计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.241200033

大语言模型驱动的多智能体协同代码生成技术

夏 鹏 张燚钧 齐 骥
中移(苏州)软件技术有限公司 江苏 苏州 215000
(actpoar@hotmail.com)

摘 要 在代码生成任务中,预训练大语言模型和智能体已经成为提升代码生成质量和效率的关键技术。在面对复杂的编程问题时,基于大语言模型的智能体技术目前仍无法有效处理和解决。对此,提出一种多智能体协同代码生成框架,构建了包含问题分析、任务规划、代码生成和代码调试4个阶段的系统,通过智能体间的协作解决复杂的编程问题,并且基于开源大模型提出了不同的智能体基础模型使用策略,验证其对系统整体表现的影响。在此基础上,引入包含反思和调试循环的迭代式编程范式,以根据各阶段的结果反馈优化代码生成。实验结果表明,相比于传统直接代码生成方法,多智能体协同方案在多个数据集上取得了显著的性能提升,尤其在采用混合模型策略时,在所有测试数据集上均达到了最优表现。采用反思和调试循环时,在测试数据集上的表现有进一步提升。

关键词:多智能体系统;大语言模型;自然语言处理;代码生成;思维链
中图分类号 TP391

Multi-agent Collaborative Code Generation Technology Driven by Large Language Models

XIA Peng,ZHANG Yijun and QI Ji
China Mobile(Suzhou) Software Technology Co.,Ltd.,Suzhou,Jiangsu 215000,China

Abstract In code generation tasks,pretrained large language models and agents have become key technologies for improving the quality and efficiency of code generation. However,when facing complex programming problems,intelligent agents based on large language models still struggle to provide effective solutions. This paper proposes a framework of multi-agent collaborative code generation to solve complex programming problems through the collaboration among agents,which includes four stages:problem analysis,task planning,code generation,and code debugging. The different base model strategies for agents based on open-source LLMs are proposed and the impact on system performance is tested. Additionally,an iterative programming paradigm incorporating reflection and debugging loops is introduced to optimize code generation based on feedback from each stage. Experimental results demonstrate that the multi-agent collaborative approach achieves significant performance improvements compared to traditional direct code generation methods across multiple datasets. Particularly,the hybrid model strategy achieves optimal performance on all tested datasets. Performance on test datasets is further improved with the adoption of reflection and debugging loops.

Keywords Multi-agent system,Large language model,Natural language processing,Code generation,Chain of thought

1 引言

在人工智能领域,预训练大模型近年来已经成为推动技术进步的关键因素,尤其是以Transformer架构[1]为基础的语言模型,如BERT[2]和GPT[3]系列模型,已经证明了其在多种自然语言处理任务上的卓越性能。随着相关技术的发展,大模型开始被应用于代码生成领域,旨在通过自动化的方式提高软件开发的效率和质量。代码生成作为软件开发自动化的重要组成部分,其核心目标是减少手工编码的工作量,同时确保生成的代码符合预期的功能和性能标准。然而,现有的代码生成技术仍面临诸多挑战,如模型泛化能力有限,难以处理复杂的编程逻辑和算法需求等。为了克服这些问题,基于预训练大模型的智能体技术成为处理复杂代码编程任务的重要研究方向。

通过在大规模数据上的预训练,大模型具备强大的自然语言理解和生成能力,为构建具有高级认知能力的智能体提供了可能。在实际应用中,智能体在代码生成领域仍面临一系列技术挑战。首先,智能体需要能够准确理解复杂的自然语言问题描述,并将其转化为可执行的编程任务。其次,智能体必须具备生成高质量代码的能力,这不仅包括算法和数据结构的设计,还包括对代码逻辑和结构的精确把握。此外,智能体还需要能够执行综合的单元测试,确保生成的代码不仅能够编译通过,而且能够正确解决问题。这些挑战要求智能体不仅要有强大的语言处理能力,还要有深入的领域知识和灵活的问题解决策略。

在面对具有挑战性的代码生成任务时,研究者们开始探

基金项目:国家重点研发计划(2021YFB2801800)
This work was supported by the National Key Research and Development Program of China(2021YFB2801800).
通信作者:齐骥(qiji@cmss.chinamobile.com)

索多智能体协同的策略，以提高代码生成的质量和效率。多智能体系统通过模拟人类团队合作的方式，使得每个智能体可以承担不同的角色和任务，通过协作来解决单一智能体难以解决的问题。此外，多智能体系统可以通过分工合作，将复杂的编程任务分解为更小的、更易于管理的子任务，每个智能体可以专注于其擅长的领域，从而提高整体的编程效率。智能体之间的交流和协作也有助于减少错误和提高代码质量。

针对专用代码生成大模型在处理复杂软件开发任务时的局限性，如何在现有单智能体代码生成技术的基础上构建和使用多智能体代码生成方案，是亟待解决的问题。本研究旨在探索一种新的多智能体协同代码生成框架，模拟人类开发者在面对编程任务时的协作和问题解决过程，以及迭代式的反馈和改进机制。通过这种协作框架，突破单个智能体在处理复杂编程任务时的局限性，实现更高效和更可靠的代码生成。此外，现有的多智能体系统主要采用超大型的通用领域模型，而本文研究了通用领域模型和代码领域专用模型在多智能体系统中的不同配置方式，面向代码生成任务探索其最优配置策略。

2 相关工作

2.1 预训练大模型

预训练大模型已经成为自然语言处理领域的研究热点，通过在海量文本数据上进行预训练，模型学习到了丰富的语言知识，能够理解和生成自然语言文本。随着模型规模的不断扩大和训练技术的进步，预训练大模型在多种语言任务上都取得了突破性的性能[4]。2017 年，Google 研究团队提出 Transformer 模型，引入了自注意力机制，允许模型在处理序列数据时捕捉长距离依赖关系。它已成为自然语言处理和预训练大模型的核心技术[1]。在 Transformer 模型的基础上，2018 年 Google 团队发布了 BERT 模型，通过双向 Transformer 模型改善了自然语言理解和生成任务[2]。OpenAI 团队在 2018 年提出的一种预训练语言模型 GPT[3]，采用基于 Transformer 的解码器架构，利用大规模文本数据进行无监督预训练，以获得生成高质量文本的能力。通过在海量文本上进行预训练，GPT 能够捕捉到丰富的语言模式和世界知识。GPT 模型的规模随着版本迭代不断扩大，从最初的 GPT 到 GPT-2[5] 和 GPT-3[6]，直至最新的 GPT-4[7]，模型参数规模显著扩大，在多项自然语言任务上的表现也得到了显著提升。在人工智能领域，尤其是大语言模型的研究中，开源大模型具有至关重要的作用，推动了大模型在学术界和工业界的发展。近年来，众多开源预训练大模型不断涌现，例如 LLaMA[8]，Qwen[9]，GLM[10]，Deepseek[11] 等，在各项语言任务能力上媲美领先的闭源模型。

2.2 基于大语言模型的代码生成

随着大模型技术的快速发展，大语言模型在代码编程领域展现出了巨大的潜力。通过学习大量的代码数据，大语言模型能够捕捉到编程语言的复杂特性和编程逻辑，从而生成高质量的代码。例如，GPT-4 这样的大规模闭源模型，作为通用模型，在代码生成任务中已展现出卓越的性能[7]。开源大模型的相关工作，针对代码相关任务进行专项训练，并且发布了一系列代码领域的专用模型[12-14]。这些专用模型在代码领域的表现已经接近最领先的闭源模型水平，而在模型参数规模上显著小于通用模型。

面向代码生成任务，众多研究团队相继构建了一系列高质量的评测数据集。其中，OpenAI 团队构建的 HumanEval 数据集[15] 和 Google 团队构建的 MBPP 数据集[16] 是代码生成领域最常用的权威评测数据集。在此基础上进一步扩展优化的评测数据集[17]，可以更全面、准确地评测代码生成效果。在 HumanEval 数据集上，常见的闭源通用大模型和代码领域专用大模型的评测指标，即代码生成的一次通过率（pass@1）如下：GPT-3.5-Turbo 为 76.2%，GPT-4 为 84.1%[7]，Code-Llama-Instruct-70B 为 67.8%[12]，StarCoder2-15B 为 46.3%[13]，DeepSeek-Coder-Instruct-6.7B 为 78.6%[14]。在 MBPP 数据集上，以上模型的指标如下：GPT-3.5-Turbo 为 70.8%，GPT-4 为 80.0%[7]，Code-Llama-Instruct-70B 为 62.2%[12]，StarCoder2-15B 为 66.2%[13]，DeepSeek-Coder-Instruct-6.7B 为 65.4%[14]。

2.3 大语言模型驱动的智能体技术

智能体技术作为人工智能领域的关键分支，近年来取得了显著的进展。智能体被定义为能够感知环境、做出决策并采取行动的人工实体[18]。随着人工智能技术的发展，智能体的研究已经从单一的算法优化转向了构建更为通用和强大的模型，以适应多样化的应用场景[19]。预训练大模型的兴起，为智能体技术的发展提供了新的动力。大模型在知识获取、指令理解、泛化能力、规划和推理方面展现出强大的能力，因此研究人员以大模型作为基础构建智能体，尤其是把大模型作为智能体的思维和控制核心。

基于预训练大语言模型的智能体不仅能够通过思维链和问题分解等[20] 技术展现出强大的推理[21] 和规划能力[22]，而且能从反馈中学习并对环境做出响应[23]，为构建具有高级认知能力的智能体提供了可能。基于智能体的自我反思和长期记忆能力[24]，智能体能够回顾历史决策并优化策略，并从经验中学习，从而在复杂环境中进行有效的任务规划和策略选择。大模型已经在大规模数据上进行预训练，展示出 few-shot[25] 和 zero-shot[26] 的泛化能力，在不需要重新训练和更新参数的情况下，可以适应智能体涉及的多种不同任务。基于预训练大模型的智能体技术已经被应用于实际场景，如软件开发[27] 和科学研究[28] 等，利用智能体的分析和规划能力[29]，有效提升应用场景中的表现。

基于大模型在自然语言理解与生成方面的能力，智能体之间可以有效地交互，从而实现多个智能体之间的协同工作[30]。微软研究团队提出的 AutoGen 多智能体框架[31]，通过多个能够相互交流以完成任务的智能体来构建应用，可以使用自然语言来灵活定义不同应用的智能体和交互行为，支持构建各种复杂和多样的应用，包括数学问题求解、检索增强的问答、文本环境中的决策制定和多智能体编程等。Hong 等提出的 MetaGPT 框架[32]，将人类软件开发流程与基于大模型的多智能体系统相结合，基于开发流程的标准化操作程序，设计智能体提示序列，采用流水线的工作范式，为不同智能体分配多样化角色，将复杂任务分解为涉及多个智能体合作的子任务。通过结构化的智能体协作和强化领域特定专业知识来处理复杂任务，生成的解决方案比基于聊天的多智能体系统具有更好的正确性和连贯性。

以上智能体代码生成方案均是以 GPT-4 为基础模型，这

是 OpenAI 公司发布的闭源大模型，一方面使用成本较高，另一方面必须联网使用，在信息安全方面存在较大的风险。为了解决以上问题，本文将完全采用开源大模型进行多智能体代码生成技术的研究，并且提出不同模型使用策略，对比不同策略下的计算开销。

3　多智能体协同代码生成

本研究提出的多智能体协同代码生成框架，旨在构建一个能够独立协作、解决复杂编程问题的系统。

3.1　多智能体协同系统框架

系统包含多个基于预训练语言大模型的智能体，通过明确的智能体任务定义、标准化的工作流程、结构化的交互机制以及迭代式的编程和调试策略，实现高质量的代码生成。系统框架如图 1 所示。预训练语言大模型是智能体能力的核心，应具备通用的自然语言理解和生成能力，以及代码理解和生成能力。此外，智能体间的协同工作机制是实现复杂任务分解和执行的关键，在代码生成和调试过程中，协同机制根据各阶段的结果反馈，进行智能体的任务调度。系统还引入了包含反思和调试循环的迭代式编程范式，以根据各阶段的结果反馈优化代码生成。

本文的多智能体协同框架包含 4 个智能体：1）问题分析智能体；2）任务规划智能体；3）代码生成智能体；4）代码调试智能体。这 4 个智能体构成的工作流程如图 1 所示，4 个智能体也分别负责工作流程中的个阶段：1）问题分析智能体是在接收代码生成需求时，对问题和潜在解决方案进行分析；2）任务规划智能体根据问题分析的结果，把需求进一步分解为代码生成任务；3）代码生成智能体根据任务规划生成具体的程序代码；4）代码调试智能体可以根据代码编译执行和测试的结果反馈，选择直接完成任务，或者进一步优化代码以通过所有的代码测试。通过上述 4 个智能体的协作，系统能够将复杂的编程任务分解为更小的、更易于管理的子任务，每个智能体专注于其擅长的领域，从而提高整体编码能力。

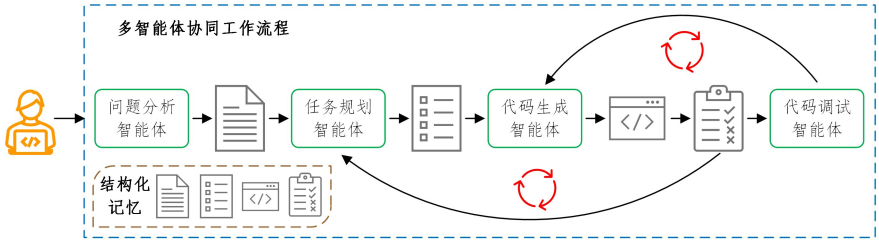

图 1　多智能体协同工作流程示意图

Fig. 1　Diagram of multi-agent collaborative workflow

3.2　问题分析智能体

在多智能体协同框架中，问题分析智能体负责工作流程的第一个阶段，其主要任务是对输入的编程问题进行深入分析，理解问题需求，识别问题的关键点和难点，提出潜在的解决方案，并为后续的任务规划和代码生成提供准确的信息和指导。该智能体需要具备对自然语言描述的准确理解能力，并能够将这些描述转化为代码编程领域的具体问题表述。

该阶段输入的问题可以是自然语言描述、相关的代码片段或示例、特定的约束条件等。问题分析智能体的任务定义和输出要求如图 2 所示。智能体解析问题的自然语言描述，提取关键需求信息和约束条件；预测需要使用的算法或数据结构；给出潜在的解决方案，包括任务规划和代码。输出结果必须按照规定格式返回，不得输出额外内容或者不符合格式要求的内容。输出的分析结果按照规定格式解析后，作为后续工作流的输入内容和其他智能体的参考知识。

问题分析智能体

```
Given a coding problem. Please analyze the problem, extract the requirement and
constraint, and identify the difficulties. Then identify the algorithm needs to be
used for the problem. Finally, propose a group of plans and codes as candidate
solutions to solve this problem.
# Problem: {输入问题}
---------------------------------------------------------------
Your response must follow the following format:
## Problem Analyzing:
## Algorithm:
## Candidate Solutions:
```

图 2　问题分析智能体的提示词

Fig. 2　Prompts for problem analysis agent

问题分析智能体在多智能体协同框架中是整个代码生成过程的起点，其分析结果直接影响后续任务规划、代码生成质量和代码调试效率。通过对输入问题的精准理解，可以为任务规划智能体和代码生成智能体提供有效的指导，并为代码调试智能体定位和解决问题提供必要的参考信息。

3.3　任务规划智能体

任务规划智能体负责多智能体协同框架工作流程中的第二个阶段，可以根据问题分析智能体提供的分析结果，将问题解决方案细化为可执行的代码生成任务，其中包括一系列可操作的子任务，任务规划智能体的输出结果包含执行这些子任务的顺序和方法。该智能体还引入了反思机制[24]，可以结合生成结果和后续流程的反馈，对已生成的任务规划进行优化修正。

该阶段的初始输入是代码编程问题和前一阶段的问题分析结果，以结构化的自然语言形式作为该阶段的输入文本。如图 3 所示，任务规划智能体负责根据问题分析结果，将代码编程问题分解为多个子任务，选择最适合该问题的算法或数据结构，设计代码的模块化结构，明确各个模块的功能和接口，制定详细的代码生成和执行步骤。最终生成详细的任务规划文档，其中不包含具体代码。该阶段的输出结果同样必须按照规定格式返回，以保证结果可以被系统解析为后续流程的输入内容。

任务规划智能体被设计为具备反思能力，不仅可以处理前一阶段的输入，还可以根据后续阶段的反馈，对任务规划进行优化修正。如图 3 所示，后续阶段的反馈可以是来自于代码生成和执行后的反馈，也可以是来自于代码调试后的反馈。如果后续阶段执行结果不符合预期，例如代码执行报错或者

不能通过测试用例，则由系统的协同调度模块将后续阶段的输出结果组织为结构化的反馈内容，作为任务规划智能体的输入，执行反思循环。反思循环通常设置为有限轮次，以防止在遇到困难问题时陷入无限循环。

任务规划智能体

任务规划提示词：
Given a coding problem. Please generate a concrete planning to solve the problem.
Problem: {输入问题}
Problem analyzing: {问题提分析阶段输出}
Algorithm: {问题分析阶段输出}
Candidate Solutions: {问题分析阶段输出}

Your response should give only the planning to solve the problem.

反思循环提示词：
Given a coding problem.
Problem: {输入问题}
Given your previous trial to solve the problem, including Planning, Codes, Test Report and Debugging Record. Please explain why the previous trial is wrong as indicated by the tests. Then generate the modified planning to solve the problem.
Planning: {前一次的任务规划阶段输出结果}
Codes: {代码生成阶段输出结果}
Test Report: {代码测试结果}
Debugging Record: {代码调试阶段记录（可选）}

Your response must follow the following format:
Explanation:
Modified Planning:

图 3 任务规划智能体的提示词

Fig. 3 Prompts for task planning agent

3.4 代码生成智能体

代码生成智能体负责多智能体协同框架工作流程中的第三个阶段，可以根据任务规划智能体提供的详细任务规划，生成具体的程序代码。智能体需具备将任务描述转化为实际的代码实现的能力，同时确保生成代码的准确性和执行效率。在该智能体的设计中，采用思维链技术[20]，利用代码生成任务逻辑的中间概念和推理步骤，进一步提高代码生成的质量。

该阶段的输入内容包含代码编程问题和前两个阶段的输出结果，被组织为代码生成智能体的结构化输入。代码生成智能体根据问题分析的结果，按照任务规划步骤生成代码，并按照规定格式返回。生成的程序代码将用于执行相应的测试用例，由系统的代码执行模块进行。在本文的实验中，为保证测试用例的有效性，在代码执行中使用的测试用例全部来自于测试集中人工标注的测试用例。

3.5 代码调试智能体

代码调试智能体负责多智能体协同框架工作流程中的第四个阶段，采用基于反馈的调试框架，可以根据代码执行情况的反馈进行后续调试，在发现错误时提供修正建议，生成修正后的可执行代码，然后继续执行测试用例。该智能体可以设置为多轮次的调试循环，直至测试用例全部通过。在该智能体的设计中，同样采用思维链技术[20]，在生成修正建议和代码时，可以进一步提高代码修正的准确性和有效性。

如图 4 所示，该阶段的输入内容主要包含前一阶段生成的可执行代码，以及代码执行测试用例的反馈信息，还包含第一和第二阶段的问题分析结果和任务规划信息。代码调试智能体根据输入内容生成结构化的输出，输出内容包括代码缺陷和修正建议，并按照规定格式生成修正后的代码。系统从输出结果中解析出程序代码，继续执行代码测试。

与代码生成阶段不同，在代码调试阶段，智能体的输入内容中包含了测试用例的具体信息和执行结果。所以，代码调试阶段的代码修正对于提升最终代码质量具有重要作用，是多智能体协同框架中不可或缺的环节。如果在预先设定的有限次调试循环后，生成代码仍不能通过全部测试用例，那么系统可以执行反思修正循环。代码调试阶段的记录内容将与其他阶段的输出内容组合，作为反思循环中任务规划智能体的输入内容。

代码调试智能体

Given a coding problem. The generated planning and {language} code to solve the problem can not pass the test cases. Please improve the planning and codes to solve the problem correctly.
Problem: {输入问题}
Planning: {任务规划阶段输出结果}
Code needs to be modified: {代码生成阶段输出结果}
Test Report: {测试用例执行结果}
Let's think step by step to modify {language} Code for solving this problem.

Your response must contain the modified planning and the {language} code inside ``` block to solve this problem.

图 4 代码调试智能体的提示词

Fig. 4 Prompts for code debugging agent

3.6 智能体协同调度

智能体协同调度模块在工作流中负责协调和管理整个多智能体系统的工作，确保代码生成流程的各个阶段能以正确的方式运行。该模块把任务分配给对应的智能体执行，对各智能体的输入和输出内容进行组织和解析，处理和保存工作流的中间结果，并将其作为各智能体共享的结构化记忆，在工作流程的各阶段使用；监控任务执行状态，及时响应异常情况，并根据任务执行反馈来调整生成策略。图 1 中的多智能体工作流程的调度算法如算法 1 所示，当代码生成结果不符合预期时，系统可以根据编译执行结果，采用迭代式的方法进行代码修改和调试，以逐步修正代码。系统还可以采用反思机制，把系统记忆中保存的代码生成和调试失败记录组织为结构化的反馈信息，调用任务规划智能体修正任务规划，然后再次进行代码生成和调试。

算法 1 智能体协同调度算法

输入：问题 Q，调试和反思循环次数上限 k_d 和 k_r

输出：生成代码 C_{output}

算法步骤：

1. 初始化问题分析、任务规划、代码生成、代码调式智能体：$A_{analysis}$，A_{plan}，A_{code}，A_{debug}
2. 初始化循环执行次数 $t_d=0$，$t_r=0$
3. 处理 Q 为问题分析提示词，输入智能体 $A_{analysis}$，获得生成结果 $R_{analysis}$
4. 解析 $R_{analysis}$ 并处理为任务规划提示词（如果解析失败，则提示词对应内容为空），输入智能体 A_{plan}，获得生成结果 R_{plan}
5. 解析 R_{plan} 并处理为代码生成提示词（如果解析失败，则提示词对应内容为空），输入智能体 A_{code}，获得生成结果 R_{code}
6. 解析 R_{code} 并处理为候选代码 $C_{candidate}$（如果解析失败，则候选代码为空）
7. 编译执行候选代码 $C_{candidate}$ 和测试用例：
   7.1. 如果通过全部测试用例，则 $C_{output}=C_{candidate}$，返回生成代码
   7.2. 如果代码执行失败或无法通过测试用例，获得执行反馈 R_{run}
   7.3. 如果 $t_d < k_d$，则处理 R_{run} 为代码调试提示词，输入智能体 A_{debug}，获得生成结果 R_{debug}；如果 $t_d \geq k_d$，则进入步骤 8
   7.4. 解析 R_{debug} 并处理为候选代码 $C_{candidate}$（如果解析失败，则候选代码为空）
   7.5. $t_d += 1$，进入步骤 7

8. 如果 $t_r < k_r$,则解析 Q,R_{code},R_{plan},R_{run},R_{debug}处理为反思循环提示词,输入 A_{plan}获得生成结果 R_{plan};如果 $t_r \geqslant k_r$,则 $C_{output} = C_{candidate}$,返回生成代码

9. t_r += 1,进入步骤 5

智能体的输入被组织为结构化的内容,以确保智能体输入信息的完整性和准确性。在输入内容中也规定了智能体的输出格式,才能确保智能体的输出内容被系统解析。但是,受限于大模型的生成能力和指令遵从效果,智能体有可能不按照规定格式输出,或者输出内容有异常,导致系统无法解析智能体的输出结果。

为此,在智能体协同调度中设计了异常处理和容错机制,以增强系统的鲁棒性和自适应性。在这类情况下,系统会捕捉到异常,然后根据系统预先设置的内容重新组织下一个阶段需要的结构化内容,以确保工作流成顺利进行。此时的输入内容中前一阶段未解析成功的内容以空字符代替,因此缺少前一阶段智能体生成的有效信息输入,会影响到整体生成效果。这类由于大模型生成能力和指令遵从性导致的整体代码生成能力下降的情况,在规模较小的预训练模型作为智能体的基础模型时更为常见。

3.7　基础模型选型和使用策略

预训练大模型作为多智能体协同代码生成系统的核心,必须具备强大的通用自然语言理解与生成能力,能够按照指令生成符合格式要求的内容,并且具备思维链能力。此外,模型还需要能准确理解和解析复杂的编程问题,并能够生成结构良好且语法正确的代码。模型还应具备处理长文本的能力,以应对在系统工作流程中可能出现的长上下文内容。

经过指令微调的通用预训练大模型,通常具有良好的自然语言理解和生成能力,并且能够利用思维链技术来提高推理能力。通用预训练大模型也具备一定的编程问题理解和代码生成能力,可以作为智能体的基础模型。此外,专用于代码编程领域的预训练大模型在编程问题理解和代码生成方面具有显著优势,而且具备一定的通用自然语言对话能力,也可以作为智能体的基础模型。

本文提出的多智能体协同系统由 4 个不同功能角色的智能体构成,其中问题分析智能体和任务规划智能体侧重于自然语言理解和复杂问题分析能力,代码生成智能体和代码调试智能体侧重于代码理解和生成能力。根据不同智能体的能力侧重点,可以采用不同的预训练大模型作为智能体的基础模型。

现有的研究主要是以 ChatGPT-3.5 或 GPT-4 等超大规模的通用预训练模型作为智能体的基础模型。在代码编程领域,已经公开的专用于垂直领域的预训练模型,尽管在模型规模上远小于以上通用模型,但是在代码类任务中的表现达到甚至超过通用模型。在多智能体协同代码生成的框架下,代码编程领域的专用模型是否能有效提升整体的代码生成能力和效率,是亟待研究的问题。在此框架下,按照智能体的能力侧重点来选择合适的基础模型,合理配置通用模型与专用模型在系统中的使用方式,也是进一步提升系统效率和代码生成能力的重要路径。在本研究中采用的基础模型分为两类:通用模型和代码领域专用模型。本研究将对比 3 种不同的基础模型使用策略:1)通用模型策略,即所有智能体均使用通用模型;2)专用模型策略,即所有智能体均使用代码领域专用模型;3)混合模型策略,即问题分析智能体和任务规划智能体使用通用模型,代码生成智能体和代码调试智能体使用代码领域专用模型。

4　实验设计与结果分析

本研究采用广泛认可的公开数据集和评价指标作为实验方案,以开源大模型作为智能体的基础模型,对大模型直接代码生成和多智能体代码生成方法进行对比,并对不同方法的计算开销进行评估。

4.1　实验数据

本研究提出的多智能体协同方案可以适用于各种不同的编程语言,只要基础模型具备相应的语言理解和生成能力,并且在提示词中表明采用的编程语言,此外还需配置各语言的编译执行环境。在具体实验中,为了简化实验过程中的代码编译执行,本研究采用最常用的解释型编程语言 Python 作为代码生成任务的语言。采用 Python 代码生成任务中常用的权威评测数据集 HumanEval 和 MBPP,以及在其基础上扩展的数据集 HumanEval-E 和 MBPP-E,作为实验数据。

HumanEval 数据集[15]是一项专为衡量大型语言模型在 Python 编程任务中的功能正确性而设计的评估工具。该数据集由 OpenAI 团队构建,包含 164 个手工编写的编程问题,每个问题均配备详尽的测试用例。这些问题覆盖了语言理解、算法和基础数学等领域,难度与初级软件面试题目相当。MBPP 数据集[16]是一个专为评估大型语言模型在代码生成领域性能而设计的基准测试集,用于评估模型根据自然语言描述生成 Python 函数的能力。该数据集的编程问题均由人工构建,并由人工二次验证筛选出 397 个测试样本。每个样本都配备了明确的问题描述和 3 个用于验证函数语义正确性的测试用例。MBPP 数据集中的问题覆盖了从基础数值操作到列表和字符串处理,再到需要特定知识的问题。HumanEval-E 和 MBPP-E 数据集[17]是在 HumanEval 和 MBPP 数据集的基础上,每个样本扩增了 100 个以上的测试用例,并且手动过滤了可能导致代码执行错误或失败的用例。以上数据集覆盖了初级程序设计任务中的 Python 语言理解、推理、算法和数学等问题,以编写函数代码的形式解决编程任务。

4.2　对比实验设计

为了验证多智能体协同代码生成方案的有效性,将对比基于预训练大模型的直接代码生成与智能体协同代码生成方案,以及具有代表性的大模型代码生成方法,包括思维链技术[20]和采用自我规划[29]的代码生成方法。直接代码生成是采用自然语言对话方式,把测试样本的编程问题作为对话输入,由模型生成回答,提取回答结果中的生成代码,然后执行样本中的测试用例。多智能体协同代码生成方案是采用本文提出的代码生成方案,把测试样本的编程问题以自然语言的方式直接输入系统,由系统执行代码生成的工作流程,最终生成的代码将执行测试用例。智能体的基础模型使用策略分别采用本文提出的 3 种策略:通用模型策略、专用模型策略和混合模型策略。

在多智能体协同代码生成方案中,工作流程可以包含反思循环和调式循环。这两种循环的输入中均包含测试用例的执行结果,导致在智能体的输入内容中包含了最终执行的测试用例,这与常用评测基准的测试方式不同。在常用评测基

准的测试中，编程问题可以包含测试用例的接口示例，但是不包含实际执行的测试用例。但是，在编程任务中，根据实际的测试用例进行调试和修正是常见的开发过程。在本研究的实验中，包含反思循环和调式循环的代码生成方式作为单独的测试对比项。以迭代循环方式的代码生成方式需要控制执行过程的资源消耗，所以在实验中系统设置为每个问题最多执行 1 次反思循环，每次代码生成阶段后最多执行 2 次调试循环。

多智能体协同需要多次调用大语言模型，相比于直接与大模型进行单轮问答，会产生额外的计算开销。对其量化分析需要考虑影响大模型计算开销的诸多因素，包括模型参数规模、模型结构、推理框架、硬件特性、上下文长度等。为了简化对比计算方式，并且能将不同模型的计算开销在同一个标准下对比，本文把代码生成过程中输入和输出大模型的 Token 数量总和作为衡量计算开销的基础。此外，通用模型和专用模型的规模相差较大，单位 Token 对应的计算开销有显著差异。为了使两类模型的计算开销可以在同一个基准下对比，本文根据基础模型规模，把专用模型输入输出的 Token 数量乘 0.25，而通用模型的 Token 数量不变，计算出代码生成过程中的等效 Token 数量，以每个问题样本所需的平均等效 Token 数量来衡量计算开销。

实验硬件为一台高性能服务器，内存 976GB，包含 8 块 Nvidia-A100-40GB 显卡，安装操作系统为 Ubuntu20.04。使用 vLLM 推理框架部署基础模型，模拟 OpenAI API 服务协议，提供可供系统调用的 API 服务。

4.3 基础模型

本文实验中采用的两类基础模型均采用开源模型，为了避免同源模型可能带来的测试结果相似性，这两类模型采用来自不同组织提供的开源模型，并根据模型部署的资源开销选择合适的模型规模。

通用模型采用 Qwen 大模型系列[9]中的 Qwen1.5-32B-Chat 模型。Qwen 大模型是由 Qwen 团队研发的一系列大型语言模型，预训练阶段通过对高达 3 万亿个 Token 的文本和代码数据进行学习，不仅掌握了基础语言技能，还在算术、编程和逻辑推理等领域表现出色。通过监督式微调和基于人类反馈的强化学习，Qwen 模型进一步与人类偏好对齐，在复杂任务上展现出与更大规模模型相近的能力。本文采用的 Qwen1.5-32B-Chat 模型即为已经进行人类偏好对齐后的通用对话模型，具有 320 亿的参数规模，可在两块 Nvidia-A100-40GB 显卡上实现部署。

代码领域的专用模型采用 DeepSeek-Coder 系列[11]中的 DeepSeek-Coder-Instruct-6.7B 模型。DeepSeek-Coder 是由 DeepSeek-AI 团队研发的一系列开源代码领域模型，使用高达 2 万亿个 Token 的数据量，覆盖 87 种编程语言，包含高质量的项目级代码语料库，确保了模型对编程语言和语法的全面理解。DeepSeek-Coder 在多个代码领域基准测试中表现出色，不仅在开源代码模型中实现了最先进的性能，而且在多数评估基准上超越了现有的闭源模型。本文采用的 DeepSeek-Coder-Instruct-6.7B 模型，在多语言的评测基准上已经达到闭源模型 ChatGPT-3.5 的水平，具有 67 亿的参数规模，可在单块 Nvidia-A100-40GB 显卡上实现部署。

4.4 评价指标

HumanEval 和 MBPP 数据集的代码生成任务主要采用 Pass@k 作为评价指标。Pass@k 指标表示在 k 个代码生成结果的情况下，至少有一个代码结果通过所有单元测试的概率。这个指标考虑了生成多个样本并从中选择最佳结果的情况，模拟了现实世界编程任务中迭代和错误修复的过程。本文旨在研究有限资源下系统自动生成的代码效果，并不包含人工介入的多次尝试，而且系统工作流中已经采用迭代循环的代码生成方式。所以本研究采用 Pass@1 作为评价指标，即只生成 1 个代码结果通过所有单元测试的概率。此外，为了保证生成结果的可重复性，大模型在推理过程中采用贪心搜索策略，即在生成的每一步直接选择预测概率最大的词作为解码结果。

4.5 实验结果分析

4.5.1 多智能体协同的模型策略和循环策略分析

在实验中，以问答形式直接生成代码的方案为基线方案，对比方案是本文提出的多智能体协同代码生成方案。在多智能体协同方案中，进一步对比了基础模型的选型策略，以及反思和调试循环的作用效果。如表 1 所列，“Direct”表示问答形式的直接代码生成，“Ag”表示采用多智能体协同方案；“-G”表示采用通用模型策略，“-S”表示采用专用模型策略，“-M”表示采用混合模型策略。为了对比反思和调试循环中不同阶段的效果，把代码生成过程进一步细分为 4 个阶段。第一阶段是首次代码生成后未进入调试循环，无额外后缀；第二阶段的后缀是“-C”，表示代码生成后首次进入调试循环；第三阶段的后缀是“-CR”，表示调试循环后仍不能解决问题，进入反思循环生成代码，但没有进入调试循环；第四阶段的后缀是“-CRC”，表示反思循环生成代码后，再次进入调试循环。

表 1 多智能体系统采用不同基础模型和工作流程的影响

Table 1 Impact of different base models and workflows on multi-agent system

评价指标	Pass@1/%			
对比方案	Human-Eval	Human-Eval-E	MBPP	MBPP-E
Direct-G	62.80	51.83	51.89	44.84
Direct-S	73.78	68.90	63.48	53.40
Ag-G	64.02	56.10	64.23	50.88
Ag-G-C	67.07	58.54	68.26	52.39
Ag-G-CR	67.68	60.37	69.27	52.39
Ag-G-CRC	70.12	63.41	71.79	53.15
Ag-S	71.34	65.24	68.77	56.42
Ag-S-C	78.05	68.29	73.80	57.43
Ag-S-CR	78.05	68.90	74.81	57.93
Ag-S-CRC	81.71	70.12	76.57	59.19
Ag-M	77.43	71.95	73.05	57.43
Ag-M-C	79.88	73.17	74.56	59.95
Ag-M-CR	80.49	73.17	74.81	59.95
Ag-M-CRC	82.32	73.17	78.59	60.45

相比于直接代码生成方案，多智能体协同方案未采用反思和调试循环时，在不同测试数据上的评估指标已经有显著提升。当基础模型采用通用模型策略时，在 4 个数据集上，多智能体协同方案相比直接代码生成方案的指标有显著提升，其中在 MBPP 数据集上的提升幅度最大，提升了 12.34%。当基础模型采用专用模型策略时，在 MBPP 和 MBPP-E 数据集上，多智能体协同方案相比直接代码生成方案的指标分别上升 5.29%和 3.02%。但是，在 HumanEval 和 HumanEval-E 数据集上，多智能体协同方案指标分别下降 2.44%和

3.66%。专用模型在问题分析阶段和任务规划阶段的通用能力和指令遵从较差,导致输出结果不符合格式要求或内容混乱,是这组指标下降的主要原因。基础模型采用混合模型策略时,在4个数据集上均达到最优表现。结果表明,根据智能体的能力侧重点优化基础模型配置,一方面可以借助于通用模型的能力,克服专用模型在通用能力和指令遵从方面的局限性;另一方面可以借助于专用模型在代码领域的能力,进一步提升整体的代码生成效果。

采用反思和调试循环的多智能体协同方案相比于未采用的方案,指标有显著提升。采用通用模型策略时,在MBPP数据集上的提升幅度最大,提升了7.56%。采用专用模型策略时,在HumanEval数据集上的提升幅度最大,提升了10.37%。采用混合模型策略时,在MBPP数据集上的提升幅度最大,提升5.54%。当基础模型采用混合模型策略时,采用反思和调试循环的多智能体协同方案在4个数据集上均达到最优表现,表明反思和调试循环在进一步提高系统的代码生成能力方面具有显著作用。

对比反思和调试循环的3个阶段指标,可以分析引入反思和调试循环对系统性能的影响。在首次进入调试循环时,提升效果最为显著,其中提升幅度最大的是采用专用模型策略,在HumanEval数据集上提升了6.71%。进入反思循环后但没有进入调试循环时,提升效果最小,平均提升幅度小于1%。在反思循环中再次进入调试循环后的提升效果也较显著,其中提升幅度最大的是采用混合模型策略,在MBPP数据集上提升了3.78%。以上结果表明,基于执行结果反馈的调试循环对于提升代码生成质量有显著作用,在3种模型策略下均有效。尽管反思循环中首次生成代码的指标提升幅度较小,但是反思循环中的调试循环对代码质量提升依然显著,也表明反思循环中重新进行的任务规划对于后续代码生成和调试有较为显著的作用。

4.5.2　计算开销评估

以每个问题样本所需的平均等效Token数量表示对应方案的计算开销,统计实验过程的计算开销,如表2所列。

表2　多智能体系统的计算开销对比

Table 2　Comparison of computational overhead in multi-agent systems

计算开销	平均等效Token数量			
对比方案	Human-Eval	Human-Eval-E	MBPP	MBPP-E
Direct-G	372	372	306	306
Direct-S	110	110	93	93
Ag-G	1661	1661	1375	1375
Ag-S	403	403	376	376
Ag-M	1393	1393	1179	1179
Ag-G-CRC	3199	4072	2978	3783
Ag-S-CRC	842	1107	775	1096
Ag-M-CRC	1779	2375	1382	1573

直接代码生成方案的计算开销显著小于采用智能体的方案。其中,采用专用模型策略时计算开销最小,在Human-Eval和MBPP数据集上的平均等效Token数量分别为110和93。多智能体协同方案计算开销与调用大模型的次数有关,采用反思和调试循环的计算开销显著高于未采用的方案。在多智能体协同方案中,专用模型策略的计算开销最小,通用模型策略开销最大,混合模型策略介于两者之间。结合各方案的在各评测数据集上的表现,混合模型策略是追求最优代码生成效果时的可选方案,专用模型策略是追求低计算开销和较高代码生成质量时的可选方案。

4.5.3　代码生成方法对比分析

多智能体协同代码生成与具有代表性的大模型代码生成方法的对比如表3所列。“CoT”表示采用思维链技术[20]的代码生成方法,“Self-Planning”表示采用自我规划[29]的代码生成方法,“Ag”表示采用多智能体协同代码生成方法。采用思维链与自我规划的方法并没有多个智能体,所以不会采用多个不同的预训练模型,只有多智能体协同方法中会采用不同的预训练模型。在表2的对比方法名称中,“-G”表示采用通用模型,“-S”表示采用专用模型,“-M”表示采用与表2中相同混合模型策略。

表3　不同代码生成方法的对比实验结果

Table 3　Comparative experimental results of different code generation methods

评价指标	Pass@1/%			
对比方案	Human-Eval	Human-Eval-E	MBPP	MBPP-E
CoT-G	62.80	54.27	54.91	50.63
CoT-S	78.05	66.46	70.28	55.67
Self-Planning-G	65.24	55.48	62.72	51.13
Self-Planning-S	73.78	64.63	68.51	54.66
Ag-G	64.02	56.10	64.23	50.88
Ag-S	71.34	65.24	68.77	56.42
Ag-M	77.33	71.95	73.05	57.43

在MBPP,MBPP-E和HumanEval-E数据集上,采用混合模型策略的多智能体协同代码生成方法“Agents-Mix”均取得最优结果。在HumanEval数据集上,采用代码领域专用模型和思维链技术的方法“CoT-S”取得最优结果,比“Agents-Mix”高出0.72%。这表明,采用思维链技术即可显著增强专用模型在代码生成方面的能力,多智能体协同和自我规划等技术对专用模型的提升作用不如思维链技术。此外,在仅使用专用模型时,采用思维链技术的代码生成方法在4个数据集上平均表现最优。这在一定程度上也表明专用模型在代码领域和特定任务上进行专项训练优化后,复杂的分析和规划流程并不能有效提升其在特定任务中的表现。在仅使用通用模型时,自我规划方法“Self-Planning-G”在HumanEval和MBPP-E数据集上取得最优结果,多智能体协同方法“Agents-G”在HumanEval-E和MBPP数据集上取得最优结果。这表明对于通用模型而言,分析和规划流程能更有效地提升其代码生成的表现。

结束语　为提升预训练大语言模型在代码生成任务中的表现,本研究提出了一种创新的多智能体协同代码生成框架,旨在通过多智能体之间的协作来解决复杂的编程问题。本研究通过构建一个包含问题分析、任务规划、代码生成和代码调试4个阶段的系统,利用预训练大模型的协同工作来提高代码生成的质量和效率。通过实验设计与结果分析,验证了多智能体协同方案相比传统直接代码生成方法的有效性。以开源大模型作为基础模型来源,提出和对比了通用模型、专用模型和混合模型等不同的智能体基础模型使用策略,然后进一步探讨了反思调试循环在多智能方案中对代码生成的提升效

果，并且评估了在不同策略下系统的计算开销。

实验结果表明，相比于直接代码生成，多智能体协同在多个数据集上均取得了显著的性能提升，特别是当采用混合模型策略，即在问题分析和任务规划阶段使用通用模型，在代码生成和调试阶段使用专用模型时，多智能体协同方案在所有数据集上均达到了最优表现。此外，反思和调试循环的引入进一步增强了系统的代码生成能力，验证了迭代方法在实际编程任务中的重要性。研究还发现，尽管专用模型在代码生成方面具有优势，但在问题分析和任务规划阶段，通用模型在自然语言理解和复杂问题分析方面的能力依然不可或缺。在构建多智能体协同系统时，合理配置不同能力侧重点的模型是提升整体性能和降低计算开销的关键。

在未来的工作中，面向更具挑战性的编程问题和更大规模的软件开发任务，将进一步探索预训练大模型和智能体技术，支持多种编程语言和编程范式，提高生成代码的质量和安全性，并且引入人机交互机制，根据特定的需求和偏好定制代码生成过程。

参 考 文 献

[1] VASWANI A, SHAZEER N, PARMAR N, et al. Attention is all you need[J]. Advances in Neural Information Processing Systems, 2017, 30: 5998-6008.

[2] KENTON J D M W C, TOUTANOVA L K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding [C]// Proceedings of NAACL-HLT. Minneapolis, 2019: 4171-4186.

[3] RADFORD A, NARASIMHAN K, SALIMANS T, et al. Improving language understanding by generative pretraining [EB/OL]. (2018-06-11) [2024-06-11]. https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.

[4] CHANG Y, WANG X, WANG J, et al. A survey on evaluation of large language models[J]. ACM Transactions on Intelligent Systems and Technology, 2023, 39: 1-45.

[5] RADFORD A, WU J, CHILD R, et al. Language models are unsupervised multitask learners[EB/OL]. (2019-02-14) [2024-06-11]. https://insightcivic.s3.us-east-1.amazonaws.com/language-models.pdf.

[6] BROWN T, MANN B, RYDER N, et al. Language models are few-shot learners[J]. Advances in Neural Information Processing Systems, 2020, 33: 1877-1901.

[7] ACHIAM J, ADLER S, AGARWAL S, et al. Gpt-4 technical report [EB/OL]. (2024-03-04) [2024-06-11]. https://cdn.openai.com/papers/gpt-4.pdf.

[8] TOUVRON H, LAVRIL T, IZACARD G, et al. Llama: Open and efficient foundation language models[EB/OL]. (2023-02-17) [2024-06-11]. https://arxiv.org/abs/2302.13971.

[9] BAI J, BAI S, CHU Y, et al. Qwen technical report [EB/OL]. (2023-09-28) [2024-06-11]. https://arxiv.org/abs/2309.16609.

[10] ZENG A, LIU X, DU Z, et al. GLM-130B: An Open Bilingual Pretrained Model[C]// International Conference on Learning Representations(ICLR). Kigali Rwanda, 2023.

[11] BI X, CHEN D, CHEN G, et al. Deepseek llm: Scaling open-source language models with longtermism[EB/OL]. (2024-01-05) [2024-06-11]. https://arxiv.org/abs/2401.02954.

[12] ROZIERE B, GEHRING J, GLOECKLE F, et al. Code llama: Open foundation models for code [EB/OL]. (2024-01-31) [2024-06-11]. https://arxiv.org/abs/2308.12950.

[13] LOZHKOV A, LI R, ALLAL L B, et al. StarCoder 2 and The Stack v2: The Next Generation[EB/OL]. (2024-02-29) [2024-06-11]. https://arxiv.org/abs/2402.19173.

[14] GUO D, ZHU Q, YANG D, et al. DeepSeek-Coder: When the Large Language Model Meets Programming-The Rise of Code Intelligence[EB/OL]. (2024-01-26) [2024-06-11]. https://arxiv.org/abs/2401.14196

[15] CHEN M, TWOREK J, JUN H, et al. Evaluating large language models trained on code[EB/OL]. (2021-07-14) [2024-06-11]. https://arxiv.org/abs/2107.03374.

[16] AUSTIN J, ODENA A, NYE M, et al. Program synthesis with large language models[EB/OL]. (2021-08-16) [2024-06-11]. https://arxiv.org/abs/2108.07732

[17] DONG Y, DING J, JIANG X, et al. Codescore: Evaluating code generation by learning code execution[EB/OL]. (2023-12-01) [2024-06-11]. https://arxiv.org/abs/2301.09043

[18] WOOLDRIDGE M, JENNINGS N R. Intelligent agents: Theory and practice[J]. The knowledge Engineering Review, 1995, 10(2): 115-152.

[19] PARK J S, O'BRIEN J, CAI C J, et al. Generative agents: Interactive simulacra of human behavior[C]// Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology. San Francisco, 2023: 1-22.

[20] WEI J, WANG X, SCHUURMANS D, et al. Chain-of-thought prompting elicits reasoning in large language models[J]. Advances in Neural Information Processing Systems, 2022, 35: 24824-24837.

[21] KOJIMA T, GU S S, REID M, et al. Large language models are zero-shot reasoners[J]. Advances in Neural Information Processing Systems, 2022, 35: 22199-22213.

[22] SONG C H, WU J, WASHINGTON C, et al. Llm-planner: Few-shot grounded planning for embodied agents with large language models[C]// Proceedings of the IEEE/CVF International Conference on Computer Vision. Paris, 2023: 2998-3009.

[23] YAO S, ZHAO J, YU D, et al. ReAct: Synergizing Reasoning and Acting in Language Models[C]// International Conference on Learning Representations(ICLR). Kigali Rwanda, 2023.

[24] SHINN N, CASSANO F, GOPINATH A, et al. Reflexion: Language agents with verbal reinforcement learning[J]. Advances in Neural Information Processing Systems, 2024, 36.

[25] BROWN T, MANN B, RYDER N, et al. Language models are few-shot learners[J]. Advances in Neural Information Processing Systems, 2020, 33: 1877-1901.

[26] WEI J, BOSMA M, ZHAO V, et al. Finetuned Language Models are Zero-Shot Learners[C]// International Conference on Learning Representations(ICLR). 2021.

[27] LI G, HAMMOUD H, ITANI H, et al. Camel: Communicative agents for "mind" exploration of large language model society [J]. Advances in Neural Information Processing Systems, 2023, 36: 51991-52008.

[28] RUAN J,CHEN Y H,ZHANG B,et al. TPTU:Task Planning and Tool Usage of Large Language Model-based AI Agents [C]// NeurIPS 2023 Foundation Models for Decision Making Workshop. New Orleans,2023.

[29] JIANG X,DONG Y,WANG L,et al. Self-planning code generation with large language model[EB/OL]. (2024-05-31) [2024-06-11]. https://arxiv.org/abs/2303.06689.

[30] SCHICK T,JANE A Y,JIANG Z,et al. PEER:A Collaborative Language Model[C] // International Conference on Learning Representations(ICLR). 2022.

[31] WU Q,BANSAL G,ZHANG J,et al. Autogen:Enabling next-gen LLM applications via multi-agent conversation framework [EB/OL]. (2023-10-03) [2024-06-11]. https://arxiv.org/abs/2308.08155.

[32] HONG S,ZHUGE M,CHEN J,et al. MetaGPT:Meta Programming for Multi-Agent Collaborative Framework[C] // International Conference on Learning Representations(ICLR). Vienna,2024.

XIA Peng, born in 1988, Ph. D, senior engineer, is a member of CCF (No. U6207M). His main research interests include multimodal large models and multi-agent collaboration driven by large models.

QI Ji, born in 1978, Ph.D, senior engineer. His main research interests include pre-trained large models, big data, and cloud computing.