大模型函数调用机制下的计算优化

肖明魁[1] 张 亮[1]
XIAO Mingkui ZHANG Liang

摘 要 随着大语言模型技术的快速发展，其在众多领域展现出强大的能力，但也暴露出诸多不容忽视的局限性，尤其是在数学运算方面。因此，文章针对大语言模型在处理复杂数学运算时出现的不准确性问题进行了深入探讨，并提出了一种基于 Function Calling 的解决方案，以期改善模型在数学运算任务中的表现。首先介绍了大语言模型在数学运算中存在的问题，对超长数字的处理能力有限、缺乏精确的计算逻辑等。这些问题极大限制了大语言模型的应用范围，特别是在需要高精度计算的场景下。为更好地解决上述问题，文章重点研究了函数调用 Function Calling 机制，包括如何定义外部函数库、如何配置模型参数以启用 Function Calling 功能，以及如何构建高效的交互流程以确保模型能够准确地识别用户意图并调用正确的函数。通过对实验结果的分析，证明 Function Calling 机制能有效提升大语言模型在数学运算方面的准确性和实用性，其成果为进一步拓展大语言模型的应用场景提供了新的方向和技术支撑。

关键词 机器学习；大模型；函数调用；意图识别

doi：10.3969/j.issn.1672-9528.2025.03.005

0 引言

近年来，随着深度学习技术的进步和大规模计算资源的普及，大语言模型（large language models, LLMs）成为自然语言处理领域的一项重要突破。这些模型通常基于 Transformer 架构，通过在海量文本数据上进行无监督预训练，能够掌握丰富的语言知识和模式[1-2]。自 2018 年 Google 发布 BERT 模型以来，一系列更为先进的模型相继问世，如 GPT-4、GLM-4 等，这些模型的参数规模达到了数十亿乃至千亿级别，展现了前所未有的语言理解和生成能力。

尽管大语言模型在许多方面表现出色，但同样也面临着一些固有的局限性。这些局限性不仅限制了模型的实用性，也阻碍了其进一步发展。其中，最突出的挑战之一是在数学运算方面的不足。尤其在进行复杂数学运算时[3]。例如，对于超长数字的乘法运算，模型往往会计算出错误的结果。原因是模型内部计算机制并不适合进行精确的数学运算，且模型训练的数据集中包含较少这类特定的数学计算任务。因此，模型倾向于根据上下文推测答案而非进行实际的数学计算。

这种局限性限制了大语言模型在需要高精度计算的场景下的应用，例如财务分析、科学计算等。为克服这一局限，诸多研究者正在探索不同的解决方案。

1 国内外近年主要技术综述

随着人工智能技术的飞速发展，大语言模型在自然语言处理领域取得了显著成果。LLMs 不仅在文本生成、机器翻译、问答系统等方面表现出色，而且在数值计算领域也展现出巨大的潜力。

Brown 等人[4] 指出，通过提示工程， GPT-3 在简单的数学运算（如加减乘除）中表现出色，尤其是在少样本学习的情况下。例如，在两位数加法和减法中可以达到 100% 的准确率，在三位数运算中也表现出较高准确率。在更复杂的数学运算（如多位数乘法和除法）中准确率较低，但仍然能够进行一些推理和计算。由此证明， GPT-3 在理解数学概念和进行高级数学推理方面存在局限性。

Yuan 等人[5] 评估了多个 LLMs 在算术表达式计算任务上的性能，并提出了 MATH 数据集，用于测试模型在数值计算上的能力。研究认为，大语言模型的算术能力受多种因素的影响，包括分词器、预训练语料库、指令微调、模型参数数量、提示语、上下文学习、思维链和模型架构等，通过优化这些因素，可以提升大语言模型在算术任务上的表现。结果表明，LLMs 在算术任务上表现出色，但仍然存在精度问题和对于某些类型的问题解决能力有限。

1. 江苏经贸职业技术学院信管处 江苏南京 210000

[基金项目] 江苏经贸职业技术学院校级重点课题（JSJMZH24005）

Fu 等人[6]介绍了 KwaiYiiMath 模型，该模型通过监督微调和基于人类反馈的强化学习来提高数学推理能力。KwaiYiiMath 通过多种方法来提高计算精确度，包括使用高质量的训练数据、有效的微调方法、数据质量控制、模型设计和提示设计。这些方法共同作用，使得 KwaiYiiMath 能够在数学推理和计算方面表现出色，并给出准确的答案。通过实验分析模型在多个数学任务上基准测试结果，证明了 LLMs 通过适当的训练可以显著提高数值计算精度。

Yang 等人[7]评估了大型语言模型（LLM）在数学推理方面的能力，并设计了一个名为 MathGLM 的模型来提升 LLM 在数学问题解决方面的表现。MathGLM 将复杂的数学表达式分解成一系列简单的步骤，逐步计算并得出答案。这种方法与人类解决数学问题的过程相似，能够有效提高准确性。此外，MathGLM 能够逐步学习更复杂的算术表达式，使其可以处理更大范围的数字和更复杂的运算。MathGLM 还通过在大量算术数据集上进行预训练，并针对特定类型的数学问题进行微调，以提升其解决问题的能力。

Zhao 等人[8]提出了一种名为 JiuZhang 2.0 的统一中文预训练语言模型，旨在解决多任务数学问题。研究团队构建了一个混合专家（MoE）架构来建模数学文本，捕捉任务间的通用数学知识。MoE 架构将数学知识分解到不同的专家网络中，每个专家网络专注于特定类型的数学知识（例如代数、几何、三角函数等）。当模型遇到新的数学问题时，会根据问题的类型和内容，将问题分解成多个子问题，并分别交给不同的专家网络进行处理。这种知识分解和复用机制有助于提高模型对复杂数学问题的理解和求解精度。

以上研究涵盖了当前大模型领域多项重要技术，涉及提示工程、思考链、预训练、监督微调以及混合模型等，充分展示了大语言模型在数学问题解决方面的巨大潜力，同时也反映了当前模型在精确计算方面的局限性。首先，模型的计算能力有限，尤其是在处理复杂的数学问题时，如多步计算、符号操作和精确的数值计算；其次，模型的训练数据集通常缺乏高质量的数学问题样本，导致模型在实际应用中的表现不如预期；最后，大模型在处理复杂计算任务时，需要消耗大量算力资源，大幅增加计算成本。

因此，本文提出一种新型方案，通过大模型的函数调用（Function Calling）机制，在处理特定任务时调用外部函数库，使其能够处理更复杂的计算任务，从而弥补模型在计算能力上的不足。

2 Function Calling 机制

函数调用（Function Calling）是一种使大型语言模型能够调用外部函数或服务的功能。通过这种方式，模型不仅能够根据自身的知识库生成响应，还能通过调用特定的函数或 API 来执行特定的任务或获取最新的信息。Function Calling 允许用户以 JSON 格式向模型描述函数，模型则可以根据上下文和问题内容决定是否需要调用这些函数。

2.1 Function Calling 的发展历程

Function Calling 的概念最初是由 OpenAI 在其 API 更新中引入，这使得 GPT 等模型能够与外部世界交互。随着时间的推移，各公司和研究机构也开始采用类似的技术来增强他们的语言模型。在 Function Calling 出现之前，语言模型主要依赖于其训练数据集来生成响应，这意味着它们无法访问实时信息或执行特定的操作[9-10]。随着技术的发展，Function Calling 成为了扩展模型功能的关键。OpenAI 在 2023 年 6 月 13 日的更新中正式推出了这项功能，而智谱 AI 也在其 GLM-4 模型中实现了类似的功能。Function Calling 机制为大语言模型提供了与外部世界互动的能力，从而使模型能够执行更多复杂的任务，并为用户提供更有价值的服务[11-12]。通过这种方式，模型不仅可以提供信息，还可以作为工具帮助用户执行特定的操作，从而提高了模型的实际应用价值。

2.2 Function Calling 技术实现

Function Calling 的基础框架涉及到几个关键组成部分：定义外部函数库、函数定义的 JSON Schema 格式、配置模型参数以启用 Function Calling 功能以及 Function Calling 的工作流程。外部函数库可以包括简单的自定义函数，也可以是封装了外部 API 的服务。例如，可以是调用 Google 搜索 API 的函数，或者是获取天气信息的函数[13]。为使模型能够理解和调用这些函数，需要以 JSON Schema 格式定义每个函数。JSON Schema 中需要包含函数的名称、输入参数和输出格式等信息。这些信息可以帮助模型了解函数的功能及如何使用[14-15]。函数参数的定义应当清晰明了，遵循一致的命名规范，避免使用相似或易混淆的名称。此外，函数描述应当尽可能详尽，以便模型更好地理解函数的功能。

Function Calling 工作流程如图 1 所示，图中展示了 Function Calling 功能如何在模型中被触发和执行，以及如何与外部函数库和模型配置交互，这种机制允许模型利用外部数据或逻辑来增强其生成的文本。在整个流程中，模型充当中介的“角色”，不仅需要理解用户的输入，还需要知道何时以及如何调用外部函数，并将函数的输出转换为用户可以理解的形式。

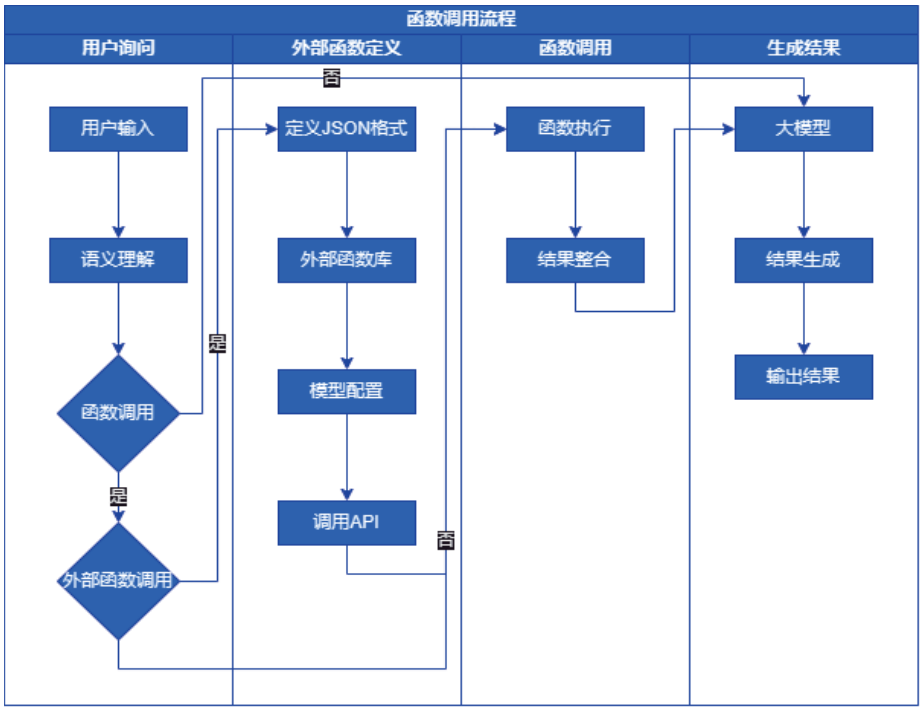

图 1 Function Calling 工作流程

2.3 Function Calling 结构

实现 function calling 功能，需要使用大模型的 client.chat.completions.create() 函数，该函数包含两个重新参数：tools 参数和 tool_choice 参数。

tools 参数是一个列表，其中每个元素是一个字典，描述了一个可用的外部函数或工具。每个字典通常包含以下键值对：

name：函数的名称。

description：函数的描述。

parameters：函数所需的参数描述，应遵循 JSON Schema 格式。

tool_choice 参数决定了模型是否以及如何调用工具。它有两个主要的取值：

auto：模型将根据上下文自动选择合适的工具进行调用。

指定工具名称：模型将直接调用指定名称的工具，而不是自动选择。

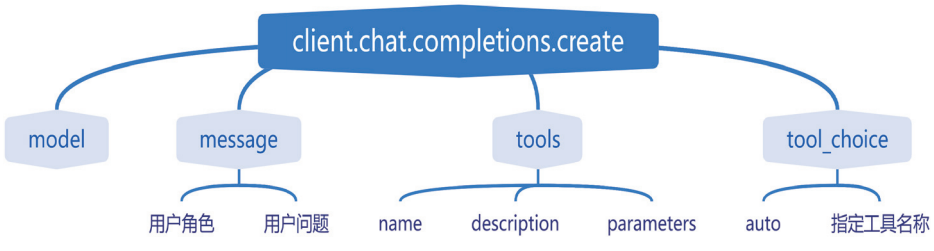

图 2 client.chat.completions.create 函数结构

函数结构如图 2 所示。

3 实验与案例分析

为验证 Function Calling 机制对提升大语言模型在处理复杂数学运算任务上的效果，本文选取大语言模型 GLM-4（general language model-4）作为研究对象。GLM-4 是一款具备强大语言理解和生成能力的语言模型，但如同其他大型语言模型一样，在处理特定类型的数学运算时可能会遇到挑战。本实验旨在通过引入 Function Calling 机制来解决这些问题，选择涉及复杂数学运算的任务作为测试用例，尤其是传统大语言模型难以直接处理的任务，例如大数值计算。这些任务能够有效地评估 Function Calling 机制的有效性及其对模型性能的影响。

3.1 实验设计

本实验中，代码将调用两种不同性能的大模型（"glm-4-flash" 和 "glm-4-plus"），随机生成两个整数，分别进行加法、减法、乘法、除法、平方、开方及绝对值运算，连续计算 3 次并输出结果。实验采用对比研究方法，分别用以验证未启用 Function Calling 和启用 Function Calling 的两种不同情况的结果。

首先，在不使用 Function Calling 的情况下，实验通过直接调用大模型方法计算两个整数的运算结果，小数点保留两位，部分代码如下所示：

```
# 随机生成两个十位整数
num1 = random.randint(1000, 9999)
num2 = random.randint(1000, 9999)

# 构建请求的消息
operations = [" 乘以 ", " 除以 ", " 加 ", " 减 ", " 平方 ", " 平方根 ", " 绝对值 "]
messages = []

# 添加计算 num1 的二次幂的消息
messages.append({"role": "user", "content": f " 请帮我计算一下 {num1} 的二次幂是多少 "})
```

```
    # 添加其他操作的消息
    for operation in operations:
        if operation in [" 平方根 ", " 绝对值 "]:
            messages.append({"role": "user", "content": f " 请帮我计算一下 {num1} 的 {operation} 是多少？ "})
        else:
            messages.append({"role": "user", "content": f " 请帮我计算一下 {num1} {operation} {num2} 等于多少？ "})

    # 调用模型并打印结果
    models = ["glm-4-flash",  "glm-4-plus"]
    count = 0
    for model in models:
     for message in messages:
        for _ in range(3):  # 假设想要重复请求 3 次
            response = client.chat.completions.create(
                model=model,
                messages=[message],
            # 注意这里需要将 message 作为列表传入
            )
            print(response.choices[0].message.content)
```

输出结果如表 1 和表 2 所示。

表 1 调用“glm-4-flash”大模型的计算结果

	8 722×1 273	8 722 / 1 273	8 722 +1 273	8 722 −1 273
第一次计算结果	11 090 826	6.88	9 995	7 449
第二次计算结果	11 069 066	6.886	9 995	7 449
第三次计算结果	11 006 426	6.88	9 995	7 449
正确结果	11 103 106	6.85	9 995	7 449
准确率	0%	0%	100%	100%
	$8\ 722^2$	Sqrt(8 722)	Abs(8 722)	
第一次计算结果	75 845 024	93.03	8 722	
第二次计算结果	76 101 884	93.03	8 722	
第三次计算结果	7 634 092 504	93.24	8 722	
正确结果	76 073 284	93.39	8 722	
准确率	0%	0%	100%	

表 2 调用“glm-4-plus”大模型的计算结果

	8 722×1 273	8 722 /1 273	8 722 +1 273	8 722 −1 273
第一次计算结果	11 070 506	6.845	9 995	7 449
第二次计算结果	11 064 706	6.849	9 995	7 449
第三次计算结果	11 047 806	6.845	9 995	7 449
正确结果	11 103 106	6.85	9 995	7 449
准确率	0%	100%	100%	100%
	$8\ 722^2$	Sqrt(8 722)	Abs(8 722)	
第一次计算结果	75 946 184	93.44	8 722	
第二次计算结果	75 967 284	93.42	8 722	
第三次计算结果	75 941 584	93.42	8 722	
正确结果	76 073 284	93.39	8 722	
准确率	0%	0%	100%	

以上结果表明，实验在调用两种不同性能的大模型，直接进行两位整数计算时，只有在处理简单的计算，如加、减和绝对值运算，才能做到100%的准确率，而面对更复杂运算，如乘、除、平方和开方运算，计算精确度明显差强人意，即使更换大模型，提升效果依然有限。

接下来，实验将根据前文综述中所提到的提示工程和思考链等模型优化方法，进一步完善本实验，观察实际效果。

尽管大模型根据预先定义的提示工程和思考链方法，对复杂计算分步处理并汇总结果，然而受限于模型本身性能，计算精度无法得到保证，甚至有可能在推理过程出现思考中断现象，从而无法生成答案。实验最终结果如表 3 和表 4 所示。

表 3 调用“glm-4-flash”大模型的推理结果

	8 722×1 273	8 722 / 1 273	8 722 +1 273	8722 -1273
第一次计算结果	11 165 926	无	9 795	无
第二次计算结果	无	无	无	无
第三次计算结果	无	无	无	7589
正确结果	11 103 106	6.85	9 995	7449
准确率	0%	0%	0%	0%

表 3（续）

	$8\ 722^2$	Sqrt(8 722)	Abs(8 722)	
第一次计算结果	无	无	8 722	
第二次计算结果	无	无	8 722	
第三次计算结果	无	无	8 722	
正确结果	76 073 284	93.39	8 722	
准确率	0%	0%	100%	

表 4 调用“glm-4-plus”大模型的推理结果

	8 722 ×1273	8 722 /1 273	8 722 +1 273	8 722−1 273
第一次计算结果	11 042 606	6.86	9 995	7449
第二次计算结果	11 109 206	6.85	9 995	7449
第三次计算结果	11 083 406	6.85	9 995	7449
正确结果	11 103 106	6.85	9 995	7449
准确率	0%	66%	100%	100%
	$8\ 722^2$	Sqrt(8 722)	Abs(8 722)	
第一次计算结果	76 073 284	93.06	8 722	
第二次计算结果	76 182 084	93.39	8 722	
第三次计算结果	75 973 284	93.42	8 722	
正确结果	76 073 284	93.39	8 722	
准确率	33%	33%	100%	

实验表明，比较两种不同性能的大模型，尤其是高性能模型，加入提示工程和思考链的方式，在一定程度上可以提升复杂运算的精度。然而，对于性能较弱的模型，由于推理能力不足，思考过程中时常出现运行中断现象，不但计算错误率较高，甚至无法生成结果。实验进一步拓展，调用高性能大模型，比较不同位数的两个整数的计算准确率，结果如表 5 所示。

表 5 不同位数的两个整数运算结果

	四位整数	六位整数	八位整数	十位整数
加 / 减	100%	100%	66%	0%
乘 / 除	66%	33%	33%	0%
开方	33%	33%	0%	0%
乘方	33%	0%	0%	0%
绝对值	100%	100%	100%	100%

综合实验表明，未启用 Function Calling 的情况下，调用不同大模型进行数值计算，尤其对于大整数或复杂计算，不仅精度差，每次计算结果也不尽相同，即使通过提示工程和思考链优化，准确率依然无法满足实际工作要求。因而，本文采用函数调用的方法，可以更好地解决大数计算问题。

由于大语言模型本身不具备精确计算大数值的能力，而是通过调用外部计算函数实现运算功能，因此，代码核心设计思想是构建一个包含各种运算功能的外部函数字典，并将其转换为 JSON Schema 格式，而后通过大模型自身的语义理解能力，识别用户输入的计算要求，并从 JSON Schema 格式的字典中检索调用相应的运算功能，最终返回问题答案。原理如图 3 所示。

图 3 大模型运算架构

代码首先定义了 7 个外部运算函数，分别为加、减、乘、除、幂、开方和绝对值 7 种运算功能，参数包括两个浮点数 *a* 和 *b*，函数返回值也为浮点格式，并且每个函数内部增加了计算功能的注释，以下代码所示为其中的加法函数：

```
def add(a: float, b: float) -> float:
    "" "
    计算两个浮点数的和。

    参数 :
    a (float): 第一个加数。
    b (float): 第二个加数。

    返回 :
    float: 两个参数的和。
    "" "
    return a + b
```

由于大模型无法直接解析外部函数意义，需要构建一个转换函数，将 7 个外部运算函数转换为 JSON Schema 格式的

字典，便于大模型识别和理解用户意图。转换函数从 7 个外部运算中提取重要信息，包括函数名、功能描述、参数名等，整理输出为 JSON Schema 格式，并可作为大模型对话函数 client.chat.completions.create() 中的 message 参数。例如，经过 JSON Schema 格式转换后的加法函数输出结果如下所示：

```
{
    "type": "function",
    "function": {
      "name": "add",
        "description": "计算两个浮点数的和。\n\n    参数 :\n    a (float): 第一个加数。\n    b (float): 第二个加数。\n\n    返回 :\n    float: 两个参数的和。",
      "parameters": {
        "type": "object",
        "properties": {
          "a": {
          "description": "Convert a string or number to a floating point number, if possible.",
          "type": "<class 'float'>"
          },
          "b": {
          "description": "Convert a string or number to a floating point number, if possible.",
          "type": "<class 'float'>"
          }
        },
        "required": []
      }
    }
  },
```

依照上述方法，分别将自定义的 7 种运算函数转换为 JSON Schema 格式，并保存在一个名为 generate_json_description() 函数字典之中。用户需要运行计算任务时，可以通过大模型对话函数中的 tools 参数，选择字典中相应的计算功能，实现最终计算目标。

完成以上步骤之后，大模型可以根据用户提出的任务，检索和调用字典中对应的运算功能，例如，用户提问，“请帮我计算一下 1 250 908 070 ^{2} 等于几”，大模型自动识别并获取参与计算的两个参数，底数 1 250 908 070，指数为 2，通过索引函数从字典中调用幂运算函数，计算并得到正确结果 1 564 770 999 591 124 900。

为确保模型能够充分利用之前调用函数得到的信息来生成最终的回答或执行任务，代码还需进行第二次模型调用，可以让模型基于新的信息做出更准确的判断或给出更具体的回答。二次调用函数 client.chat.completions.create() 的用户提示内容是：第一次用户提问 + 第一次模型输出结果，即“请帮我计算一下 1 250 908 070^2 等于几 1 564 770 999 591 124 900”，代码最终输出结果为：“好的，根据您的要求，我使用了 Python 内置的 power 函数来计算 1 250 908 070 的平方。计算结果为 1 564 770 999 591 124 900。请问还有其他需要我帮忙的地方吗？”程序完整流程如图 4 所示。

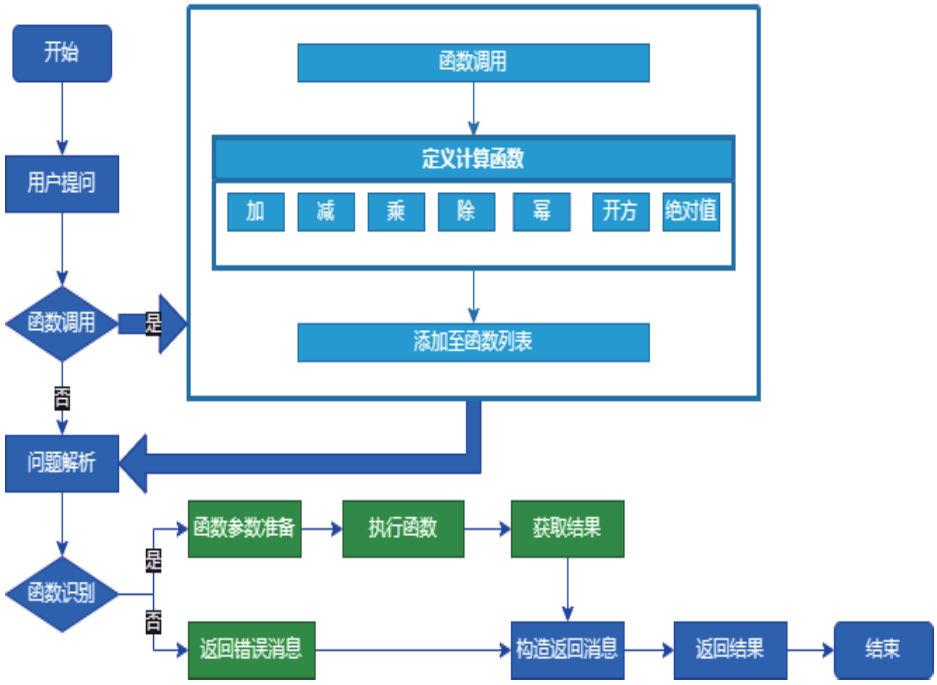

图 4 程序完整流程图

3.2 实验结果

经过大量试验和反复测试，大模型对于大数值计算处理效率得到显著提升，代码随机生成两个九位整数，经过 7 种运算，输出准确率为 100%，并且每次运算结果都保持一致，完全满足实际工作需要。实验结果如表 6 所示。

3.3 讨论分析

Function Calling 机制使大语言模型能够调用外部函数，解决了模型在数学运算、获取最新信息等方面的不足，从而提高了模型的实用性。此外，通过调用外部 API，模型可以实现实时搜索、自动邮件回复等功能，极大扩展了模型的应用场景，使其能够更好地服务于各种实际需求。

表 6 启用 Function Calling 的计算结果

	823 788 678×633 033 558	823 788 678 / 633 033 558	823 788 678 + 633 033 558	823 788 678 - 633 033 558
第一次计算结果	521 485 877 874 456 324	1.301 334 925 438 502 5	1 456 822 236	190 755 120
第二次计算结果	521 485 877 874 456 324	1.301 334 925 438 502 5	1 456 822 236	190 755 120
第三次计算结果	521 485 877 874 456 324	1.301 334 925 438 502 5	1 456 822 236	190 755 120
正确结果	521 485 877 874 456 324	1.301 334 925 438 502 5	1 456 822 236	190 755 120
准确率	100%	100%	100%	100%
	$823\ 788\ 678^2$	Sqrt(823 788 678)	Abs(823 788 678)	
第一次计算结果	678 627 786 000 987 684	28 701.719 077 435 067	823 788 678	
第二次计算结果	678 627 786 000 987 684	28 701.719 077 435 067	823 788 678	
第三次计算结果	678 627 786 000 987 684	28 701.719 077 435 067	823 788 678	
正确结果	678 627 786 000 987 684	28 701.719 077 435 067	823 788 678	
准确率	100%	100%	100%	

然而，Function Calling 机制的局限性也不容忽视，虽然 Function Calling 机制提升了模型的能力，但也增加了对外部 API 和服务的依赖。如果服务不可用或不稳定，会影响到模型的整体性能。并且，调用外部 API 可能会涉及到数据传输和处理，带来了安全性和隐私保护方面的挑战。例如，敏感信息的泄露风险和 API 滥用等问题都需要得到妥善解决。

4 结语

Function Calling 机制为大语言模型提供了一种有效的途径来克服数学运算中的局限性。通过调用外部函数，模型能够在数学运算方面表现出更高准确性和稳定性。实验结果表明，在启用 Function Calling 后，模型能够稳定地执行复杂的数学运算任务，如大数乘法和除法，而无需担心结果的准确性。尽管 Function Calling 已经显示出了其在解决数学运算局限性方面的潜力，但仍有许多方面值得进一步研究。例如，如何设计更智能的函数选择策略，以提高函数调用的准确性和效率；如何解决安全性和隐私保护问题，以确保数据的安全传输和处理；以及如何优化 Function Calling 的性能，使其在大规模部署时更加高效可靠。随着 Function Calling 机制的广泛应用，需要开发更多实用的外部函数来满足各种实际需求。包括金融计算、科学计算、数据分析等领域的专用函数。通过不断丰富函数库，可以进一步扩展大语言模型的应用场景，使其在更多领域发挥重要作用。

参考文献：

[1] 解勉 , 陈刚 , 余晓晗 . 基于大语言模型的论文检索与分析方法研究 [J]. 计算机技术与发展 , 2024,34(12):116-124.

[2] 荣国伟 , 孙宝 , 王文婧 , 等 . 大语言模型在数学建模领域的应用探索 [J]. 科技创新与应用 , 2024,14(22): 18-21.

[3] 齐思洋 , 胡慧云 , 李洪冰 , 等 . 融合大语言模型的领域问答系统构建方法 [J]. 北京邮电大学学报 , 2024,47(4):50-56.

[4] BROWN T B, MANN B, RYDER N, et al. Language models are few-shot learners[C]//Proceedings of the 34th International Conference on Neural Information Processing Systems. NewYork: ACM, 2022: 1877-1901.

[5] YUAN Z, YUAN H Y, TAN C Q, et al. How well do large language models perform in arithmetic tasks?[DB/OL].(2023-03-16)[2024-02-22].https://doi.org/10.48550/arXiv.2304.02015.

[6] FU J Y, LIN L, GAO X Y, et al. KwaiYiiMath: technical report[DB/OL]. (2023-10-19)[2024-03-10].https://doi.org/10.48550/arXiv.2310.07488.

[7] YANG Z, DING M, LÜ Q S, et al. GPT can solve mathematical problems without a calculator[DB/OL]. (2023-09-12)[2024-02-22].https://doi.org/10.48550/arXiv.2309.03241.

[8] ZHAO W X, ZHOU K, ZHANG B C, et al. JiuZhang 2.0: a unified chinese pre-trained language model for multi-task mathematical problem solving[C]// Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining.NewYork:ACM,2023:5660-5672.

[9] 赵鑫 , 窦志成 , 文继荣 . 大语言模型时代下的信息检索研究发展趋势 [J]. 中国科学基金 , 2023,37 (5):786-792.

基于图神经网络用于预测原子相互作用研究

贺浩杰 [1]
HE Haojie

摘 要 近年来，利用图神经网络（GNN）模型方法在计算化学（CompChem）中越来越受到研究者的关注。在参考和使用适当的由从头算（Ab Initio）方法训练数据情况下，利用图神经网络模型可以精确地求解化学系统性质，如能量和力场，避免了求解电子薛定谔方程（schrodinger equation，SE）的需求。但图神经网络模型在处理边缘节点效果不佳。基于此，文章基于图神经网络数值预测类任务具有处理复杂关系和依赖性，提出一个深度学习网络模型（external attention newton net，EANTNet）。首先优化图神经网络提取单颗原子的特征方式，让神经网络提取重要的特征。其次，基于外部注意力机制的思想，重新构造能量聚合器，用来聚合图神经网络提取的特征。实验结果表明，在数据集 MD17 中，对比 NewtonNet 网络模型，EANTNet 网络模型不仅能在化学系统中提取更好的特征，并且提升了化学系统中的能量和力场的预测精度，分别平均提高了 3.39% 和 1.53%。

关键词 EANTNet；能量聚合器；力场；预测精度

doi：10.3969/j.issn.1672-9528.2025.03.006

0 引言

在分子动力学模拟中，一个最新的挑战是：如何构建更精确的力场来描述分子间相互作用 [1]。

正如狄拉克在 1929 年指出的那样，原则上，薛定谔方程包含了描述整个化学过程所必需的一切 [2]。然而，SE 是一个高维方程，变量是化学系统中所有电子的坐标。例如，在一个化学系统中包含 m 颗电子，那么变量的维度就是 $3m$ 维，而一个普通化学系统中，电子数量是以千万为计数单位，需要巨大的算力。超高的计算资源投入和超低的产出是不可接受的。因此，近似方程被创建来寻找近似解。尽管这些近似方程降低了求解电子 SE 的计算要求，但近似方程所要求的精度是有限的，而且只能处理有限数量的原子。面对大规模系统仿真问题，如何构造力场是解决这一挑战的关键。机器学习在建立力场方面有一个巨大的优势。机器学习方法不依赖于任何预设的化学键或相互作用的知识，而只通过对参考数据的训练，来找到适当的力场 [3]。随着深度学习在机器学习领域的快速发展，特别是图神经网络的发展，为构建精确力场提供了一种新的方案。图神经网络（GNNS）在构建力场方面的优势在于能够直接利用分子和材料的图结构，提供灵活和可扩展的表示，同时通过引入物理先验和有效的学习

1. 贵州财经大学信息学院 贵州贵阳 550025

[10] 宋时磊，杨逸云 . 应用场景、风险与前景：ChatGPT 类大语言模型时代的学术出版 [J]. 出版科学 , 2023, 31 (5): 76-84.
[11] 陈郅睿，陆雪松 . 基于开源代码大语言模型提示的学生代码修复 [J]. 华东师范大学学报 (自然科学版),2024(5):93-103.
[12] 毛秋力，沈庆飞，李秀红 . 面向算力中心的大模型推理优化技术 [J]. 质量与认证 ,2024(9):40-44.
[13] 叶国林，郭家文，张晓宇，等 . 基于开源大语言模型的智能体算法研究 [J]. 中国信息界 ,2024(4):161-163.
[14] 李荣涵，浦荣成，沈佳楠，等 . 基于思维链的大语言模型知识蒸馏 [J]. 数据采集与处理 ,2024,39(3):547-558.
[15] 陈小平 . 大模型关联度预测的形式化和语义解释研究 [J]. 智能系统学报 ,2023,18(4):894-900.

【作者简介】

肖明魁（1979—），男，江苏南京人，本科，工程师，研究方向：机器学习、大模型。

张亮（1983—），男，江苏无锡人，硕士研究生，高级工程师，研究方向：信息化管理。

（收稿日期：2024-11-13）