与反思指令等。业内使用的提示词模版较为典型的包括 BROKE、CRISPE、ICIO、APE 等。

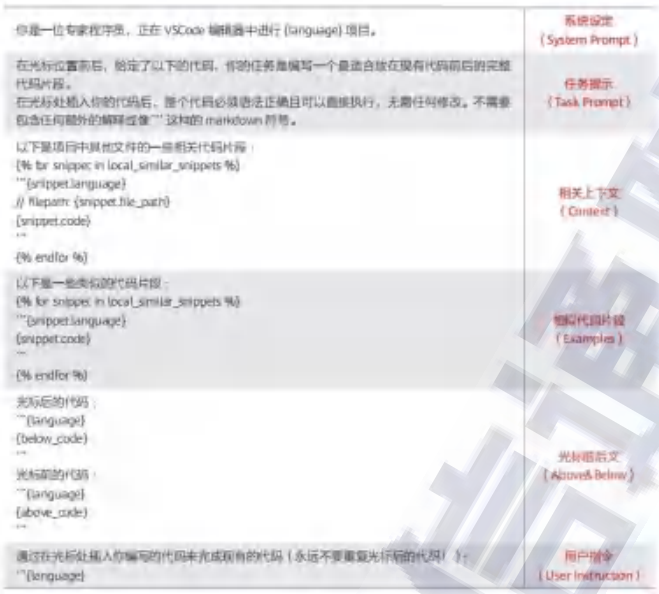
（2）提示词构造方法

提示词构造是一个复杂且多维的过程，不仅涉及设计和编写提示词本身，还可借助其他工程化方式动态构造和组装提示词。**手动编写提示词**是一种基本且直接的提示词构造方法，由用户或代码大模型服务提供方，根据各自的需求和任务背景直接手动编写，本方法简单直接但不够灵活且效率较低，适用于简单场景；**动态组装提示词**是指在运行时基于模板生成提示词的构造方法，可通过软件分析、自然语言处理、大模型推理等技术进行动态填充，如代码生成任务中的上下文、代码重构任务中的项目结构等。

（3）提示词实例参考

在代码补全任务中，提示词通常如表 2 所示。通过该提示词，系统将首先设定代码大模型扮演一个编程专家角色，激发其在编程领域的特定知识；其次通过任务提示告知代码大模型需根据光标处的上下文进行补全；最后要求代码大模型根据示例样本，按照用户指令进行代码插入，并不可生成重复代码。通过该提示词可提升代码续写能力。

表 2 提示词示例 1



你是一位专家程序员，正在 vSCode 编辑器中进行 {language} 项目。	系统设定 （System Prompt）
在光标位置前后，给定了以下的代码，你的任务是编写一个最适合放在现有代码前后的完整代码片段。 在光标处插入你的代码后，整个代码必须语法正确且可以直接执行，无需任何修改。不需要包含任何额外的解释或像 ``` 这样的 markdown 符号。	任务提示 （Task Prompt）
以下是项目中其他文件的一些相关代码片段： {% for snippet in local_similar_snippets %} ```{snippet.language} // filepath: {snippet.file_path} {snippet.code} ``` {% endfor %}	相关上下文 （Context）
以下是一些类似的代码片段： {% for snippet in local_similar_snippets %} ```{snippet.language} {snippet.code} ``` {% endfor %}	相似代码片段 （Examples）
光标后的代码： ```{language} {below_code} ``` 光标前的代码： ```{language} {above_code} ```	光标前后文 （Above&Below）
通过在光标处插入你编写的代码来完成现有的代码（永远不要重复光标后的代码！）： ```{language}	用户指令 （User Instruction）

在代码解释任务中，提示词通常如表 3 所示。通过该提示词，系统将首先设定代码大模型扮演一个编程教练角色，激发其在教学领域的特定知识；其次通过任务提示告知代码大模型在解释时所需要使用的语言以及输出格式；最后根据用户指令明确需要解释的代码片段及其所在文件，并获取上下文信息进行代码解释，通过该提示词可提升代码解释的效果。

表 3 提示词示例 2

你是一位世界级的编程导师。你的代码解释完美地平衡了高层次的概念和细节。你的教学方法确保学生不仅理解如何编写代码，而且还能把握指导有效编程的潜在原理。 当被问及姓名时，你必须回答“AI 编程助手”。 请仔细并严格按用户的要求执行。 你的专业知识严格限制在软件开发主题上。 对于不涉及软件开发的问题，只需提醒用户你是一个 AI 编程助手。	系统设定 （System Prompt）
用户使用的集成开发环境（IDE）为 Visual Studio Code，它有一些概念，如编辑器可以打开文件，支持集成单元测试，一个输出窗格显示运行代码的输出，以及一个集成的终端。 当前活动的文档是用户正在查看的源代码。 你只能在每次对话回合中给出一个回复。 回答的本地语言是 {locale} 在你的回答中使用 Markdown 格式。 确保在 Markdown 代码块的开头包含编程语言的名称。 避免将整个回复包裹在三个反引号内。	任务提示 （Task Prompt）
文件 {filename} 中被选中的代码： ```{language} {selected_code} ```	被选中代码 （User Selection）
相关的函数实现： ```{language} // filepath: {filepath} {relevant_code} ```	相关上下文 （Context）
为上面的代码编写解释，以段落形式呈现。	用户指令 （User Instruction）

2.RAG 能力

检索增强生成（RAG）是指通过语义相似性计算从外部知识库中检索相关文档块，以增强代码大模型能力，减少幻觉并提高对专业知识的快速习得能力。面对私有代码库检索、研发问答知识追溯、生成的代码需符合内部开发规范、生成内部开发文档等情况，RAG 可提供较好的解决方案。

RAG 落地过程中，通常包括数据索引服务、向量化模型服务、向量数据库服务和搜索服务四个主要环节（如图 17 所示）。

图 17 RAG 落地流程示意图

数据索引服务是指将知识进行向量化并存储于数据库的过程，其中采用不同索引构建方式处理文本和代码数据是核心要点，文本的拆分处理可选择固定字符数分块、递归分块、语义分块和特定格式分块等方式，代码拆分处理时，对于常规长度的函数体可选择整体向量化而不做拆分，对于少数超大超长函数体既可采取自动截断也可按行进行切分；**向量化模型服务**可选择 BGE(Bi-Encoder for General Embeddings) 、 M3E(Multimodal Multilingual Multitask Embeddings)等模型，通过部署容器化并对外暴露端口，提供向量化服务；**数据库服务**使用向量数据库存储向量化之后的数据，可选择

开源数据库如 Milvus，Faiss，Weaviate，Qdrant 等，并通过减少向量大小、缩小搜索范围等方式提高数据库检索效率；**搜索服务**是向量化模型服务和数据库服务的桥梁，一方面用于将查询请求转换为向量表示，并在向量数据库中搜索相似项，另一方面对于搜索结果需根据不同使用场景设置不同的相似度阈值，确保将高质量结果反馈给代码大模型作为输入的一部分。

五、智能开发落地案例分析

各行业积极探索智能开发落地之道，并在部分行业的应用成效显著。本章将对智能化软件开发在云服务、软件服务业、电力、金融及制造等行业的落地案例进行阐述和分析。

（一）云服务行业案例

某大型云服务企业云服务研发过程遵循 DevOps 最佳实践，一方面该企业产品需要基于业界前沿方法和技术不断提升研发效率和产品发布速度，以响应激烈的市场竞争并保持领先；另一方面作为公司研发工程能力的排头兵，需要优先验证和试点业界领先的智能化软件开发能力，为公司其他产品线更大规模的探索和应用打好前阵。该企业在落地前存在如下问题和需求。

1）建立全新的智能化研发工程体系：面向全新的智能化研发范式，在流程、组织、工具等方面需要尽快通过试点探索形成新的工程体系，以牵引云服务研发生产力的持续提升；

2）保持研发生产能力先进性：面对激烈市场竞争，如何通过 AI 技术升级传统研发体系，保障软件研发生产力持续领先；

3）软件研发流程中仍存在高能耗业务场景：问题包括如研发人员难以全盘掌握和快速精准获取海量知识；单元测试及接口测试脚本人工输出工作量大，效率低难以匹配业务代码的增长诉求；对高价值代码问题检视率低，质量不可控等。

1.解决方案

该企业围绕“设计-开发-测试-协同”四个核心研发阶段，基于“高

能耗”活动（如高消耗、高知识门槛）和“AI 适用性”（业务成熟度、数据成熟度、技术成熟度）”两个方面共 5 个维度，识别出各研发阶段中 AI 实践价值场景。而“开发”场景为价值最高，因此优先落地覆盖了编码的“读-写-调-测-检”全流程。

图 18 某云服务企业案例落地方案示意图

2.价值和收益

该企业首轮试点后，半年多快速覆盖用户 7000+人，日活 1800+，AI 代码生成 220+万行，单测生成 300+万行，研发问答 46+万次。其中单测全量覆盖率达到 57%（提升 89%），增量覆盖率达到 78%（提升 1.3 倍），研发效率得到明显提升。

（二）软件服务业案例

某软件服务企业为国内众多行业提供软件研发服务，包括需求设计、开发、测试、运维等，但服务过程中面临的痛点问题长期存在。

1） 业务系统沟通效率低：在被服务的企业中，业务需求复杂且多变，但需求人员少而开发人员多，造成沟通瓶颈，使得项目的

整体进度迟缓;

2） 代码开发质量待改进：开发过程中缺乏高效的代码审查机制，导致潜在缺陷和技术债务未能及时被发现和解决，另外自动化测试实现不足，手动测试覆盖面有限，增加了系统上线风险;

3）信创体系带来的迁移困难：由于历史原因部分企业存在不少 Cobal 等语言的老业务系统，而现在熟悉 Cobal 的业务逻辑和技术人员极少，给系统迁移带来极大困难。

1. 解决方案

针对上述一系列问题，该企业自研了 AISE 一体化平台，助力软件研发过程提质提效。

1）通过将项目知识资产（需求文档、设计文档、业务需求等）进行处理，利用大模型智能分析问题和归纳总结能力，为每个项目组提供独立的智能问答助手，节省项目知识沟通成本;

2）提供代码自动生成、自动审查、自动修复、单元测试用例自动生成、代码翻译和转换等功能，并将上述功能完全集成在现有开发环境中，在不改变开发人员习惯的同时提升开发效率和质量;

3）引入多级外部知识库和大模型集成，支持 Github、Gitlab、网站、异构数据源等多种格式知识库的导入，同时无缝连接企业内部系统，提供效率查询、单点登录、业务系统数据联通等多种接口技术，给业务系统赋予更多智能化能力。

图 19 某软件服务企业案例落地方案示意图

2.价值和收益

该企业的 AISE 一体化平台落地后，Cobal 代码迁移效率提升 40%以上，项目沟通时间成本从以往以星期为周期降低为一天以内，开发效率提升 30%左右。

（三）电力行业案例

某国网数字化建设支撑单位，专注于电力行业多个专业数字化系统的研发与推广，致力于推动电力行业实现数字化转型，但在软件开发过程中，面临多重挑战。

1）外部资源难以利用：由于研发环境相对封闭，导致无法充分整合和合理利用外部优质资源；

2）开发周期长：项目规模庞大，需求繁杂，但开发人员有限，致使开发周期不断延长，影响项目交付的及时性；

3）代码规范性差、维护困难：缺乏严格的代码审查流程，导致

代码质量参差不齐，为后续的软件维护和升级工作带来了极大的困难，增加维护成本和风险。

1. 解决方案

针对上述问题，该单位基于人工智能大模型技术，研发了智能编码工具，实现了代码自动生成、优化、测试、规范性检查与修复等功能。

1）以代码大模型作为基座模型，运用 LoRa 进行私有代码的定制化微调，从而更好地服务于电力行业的特定需求，并适应不同部门和项目组的独特工作流程；

2）提供基于 AI 的查询和问答系统，针对公司公共组件库和统一数据模型进行检索，帮助开发者快速定位和理解组件功能，促进代码复用和标准化，降低维护成本；

3）开发自动化检测和修正工具，可根据公司的代码规范要求对代码进行实时扫描，自动识别不规范代码片段，并提供修正建议或直接进行自动修改，提升了整体代码质量。

图 20 某电力行业企业案例落地方案示意图

2. 价值和收益

通过智能编码工具的使用，代码生成和优化速度与准确率得到了显著提升，代码质量评分提高至 90%以上，开发效率提升超过 50%，项目交付时间平均缩短 40%。

（四）金融行业案例

某国有银行在企业数字化转型过程中，面临软件应用膨胀、代码量激增和架构复杂化的挑战。为抓住金融科技创新机遇，该银行决定引入大模型技术提升软件应用的开发效率，保障代码质量，并控制研发成本。该企业在落地前存在如下问题和需求。

1）老旧应用重构及迁移困难：该行有很多海外应用使用了 Flex 编程语言进行开发，代码量大且技术过时，而新人对 Flex 不熟悉，需投入大量精力维护，期望能将其重构成 React 代码;

2）自研框架开发门槛高：行内基于 SpringBoot 的自研框架规范性设计约束多，日常开发过度依赖开发文档，需降低开发门槛，提升效率;

3）代码质量隐患：由于线上问题时有发生，其影响大、定位难、耗时长，期望从代码层面保障开发质量，强化单元测试的执行力，原有单测覆盖率<10%。

1.解决方案

针对企业前期存在的问题，与其他公司联创实现了智能开发工具的落地。

1）代码翻译与转换：针对老旧应用从 Flex 转到 React 代码，使

用大模型的代码翻译基础能力，并将行内代码和转换规则等加入学习和训练调优；

2）代码辅助生成、框架知识问答：将行内自研开发框架、历史代码和开发文档等进行加入模型训练调优，优化对 Jump 等框架的场景处理能力；

3）单测用例生成：针对行内单测覆盖率低等问题，落地智能开发工具的单测生成能力，提升单侧覆盖率。

图 21 某银行案例落地方案示意图

2.价值和收益

该行通过落地智能开发工具，Flex 代码转换至 React 代码的准确率由 20%提升至 40%，代码生成采纳率超 30%，Jump 场景评分从 20 分跃升至 70 分。同时，核心业务场景单元测试覆盖率两周内提升 30%以上，部分业务系统覆盖率提升 50%以上。

（五）制造行业案例

某大型家电制造合资企业，作为领先的数字化转型解决方案提

供商，在企业数字化转型的过程中，面对日渐复杂的业务场景、逐渐庞大的代码规模和快速的交付节奏，期望通过大模型技术提升代码开发效率，该企业在落地前存在如下问题和需求。

1）代码阅读瓶颈：现有源代码库中注释稀缺或不准确，严重影响维护性的开发进度；开发人员面对陌生代码时，常陷入理解困境，尤其是涉及不熟悉的编程语言时，理解成本倍增；

2）开发效率亟待提升：企业面对庞大的研发团队、加速的交付节奏和严苛的质量标准，迫切需要提升研发效能、实现降本增效；

3）大模型创新探索：作为行业大模型应用的先行者，企业不仅密切关注前沿技术的创新与演进，也渴望深入探索代码大模型在智能家居行业实际应用场景中的表现，评估其成熟度，并以此推动整个智能制造业的进步。

1.解决方案

该企业秉持建设“行业第一，世界一流”的目标，整体制定代码大模型落地方案，在部分研发领域实现小步快跑。

1）通过落地代码大模型，实现代码生成、代码解释、代码注释等能力，解决阅读瓶颈等问题；

2）通过集成高可用模型、高性能服务器，并借助 RAG 检索增强和 SFT 模型调优等技术，持续定制化地训练和优化代码大模型在注释生成和补全、用例生成、安全合规等方面的表现；

3）部署智能化可视化看板，实时监测人员覆盖率、接口调用频次、问答接受度、单元测试覆盖率等各项指标，量化大模型对研发

效能的提升。

图 22 某家电制造企业案例落地方案示意图

2.价值和收益

该企业落地智能开发工具后，开发人员编码效率提升 20%~30%，企业内部 80%开发人员已使用，月活用户比例达到 70%，助力企业整体研发效能提升 20%以上。

六、总结与展望

智能化软件开发作为大模型落地应用最快的场景之一，其智能化水平得到持续提升，落地成效逐渐显现，落地方案逐渐明晰，为软件工程变革性发展带来巨大推动力。

软件工程智能化变革成为必然，智能开发助推价值提升。大模型推动软件工程 3.0 围绕着交互性、智能化、自适应和持续优化等特点持续发展，智能化能力为软件开发环节带来的价值显著，包括开发效率、代码质量和产品迭代创新能力的提升，推动软件工程高质量发展。

智能开发落地策略逐渐明确，三层落地框架成为主流。通过自我诊断、方案设计、部署实施、持续优化四个关键落地步骤，围绕模型层、服务层和应用层三层落地框架，企业落地智能开发能力的路径更加明晰。

核心能力和使能能力建设助力智能开发能力高效落地。一方面通过代码生成与补全、单测用例生成、代码转换与优化、代码解释与注释等核心能力建设，可实现代码大模型和智能开发工具的基础落地；另一方面通过数据、模型、评估和安全等多维度的使能能力建设，可实现代码大模型持续优化迭代，以及应用工具能力的提升。

未来，代码大模型和智能开发工具将从技术、应用和形态等方面持续发展，从而构建和落地更智能、全流程、可解决复杂研发问题的应用工具。

技术能力的持续发展将带来更高准确性和更优性能。一是大模

型自身能力的提升，基础大模型架构方面将探索更加高效、灵活、可扩展的底层模型架构（如 Mamba、MOE-Mamba 等），代码大模型方面将从高质量数据集、模型调优量化技术、上下文长度等角度生产更优性能的模型；二是工具工程化能力的提升，提示工程、RAG、AI Agent 等技术的发展将在工具层面为代码大模型进一步辅助增强，未来还将有更多新兴技术的发展和辅助，为智能开发能力落地提供更强劲动力。

落地应用的场景将更加丰富、流程更加全面。随着智能开发能力在前端页面、数据库、桌面应用、嵌入式等场景的逐步落地，未来将衍生更多场景，包括工业领域代码开发等，为赋能新型工业化提供更多路径。同时在软件工程全生命周期中的落地将更加全面，围绕智能开发向需求设计、软件测试、系统运维、项目管理等环节拓展，通过 AI Agent 等技术打通全流程智能化落地，从而实现“人人都是开发者”的低门槛生态。

应用形态将朝着更加智能化的方向持续推进。从当前的辅助开发人员完成任务，到帮助开发人员独立完成更复杂的研发任务，再到未来替代开发人员真正实现严格意义的智能开发，从而推动副驾驶（Copilot）到驾驶员（Pilot）的逐步演进。同时组织结构从团队作战将演变为单兵作战，开发人员将更聚焦于软件设计及更具有创新价值的工作，软件研发形态将得到重塑，软件业将迎来变革。

代码安全和安全代码相辅相成推动软件安全可信发展。代码安全是指通过智能化能力检查和修复代码中的安全漏洞，从而提升其

安全性，一是将拓展检查范围，不光检查传统代码，还将检查大模型等 AI 模型的安全漏洞；二是将降低代码检查工具误报率，提升漏洞检查质量。安全代码是指通过代码大模型生成的代码的安全性，未来将从数据、模型、工具等维度持续降低生成式代码的风险，使智能开发真正安全，使未来软件更加安全可信。

中国信息通信研究院 人工智能研究所

地址：北京市海淀区花园北路 52 号

邮编：100191

电话：010-62301618

传真：010-62301618

网址：www.caict.ac.cn