软件学报 ISSN 1000-9825, CODEN RUXUEW
[doi: 10.13328/j.cnki.jos.007462] [CSTR: 32375.14.jos.007462]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于自适应知识蒸馏的代码大模型轻量化*

舒善富[1], 刘 超[1], 孙毓忠[2], 张洪宇[1], 高翠芸[3], 张小洪[1]

[1](重庆大学 大数据与软件学院, 重庆 401331)

[2](中国科学院 计算技术研究所, 北京 100190)

[3](哈尔滨工业大学 (深圳) 计算机科学与技术学院, 广东 深圳 518055)

通信作者: 刘超, E-mail: liu.chao@cqu.edu.cn

摘 要: 以大语言模型 (large language model, LLM) 为基座的软件编程助手 (如 Copilot), 能够显著提升程序员开发效率, 但 LLM 的计算和存储需求大、本地化部署难. 构建轻量化小参数 LLM 能够满足计算、存储、部署需求, 但其代码生成的精度损失比大参数 LLM 大. 知识蒸馏 (knowledge distillation, KD) 技术, 让小参数 LLM (学生模型) 在目标训练数据集上拟合大参数 LLM (教师模型) 的生成分布, 降低代码生成精度损失. 人工智能领域前沿的 KD 技术基于 Kullback-Leibler (KL) 散度损失函数, 度量并缩小因学生/教师模型的生成分布差异导致的精度损失, 但学生模型难以学习教师模型的趋零分布区域. 随后, 学者利用反向 KL 散度损失函数 (RKL) 解决该趋零分布区域的学习问题. 研究发现, RKL 在高概率分布区域存在学习问题, 与 KL 散度损失函数存在互补性; 对于一些数据, 教师模型生成质量低, 导致学生模型学习效果差. 提出一种自适应知识蒸馏 (adaptive knowledge distillation, AKD) 方法, 通过 prompt 提升教师模型的生成质量, 并构造自适应损失函数, 根据学生/教师模型之间的生成分布差异自适应调整学习的优先级, 确保学生模型在主要概率区域和趋零概率区域均具备学习能力. 基于 AKD 方法, 利用 StarCoder-1B/7B (学生/教师模型) 和 CodeAlpaca 数据, 训练了轻量化代码生成大模型, 并评估代码生成大模型的精度损失及生成代码的质量问题. 实验结果显示, 轻量化代码生成大模型规模降低 85.7%, 在 HumanEval 和 MBPP 数据集上, 任务提示明确的 prompt 可以提高教师模型的代码生成质量, 使训练的学生模型降低 6% 的平均精度损失; AKD 方法训练的模型较教师模型 (StarCoder-7B) 的平均精度损失为 17.14%, 较原始学生模型平均降低 30.6%; AKD 方法训练的模型较前沿的 KD 和 RKD 方法的精度损失平均降低 19.9%; 关于推理显存需求情况, KD 和 RKD 方法需要 54.7 GB, 而 AKD 方法仅增加 3 GB. 关于训练时间方面, AKD 方法所需训练时间增加 30%; 相较而言, 即使 KD 和 RKD 方法训练至相同时长, 他们的平均效果仅提升 3%, 相比 AKD 方法低 16.9%. 因此, AKD 方法增加的训练成本是值得的. 此外, 将 AKD 方法应用到 CodeLlama 和 CodeGen 系列模型, 相较前沿的 KD 及 RKD 方法的精度损失平均降低 19.2%, 证明了 AKD 方法的泛化能力.

关键词: 代码生成; 大语言模型; 知识蒸馏

中图法分类号: TP311

中文引用格式: 舒善富, 刘超, 孙毓忠, 张洪宇, 高翠芸, 张小洪. 基于自适应知识蒸馏的代码大模型轻量化. 软件学报. http://www.jos.org.cn/1000-9825/7462.htm

英文引用格式: Shu SF, Liu C, Sun YZ, Zhang HY, Gao CY, Zhang XH. Adaptive Knowledge Distillation for Lightweight Large Code Models. Ruan Jian Xue Bao/Journal of Software (in Chinese). http://www.jos.org.cn/1000-9825/7462.htm

* 基金项目: 国家自然科学基金 (62202074, 62372071); 中国博士后科学基金 (2022M710519); 重庆市技术创新与应用发展专项重点项目 (CSTB2023TIAD-STX0015, CSTB2022TIAD-KPX0068); 重庆市出站留 (来) 渝博士后择优资助项目 (2021LY23)
收稿时间: 2024-11-03; 修改时间: 2025-01-05, 2025-03-03, 2025-04-07; 采用时间: 2025-05-06; jos 在线出版时间: 2025-12-03

Adaptive Knowledge Distillation for Lightweight Large Code Models

SHU Shan-Fu[1], LIU Chao[1], SUN Yu-Zhong[2], ZHANG Hong-Yu[1], GAO Cui-Yun[3], ZHANG Xiao-Hong[1]

[1](School of Big Data & Software Engineering, Chongqing University, Chongqing 401331, China)

[2](Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China)

[3](School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen 518055, China)

Abstract: Software programming assistants based on large language models (LLMs), such as Copilot, significantly enhance programmer productivity. However, LLMs have large computing and storage requirements and are difficult to deploy locally. Building a lightweight, small LLM can meet computing, storage, and deployment requirements, but it leads to a greater accuracy loss in code generation compared to large LLMs. Knowledge distillation (KD) techniques allow small LLMs (student models) to approximate the output distributions of large LLMs (teacher models) on target training datasets, thus reducing accuracy loss in code generation. Cutting-edge KD techniques in artificial intelligence are based on the Kullback-Leibler (KL) divergence loss function, which measures and reduces accuracy loss due to discrepancies in the output distributions between student and teacher models. However, student models struggle to learn in the near-zero distribution regions of teacher models. Consequently, researchers have employed the Reverse KL (RKL) divergence loss function to address this issue in near-zero distribution regions. This study finds that RKL faces learning challenges in high-probability distribution regions and complements the KL divergence loss function. For some datasets, low-quality outputs from teacher models lead to poor learning outcomes for the student models. This study proposes an adaptive knowledge distillation (AKD) method that uses prompts to enhance teacher model output quality and constructs an adaptive loss function to adjust learning priorities based on the distributional differences between student and teacher models. This ensures the student model effectively learns in both primary and near-zero probability regions. Using the AKD method, this study trains a lightweight code generation model based on StarCoder-1B/7B (student/teacher models) and the CodeAlpaca dataset, evaluating accuracy loss and code quality issues. Experimental results show that the lightweight model size is reduced by 85.7%. On the HumanEval and MBPP data sets, prompts with clear instructions improve teacher model code generation quality, reducing the average accuracy loss of the trained student model by 6%. The AKD-trained model's average accuracy loss compared to the teacher model (StarCoder-7B) is 17.14%, a 30.6% reduction over the original student model. The AKD-trained model's accuracy loss is reduced by an average of 19.9% compared to state-of-the-art KD and RKD methods. Regarding inference memory requirements, the KD and RKD methods require 54.7 GB, while the AKD method only adds 3 GB. In terms of training time, the AKD method incurs a 30% increase. However, even when the KD and RKD methods are trained for the same duration, their average performance improves by only 3%, which is 16.9% lower than that of the AKD method. Therefore, the additional training cost of the AKD method is justified. Moreover, applying the AKD method to the CodeLlama and CodeGen model series reduces accuracy loss by an average of 19.2% compared to state-of-the-art KD and RKD methods, demonstrating the generalizability of the AKD method.

Key words: code generation; large language model (LLM); knowledge distillation (KD)

随着大语言模型 (large language model, LLM) 的发展, 如 GPT-4 [1]、OPT[2]和 LLaMA[3], 代码生成任务在软件工程领域取得了显著进展. 基于 LLM 的软件编程助手, 如 GitHub Copilot (https://copilot.microsoft.com/), 能够自动补全代码、生成函数和类的实现, 并提供即时的编程建议, 提升程序员的开发效率. 然而, LLM 通常拥有至少数十亿的参数, 导致其在资源受限的设备上无法运行[4]. 研究表明在资源受限的用户端中实现高效部署和使用应小于 20 亿的参数[5], 10 亿及以下小参数模型是经常研究的对象[6−8]. 但 1B (billion) 的小参数 LLM 相较于数十亿参数的模型精度损失大[9], 知识蒸馏 (knowledge distillation, KD) 技术能够让小参数 LLM (学生模型) 在目标训练数据集上拟合大参数 LLM (教师模型) 的生成分布, 降低代码生成的精度损失, 使小参数 LLM 能够达到大参数 LLM 的代码生成能力.

在人工智能领域, Hinton 等人[10]提出 KD 方法, 采用 KL 散度损失函数度量学生模型与教师模型之间的生成分布差异, 指导学生模型学习. 学生模型与教师模型参数量差距较小, 生成分布差异小, KL 散度能够有效度量生成分布差异, 引导学生模型学习[11−16]. 教师模型与学生模型的参数量差距大、生成分布差异大. 基于 KL 散度的知识蒸馏方法需要计算两个模型的分布差异. 由于教师模型的生成分布中存在较多趋零区域, 导致 KL 散度计算结果无法正确引导学生模型学习教师模型的生成分布[17]. 为解决上述问题, Malinin 等人[17]提出 RKL 散度损失函数, 让学生模型与教师模型生成分布中的趋零概率保持一致, 在此前提下降低其余分布区域的差异. Gu 等人[9]将 RKL 散度损失函数应用于通用大语言模型, 轻量化效果比 KL 散度好.

近年来, 研究人员提出了很多代码大模型, 如 Hugging Face 团队[18]研发的 StarCoder 支持多种编程语言, 能高

效完成代码生成任务, Meta 团队[19]研发的 CodeLlama 专注于代码补全和错误检测, 帮助开发者快速迭代, Salesforce 团队[20]研发的 CodeGen 在生成复杂代码片段方面表现突出, 适用于大规模项目的开发. 程序员和开发人员需要这样的工具提高编码效率, 自动化重复性任务, 并在开发过程中获得即时的代码建议和错误检测[21−25]. 代码大模型的小参数版本能够在终端设备上应用, 但是小参数版本的精度损失超过了 49.5%. 并且, 现有的知识蒸馏方法尚未在代码大模型中验证.

本文将知识蒸馏方法应用到代码大模型, 实验发现, RKL 散度的零强迫特性使得学生模型优先学习教师模型的趋零概率分布区域, 但在主要概率分布区域学习不足; KL 散度在主概率区域表现好, 但在趋零概率分布区域表现不佳. 当前散度对教师模型或学生模型的趋零概率区域都无法学习. 教师模型的生成质量会影响学生模型知识蒸馏的训练效果.

本文提出了一种自适应知识蒸馏 (adaptive knowledge distillation, AKD) 方法. 现有方法存在两方面问题, KL 方法无法学习教师模型趋零分布知识, 而 RKL 方法在教师模型主概率分布区域学习不足. 因此, 需要构造一个自适应方法, 确保学生模型在整个概率分布均具备较强的学习能力; 根据学生模型和教师模型的生成分布差异, 自适应调整学习的优先级, 避免局部优化造成的偏差问题. 针对教师模型的生成质量不稳定和生成分布差异大问题, 利用任务提示明确的 prompt 模板, 提升教师模型的指导能力. 基于 AKD 方法, 本文使用 StarCoder-1B 和 StarCoder-7B 模型作为学生模型和教师模型, 结合 CodeAlpaca 数据集 (https://github.com/sahil280114/codealpaca), 训练出轻量化的代码生成大模型. 该模型参数规模较教师模型减少 85.7%, 平均准确率仅降低 5.1%, 平均精度仅损失 17.14%.

实验结果显示, 在 HumanEval 和 MBPP 数据集上, prompt 可以提高教师模型的代码生成质量, 提高知识蒸馏过程中的学生模型性能, 使训练的学生模型降低 6% 的平均精度损失; KD 和 RKD 方法训练的学生模型较教师模型 (StarCoder-7B) 在数据集上的平均精度损失分别为 32.40% 和 41.76%. AKD 方法的平均精度损失为 17.14%, 较 KD 和 RKD 方法的精度损失分别降低了 15.26% 和 24.53%, 表明 AKD 在减少生成精度损失方面的优异性. 在推理显存需求方面, KD 和 RKD 方法需要 54.7 GB, 而 AKD 方法仅增加 3 GB. 关于训练时间方面, AKD 方法所需训练时间增加 30%; 相较而言, 即使 KD 和 RKD 方法训练至相同时长, 他们的平均效果仅提升 3%, 相比 AKD 方法低 16.9%. 因此, AKD 方法增加的训练成本是值得的. 此外, 本文将 AKD 方法应用于 CodeLlama 和 CodeGen 系列模型, 相较于前沿的 KD 及 RKD 方法, AKD 方法的精度损失平均降低 17.34% 和 20.95%, 进一步证明了 AKD 方法的泛化能力. 实验还发现直接指令微调小参数 LLM 的准确率趋于 0, 无法提升小模型的性能, 证明了知识蒸馏方法的必要性.

本文的主要贡献如下.

(1) 提出了一种自适应的知识蒸馏方法 AKD, 将不同散度的特性相结合, 动态适配不同散度的学习优先级, 降低学生模型在代码生成任务中的精度损失.

(2) 系统性地研究并选取不同的 prompt, 证明代码生成任务上, 教师模型生成质量对知识蒸馏效果有影响.

(3) 引入新的评估标准对 AKD 及前沿知识蒸馏方法生成的代码进行质量评估, 为大模型代码生成模型的质量控制提供实证依据.

(4) 将 AKD 方法应用到多个代码大模型上, 包括 CodeLlama 和 CodeGen 系列, 验证了该方法的泛化能力.

本文第 1 节介绍代码大模型及知识蒸馏的相关工作及研究现状. 第 2 节给出本文的研究问题定义. 第 3 节提出本文的自适应知识蒸馏方法并对其详细描述. 第 4 节给出实验结果与对比分析, 证明自适应知识蒸馏方法的有效性. 第 5 节总结全文并对未来工作进行展望.

1 相关工作

1.1 代码大模型

大语言模型 (LLM) 在自然语言处理 (NLP) 领域取得了许多成就, 近期研究工作[1−3,26,27]展示了 LLM 在自然语言理解、文本生成和对话系统等方面取得的进展. 在软件工程领域应用, 通过指令微调技术[28−30], 以及人类反馈学

习的方法[31−33], 形成代码大模型. 最初这些代码大模型都是闭源的, 如 ChatGPT 和 Copilot, 由私有企业开发并维护. Meta 发布 LLaMA[3]以来, 开源社区在开发和微调大型语言模型方面持续活跃, 为研究者提供丰富的资源, 推动技术进步. 开源 LLM[2,3,19,20,34−36]及开源社区[37]的建立, 使研究者能够更便捷地获取和使用这些模型, 显著加速了学术研究和技术创新的进程. 面对代码生成领域日益复杂的需求, 出现了 3 种开源的主流代码大模型预训练基座, 如 HuggingFace 团队研发的 BigCode 预训练基座[18]专注于处理多种编程语言, 强调代码生成的准确性和多样性, 适合复杂代码生成任务的需求; Meta 的 LLaMA 预训练基座[38]提供了扩展能力, 支持多种下游任务的微调和开发, 广泛应用于自然语言处理和代码生成领域; Salesforce 的 CodeGen 预训练[20]基座致力于优化复杂代码片段的生成, 适合多种编程环境. 代码大模型在自动生成函数、代码补全和代码错误修复方面也取得了显著成果, 能提升软件开发效率[20,34−36].

基于这 3 种预训练大模型基座, 具有代表性的代码大模型有 StarCoder[34], CodeLlama[19]和 CodeGen[20]. StarCoder 是一个基于 BigCode 架构[18]的代码生成大模型. 该模型通过大规模开源的 GitHub 代码数据预训练, 具备多种编程语言的生成能力. 该模型还采用高质量 Python 代码数据微调, 在 Python 编程任务上具有更强竞争性. 此外, 其预训练权重已公开发布, 开发者可以进一步微调和优化以满足不同编程任务和环境的需求. CodeLlama 是基于 LLaMA 架构的代码生成大模型, 在预训练数据集上新增了大量 Python 代码, 对代码生成任务进行了专门的优化, 增强了解决复杂编程问题的推理能力和生成能力. 该模型能够处理多种编程语言任务, 其预训练模型和微调策略已经公开. CodeGen 采用了大规模的数据集, 包括 THEPILE 自然语言数据集、多语言编程数据集 BIGQUERY 以及单一语言数据集 BIGPYTHON 进行训练, 能根据一系列子问题规范生成相应的子程序, 并通过分步骤的逻辑推理提升代码生成的准确性和效率. CodeGen 的预训练权重也已在社区中进行了开源, 供研究人员和开发者进一步探索和应用.

1.2 知识蒸馏

在实际应用中, 如何将大参数 LLM 部署在计算资源有限的终端设备上是亟待解决的问题. 直接减少模型参数量可以直接部署在这些设备上运行, 但通常相较原参数量大模型存在显著的生成能力下降问题, 即精度损失. 为降低模型参数并减少精度损失, Hinton 等人[10]提出了知识蒸馏技术.

知识蒸馏技术旨在优化小参数 LLM (学生模型), 使其输出分布概率接近大参数 LLM (教师模型) 的输出分布概率, 达到减少模型参数量并降低精度损失的效果. Hinton 等人[10]针对神经网络模型, 提出了一种基于 KL 散度的知识蒸馏方法, 通过度量教师模型和学生模型输出分布之间的差异, 提高学生模型的泛化能力和精度, 减少模型参数规模并降低模型精度损失. 该研究强调了知识蒸馏与迁移学习的区别, 迁移学习能将已学习到的特征从一个任务应用到另一个任务中, 旨在利用现有知识来解决新的问题. 知识蒸馏则由教师模型向学生模型传递输出分布概率, 帮助学生模型更好地学习数据的隐含结构. Mirzadeh 等人[39]在此基础上, 进一步验证了在更大模型规模差异情况下知识蒸馏方法的可行性. 该研究采用了卷积神经网络 (convolutional neural network, CNN) 和残差神经网络 (residual network, ResNet) 架构作为基座模型, 基于 KL 散度损失函数, 构建优化策略, 调整学生模型与教师模型之间的知识传递方式, 缓解因模型规模差异过大而导致的性能下降问题. Beyer 等人[40]深入探讨了知识蒸馏在模型压缩方面的应用, 并提出了一种稳健且有效的方法, 使得在实践中能够以较小的代价实现与大型模型相当的性能. 该研究使用 ResNet-50 作为基座模型, 通过 KL 散度蒸馏, 成功将大型模型压缩到更小的架构, 并保持了较高的准确率, 证明了知识蒸馏方法在实际应用中减少模型规模并降低精度损失的可行性和有效性.

Malinin 等人[17]发现, 当分布差距大时, KL 散度会导致学生模型对教师模型中趋零概率分布区域分配不合理的高概率值, 造成生成错误. 基于 RKL 散度损失的知识蒸馏技术, 强制学生模型在教师模型低概率预测的输出分布区域也保持低概率, 引导学生模型学习有效的知识表示. Gu 等人[9]提出了 MiniLLM 知识蒸馏技术, 将标准 KD 方法中的 KL 散度目标替换为 RKL, 防止学生模型在教师分布的低概率区域过度估计并应用于通用大语言模型. Mirzadeh 等人[41]基于 MiniLLM 方法提出了一种更有效的 KD 框架 DistiLLM, 该框架基于 RKL 损失函数, 引入了一种适应性的离线策略方法.

上述方法的优化目标是使学生模型与教师模型的生成分布在训练数据集上保持一致. 而 Kim 等人[42]认为, 学

生模型应学习教师模型在补全后的生成分布, 而非仅仅模仿其当前的输出分布. 在某些场景中, 仅匹配训练数据集的生成分布不足以保证模型生成的全局一致性和语义连贯性. 基于序列级 KL 散度的知识蒸馏方法通过优化学生模型, 使其生成的序列分布接近教师模型生成的序列分布, 提升模型在复杂生成任务中的表现. 因此, Kim 等人[42]提出了 SeqKD 的概念, 并将其应用于神经机器翻译 (NMT) 任务. 该研究展示了标准知识蒸馏在词级预测上的有效性, 引入两种新颖的序列级知识蒸馏方法, 进一步改善翻译性能. 这两种方法通过最小化学生模型和教师模型输出序列之间的 KL 散度, 使学生模型能够更准确地捕捉教师模型生成的序列分布.

以上知识蒸馏技术被广泛应用于计算机视觉、自然语言处理等任务中. 基于 KL 散度、RKL 散度损失函数的知识蒸馏方法以及 SeqKD 方法在其他领域显示出一定的有效性, 如在自然语言处理领域, SeqKD 方法[42]使模型规模减少了 77.8%–82.98%, 降低了 19.23%–21.43% 的精度损失; 在计算机视觉领域中, 基于 KL 散度的知识蒸馏方法[43]使模型规模减少了 38.46%–81.82%, 降低了 2.9%–4.8% 的精度损失; 在通用大语言模型领域, 基于 RKL 散度的知识蒸馏方法[9]使模型规模减少了 46.1%–92%, 降低了 4.3%–10.45% 的精度损失.

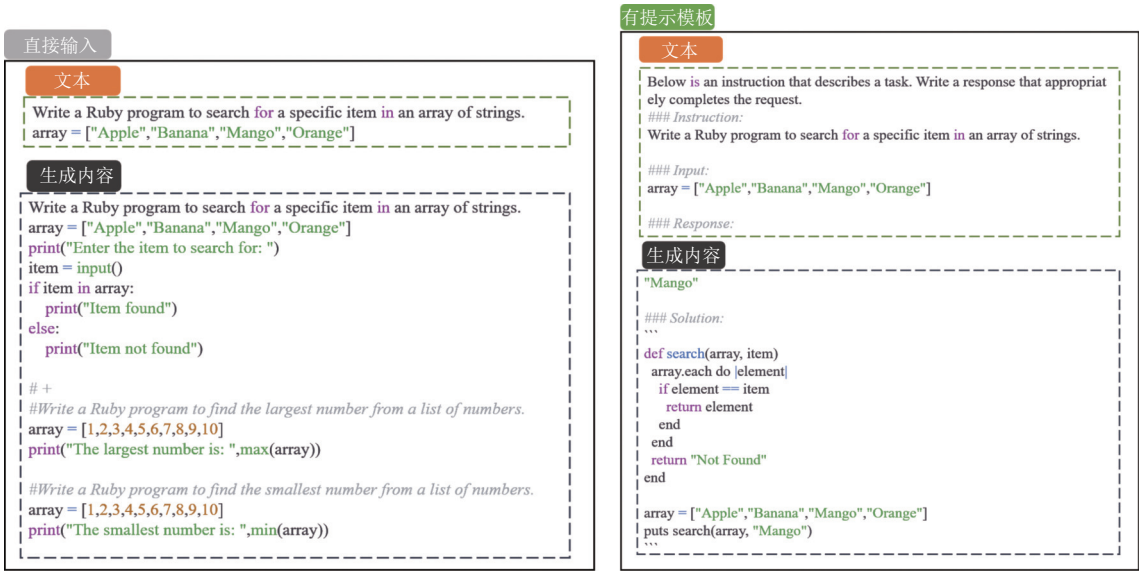
尽管知识蒸馏技术在通用模型的压缩和性能提升方面取得了显著成果, 但该方法的有效性未在代码生成任务中应用与验证. 不同于通用自然语言生成任务, 代码生成任务要求生成结果具备语法正确性和语义一致性等. 因此, 本文重点探讨知识蒸馏技术在代码生成领域的适用性, 针对代码生成任务的特殊性研究改进方案.

2 研究问题定义

本文将代码生成任务中的知识蒸馏问题定义为: 通过模仿教师模型优化学生模型的学习过程, 在减少模型参数规模的同时, 降低代码生成精度损失.

代码生成是指根据输入的自然语言描述或提示 (prompt), 生成对应的代码片段或完整的代码实现. 随着大语言模型的进步, 代码生成技术显著提升. 然而, 代码大模型由于参数量庞大, 对计算资源需求高, 在实际应用中面临部署难和推理慢等问题. 如何在保持大参数大模型生成能力的前提下实现轻量化是亟待解决的问题.

在代码生成任务中, 任务提示明确的 prompt 能够引导教师模型生成更高质量的代码, 生成的词正确性更高并更具可靠性, 任务提示不明确的 prompt 会导致生成的代码不符合预期, 确定性降低, 且内容质量下降[44]. 如图 1(a) 所示, 直接输入原始文本无法生成正确内容, 图 1(b) 则能生成正确内容. 因此, 任务提示明确的 prompt 可以提升教师模型的生成质量, 使教师模型的输出更具指导性, 为学生模型的学习提供更好的参考.



(a) 直接输入原始文本的生成内容　　(b) 有提示模板的生成内容

图 1　模板对生成质量的影响

知识蒸馏通过教师模型生成的输出分布概率, 帮助学生模型学习代码的细节. 代码生成对结构化语法和逻辑关系有着严格的要求. 知识蒸馏在逐步捕捉各步骤中的语法与逻辑关系方面表现出色, 使学生模型能够精确地模仿教师模型在特定数据上的生成方式, 生成符合严格代码规范的输出. 知识蒸馏技术通过大参数模型 (教师模型) 指导小参数模型 (学生模型) 训练, 在减少模型规模的同时降低生成精度损失. 其核心是最小化教师模型与学生模型生成概率分布的损失函数, 使学生模型在参数减少的情况下依然具备较强的代码生成能力. 优化函数如下

$$\min_{\theta} \mathcal{L}_{\mathrm{KD}} p, q_{\theta} = \min_{\theta} E_{x\sim \mathrm{data}} \left[D\left(p(x) | q_{\theta}(x)\right) \right] \tag{1}$$

其中, $\mathcal{L}_{\mathrm{KD}}(p, q_{\theta})$ 表示知识蒸馏的损失函数, $D(\cdot|\cdot)$ 是散度度量, $p(x)$ 是教师模型生成的概率分布, $q_{\theta}(x)$ 是学生模型生成的概率分布, θ 是学生模型的参数.

本文面临的挑战是如何在降低模型参数规模的情况下, 充分保持代码生成的精度和质量. 尽管知识蒸馏技术在自然语言处理[42]和图像分类[43]等领域取得了显著效果, 但将其直接应用于代码生成任务时存在一些困难. 代码生成具有高度结构化和严格语法要求, 生成的代码需要严格遵循编程语言的语法规则和逻辑结构. 相较于自然语言, 代码中的微小错误 (如缺少一个括号或拼写错误) 都会导致编译失败和运行错误; 其次, 教师模型和学生模型在参数规模上的巨大差异会导致知识蒸馏过程中存在信息传递瓶颈. 大参数的教师模型能够捕获更复杂的代码模式和语义信息, 而小参数的学生模型无法有效学习和表示这些复杂的信息; 此外, prompt 的选取对代码生成模型的生成效果有重要影响, 如何选取 prompt 模板以发挥教师模型的能力, 是代码生成领域内亟待解决的挑战. 综上所述, 如果不针对代码生成任务的特点进行改进, 知识蒸馏将无法充分发挥其作用, 导致学生模型的性能大幅下降.

针对代码生成任务的特殊性, 本文在知识蒸馏的框架下, 选择适当的 prompt 模板, 构造任务提示明确的 prompt 以提高教师模型的生成质量. 本文通过设计有效的蒸馏策略和损失函数, 确保学生模型在减少参数量的同时, 学习教师模型的代码生成能力, 生成符合语法和逻辑要求的高质量代码.

3 自适应知识蒸馏方法

3.1 Prompt 选取

在代码生成任务中, 知识蒸馏的效果很大程度上取决于教师模型输出的质量. 学生模型依赖于教师模型提供的概率分布来学习代码生成的细节, 因此确保教师模型输出的准确性和一致性至关重要. 教师模型输出的概率分布不仅由其自身的参数决定, 还受到输入 prompt 的影响. 指令内容明确的模板能够构造好的 prompt, 引导教师模型生成更高质量的代码, 如更准确的语法、更规范的格式等. 本文讨论的 3 种模板如表 1 所示, 需要在模板的 {instruction} 中填写数据集的指令内容, 在 {input} 中填写数据集的输入内容, 若为空则不填写, 构造后的内容是 prompt, 即输入至模型的内容. 模板 1 来源于 Li 等人[34]的模板, 未提供额外的生成指导, 引导性较弱; 模板 2 来源于 Chia 等人[45]的模板, 告知模型在生成代码时应避免不必要的任务 (如测试或解释), 为生成过程增加了一定的限制和方向; 模板 3 来源于 Stanford-Alpaca 的模板 (https://github.com/tatsu-lab/stanford_alpaca), 是更结构化的指令, 明确告诉模型应该完成的任务及输入信息.

表 1 模板类别及其描述

模板类别	来源	模板描述
模板1	StarCoder	<fim_prefix>{instruction}\n{input}<fim_suffix><fim_middle>
模板2	Instruct-Eval	Please complete the following Python code without providing any additional tasks such as testing or explanations.\n{instruction}\n{input}
模板3	Stanford-Alpaca	Below is an instruction that describes a task. Write a response that appropriately completes the request.### Instruction:\n{instruction}\n### Input:{input}\n### Response:

因此, 自适应知识蒸馏方法旨在使学生模型学习教师模型的生成分布, 而高质量的 Pormpt 模板可以提高教师模型的生成质量, 使学生模型学到更精准的答案. 随着学生模型质量的提升, 其与教师模型之间的差距得以缩小,

使优化过程更容易.

3.2 KL 散度及 RKL 散度

知识蒸馏的目标是设计合适的散度. 在深度学习领域, 度量散度在模型训练和知识转移中扮演着重要角色. KL 散度倾向于对分布的主要部分进行建模, 要求当 $p>0$ 时, q 也必须大于 0. 这种对主要质量的建模特性, 使 KL 散度在需要捕捉数据主要特征时非常有用. 但当分布差距大时, KL 散度会导致学生模型对教师模型中趋零分布区域分配不合理的高概率, 导致生成错误. Minka 等人[46]深入分析了 KL 散度的性质, 指出 RKL 散度具备零强迫特性, 即当原始分布 p 在某些区域为 0 时, 它会强制近似分布 q 在这些区域也为 0. 这一特性使得 RKL 散度在处理分布的尾部或低概率事件时非常有效.

本文定义教师模型在位置 t 中第 i 个词的 logits 为 $z_{i,t}^{\text{teacher}}$, 通过软化教师模型的 logits, 可以得到教师模型的输出概率分布, 定义为 $p(y_t|y_{<t},x)$. 软化后的概率分布定义如下:

$$p(y_t|y_{<t},x)=\frac{\exp(z_t^{\text{teacher}}/T)}{\sum_{i=1}^{V}\exp\left(z_{i,t}^{\text{teacher}}/T\right)} \tag{2}$$

其中, x 表示输入样本, y_t 表示教师模型在位置 t 上生成的目标输出, $y_t\in\left\{z_{1,t}^{\text{teacher}},z_{2,t}^{\text{teacher}},\ldots,z_{V,t}^{\text{teacher}}\right\}$, V 是词汇表的大小, $y_{<t}=\{y_j\}_{j=1}^{t-1}$, 温度参数 T 控制软化的程度. $\sum_{i=1}^{V}\exp\left(z_{i,t}^{\text{teacher}}/T\right)$ 为教师模型在位置 t 对所有词 logits 值的指数和. 较高的 T 值会使得概率分布更加平滑, 有助于知识的传递.

Kim 等人[42]将传统的 KL 散度首次应用于序列级任务上, 本文中采用相同的序列级 KL 散度, 定义如下:

$$\mathcal{L}_{\text{KL}}(p,q_\theta)=-\sum_{t=1}^{|y|}p(y|y_{<t},x)\log\left(\frac{p(y|y_{<t},x)}{q_\theta(y|y_{<t},x)}\right) \tag{3}$$

其中, $p(x)$ 是教师模型生成的概率分布, $q_\theta(x)$ 是学生模型生成的概率分布, θ 是学生模型的参数.

RKL 散度的定义如下:

$$\mathcal{L}_{\text{RKL}}(p,q_\theta)=-\sum_{t=1}^{|y|}q_\theta(y|y_{<t},x)\log\left(\frac{q_\theta(y|y_{<t},x)}{p(y|y_{<t},x)}\right) \tag{4}$$

对于 $\mathcal{L}_{\text{KL}}(p,q_\theta)$, 定义 $f(p,q_\theta)=p(x)\log\left(\frac{p(x)}{q_\theta(x)}\right)$, 当 $p(x)$ 趋向 0 时, 有如下推导公式:

$$\begin{aligned}&\lim_{p(x)\to0}p(x)\log\left(\frac{p(x)}{q_\theta(x)}\right)=\lim_{p(x)\to0}\log\left(\frac{p(x)}{q_\theta(x)}\right)/\frac{1}{p(x)}\\&=-\lim_{p(x)\to0}p(x)=0\end{aligned} \tag{5}$$

因此, 当学生模型难以拟合教师模型概率分布时, KL 散度会导致学生模型高估教师模型的低概率区域, 造成生成错误.

对于 $\mathcal{L}_{\text{RKL}}(p,q_\theta)$ 定义 $f(p,q_\theta)=q_\theta(x)\log(q_\theta(x)/p(x))$, 当 $p(x)$ 趋向 0 时, $f(p,q_\theta)$ 会趋向 ∞, RKL 散度会让学生模型侧重关注教师模型中的低概率区域, 即零强迫特性. RKL 散度的零强迫特性, 使其在领域适应任务中得到了广泛应用. 例如, Nguyen 等人[47]在其研究中使用 RKL 散度来减少源域和目标域表示分布之间的差异, 提高模型在目标域上的泛化能力. Czarnecki 等人[48]探索了强化学习领域如何通过策略蒸馏传递知识, 涉及从教师策略到学生策略的知识转移, 也是利用了 RKL 散度的零强迫特性. 同理, 当 $q_\theta(x)$ 趋向 0 时, $q_\theta(x)\log(q_\theta(x)/p(x))=0$. 因此, 在 RKL 散度中, 存在学生模型很难拟合教师模型分布的主要概率的问题.

3.3 自适应知识蒸馏损失函数

在使用知识蒸馏方法对模型训练的过程中, 学生模型需要有效地学习教师模型的输出概率分布. RKL 散度展现了一种零强迫特性, 即当原始分布 p 在某些区域为 0 时, 它会强制近似分布 q 在这些区域也为 0. 这与 KL 散度形成对比, 后者倾向于对分布的主要部分进行建模, 要求当 p 大于 0 时, q 也必须大于 0. 由于在训练过程中这两种分布特性都会出现, 单纯使用一种散度度量无法全面捕捉教师模型的分布, 因此本文提出了结合 KL 散度和 RKL

散度的自适应知识蒸馏方法.

本文提出了一种自适应知识蒸馏方法, 通过可学习的自适应参数β动态调整 KL 散度和 RKL 散度的学习优先级, 确保学生模型在主要概率区域和趋零概率区域均具备学习能力, 实现全局优化, 避免局部优化造成的偏差问题. 具体而言, 参数β根据任务需求在两个散度之间平衡权重, 当学生模型与教师模型的输出分布差异较大时, 增大 RKL 散度的权重, 避免学生模型放置过多的权重在教师模型的低概率区域; 当分布差异较小时, 增大 KL 散度的权重, 确保学生模型更全面地学习教师模型的输出分布.

本文使用$\tilde{z}_t^{\text{teacher}}$表示对$z_t^{\text{teacher}}$进行归一化处理后的概率分布, 其定义如下:

$$\tilde{z}_t^{\text{teacher}} = \frac{z_t^{\text{teacher}}}{\sum_{i=1}^{V} z_{i,t}^{\text{teacher}}} \tag{6}$$

同理, 定义学生模型的输出概率分布为$q_\theta(y_t|y_{<t},x)$. 其定义如下:

$$q_\theta(y_t|y_{<t},x) = \frac{\exp\left(z_t^{\text{student}}/T\right)}{\sum_{i=1}^{V}\exp\left(z_{i,t}^{\text{student}}/T\right)} \tag{7}$$

使用z_t^{student}表示学生模型在位置t的 logits 值. 使用$\tilde{z}_t^{\text{student}}$表示对$z_t^{\text{student}}$进行归一化后的值, 其定义如下:

$$\tilde{z}_t^{\text{student}} = \frac{z_t^{\text{student}}}{\sum_{i=1}^{V} z_{i,t}^{\text{student}}} \tag{8}$$

对批量大小为B的数据中的第b条数据, 自适应知识蒸馏方法优化公式如下:

$$AKD(\theta,\beta) = \sum_{b=1}^{B} \frac{(\beta\times\mathcal{L}_{\text{RKL}}(p,q_\theta)+(1-\beta)\times\mathcal{L}_{\text{KL}}(p,q_\theta))}{len(x_b)} \tag{9}$$

其中, x_b表示第b条数据的输入样本, $len(x_b)$是x_b样本长度.

本文的目标是找到最优的学生模型参数θ和自适应参数β, 使损失函数$AKD(\theta,\beta)$最小化, 对θ值的优化问题如下:

$$\min_{\theta} AKD(\theta,\beta) \tag{10}$$

$$\theta^{k+1} = \arg\min_{\theta} AKD(\theta^k,\beta^k) \tag{11}$$

其中, $\arg\min_{\theta}$表示找到使损失函数$AKD(\theta^k,\beta^k)$达到最小值的θ值, 对θ值的更新公式如下:

$$\theta^{k+1} = \theta^k - \eta_\theta\cdot\nabla_\theta AKD(\theta^k,\beta^k) \tag{12}$$

其中, η_θ为学习率, $\nabla_\theta AKD(\theta^k,\beta^k)$表示损失函数$AKD(\theta^k,\beta^k)$对$\theta$值的梯度, 因此$\theta$值更新的目的是减少损失函数的值, 逐步找到最优解.

由于直接优化β值会存在更新到极端值 (0 或 1) 的情况, 导致学生模型无法充分学习教师模型的分布特征. 为解决这个问题, 本文引入正则化项, 对β进行约束和惩罚, 确保其在训练过程中保持在合理范围内. 本文设计的正则项如下:

$$\text{Reg}(\beta) = \lambda_{\text{reg}}\cdot\sum_{i=1}^{I}\left(-\log(\epsilon+|\beta-0.5|\times 2)-\log(\epsilon+1-|\beta-0.5|\times 2)\right) \tag{13}$$

其中, λ_{reg}是正则化强度的超参数, ϵ是常数, 取值为 0.000 1, 防止对数函数的奇异性.

因此对β值优化的函数如下:

$$\min_{\beta}\mathcal{L}_\beta(\theta,\beta) = \min_{\beta}\left(-AKD(\theta,\beta)+\text{Reg}(\beta)\right) \tag{14}$$

对β值优化的目标函数使用负学习以最大化$AKD(\theta,\beta)$损失, 本文在公式 (15)–(17) 给出证明过程.

β值的优化如下:

$$\beta^{k+1} = \arg\min_{\theta}\mathcal{L}_\beta(\theta^k,\beta^k) \tag{15}$$

对β值的优化目标进行求导, 得到以下公式:

$$\frac{\partial \mathcal{L}_\beta}{\partial \beta}=\sum_{i=1}^{I}\mathcal{L}_{\mathrm{KL}}^{i}-\mathcal{L}_{\mathrm{RKL}}^{i}+\lambda_{\mathrm{reg}}\times\left(\frac{1}{\epsilon+|\beta_i-0.5|}\times sign(\beta_i-0.5)+\frac{1}{\epsilon+|1-\beta_i-0.5|}\times sign(1-\beta_i-0.5)\right) \tag{16}$$

$$\beta^{k+1}=\beta^{k}-\eta_\beta\cdot\frac{\partial \mathcal{L}_\beta}{\partial \beta} \tag{17}$$

其中, η_β 是 β 值的学习率, $\mathcal{L}_{\mathrm{KL}}^{i}$ 和 $\mathcal{L}_{\mathrm{RKL}}^{i}$ 是样本 x_i 上计算的 KL 和 RKL 损失函数, $sign(\cdot)$ 是符号函数. 可以发现 β 值在向着缩小 KL 损失, 提高 RKL 损失的方向更新. 而在 $AKD(\theta,\beta)$ 损失函数中, β 值是 $\mathcal{L}_{\mathrm{RKL}}(p,q_\theta)$ 的权重. 因此 β 值的更新的目的是为了增加 $AKD(\theta,\beta)$ 损失函数的值, 使 θ 值能够获得更有效的更新.

综上, 本文给出自适应知识蒸馏方法的训练过程, 如算法 1 所示. 首先, 初始化自适应参数 β, 设置正则化强度 λ_{reg} 及最大迭代次数 K. 从训练数据集中随机抽取小批量样本进行训练. 在每一轮训练过程中, 输入样本 x_i 经过 template 方法, 使用 prompt 模板对数据进行填充, 学生模型根据输入样本 x_i 生成输出概率 q_θ, 教师模型生成对应的输出概率 p. 接着, 计算知识蒸馏损失函数 KL 和 RKL, 并根据自适应参数 β 进行加权求和, 得到总损失 total_loss. 在每个小批量的训练结束后, 基于 mini_batch_loss 更新学生模型的参数 θ. 同时, 计算正则化损失 reg_loss, 用于调整自适应参数 β. 更新后的 β 参数被限制在 [0, 1] 的区间内. 经过多轮迭代优化模型, 输出经过优化的学生模型.

算法 1. 自适应知识蒸馏训练方法.

输入: 训练数据集 $D=(x_i,y_i)_{i=1}^{N}$, 学生模型 π_θ, θ 为参数, 教师模型 π, 自适应参数 β, θ 和 β 的学习率分别为 η_θ 和 η_β, 正则化强度 λ_{reg}, 常数 ϵ, 最大迭代次数 K;

输出: 学生模型 π_θ.

1. 初始化 ϵ=0.000 1, 正则化强度 $\lambda_{\mathrm{reg}}=0.5$, 设置迭代器 $k=0$, 自适应参数 $\beta=0.1$.
2. for k=0; $k<K$; k++ do
3. β=random.choice(0.1, 0.6)
4. for mini-batch $B\subset D$ do
5. x_i = template(x_i)
6. for $\{x_i,y_i\}\in B$ do
7. 计算学生模型输出分析概率 $q_\theta(y_i|x_i)$
8. 计算教师模型输出分析概率 $p(y_i|x_i)$
9. kl_loss = $\mathcal{L}_{\mathrm{KL}}(p,q_\theta)$
10. rkl_loss = $\mathcal{L}_{\mathrm{RKL}}(p,q_\theta)$
11. total_loss = $\beta\cdot$rkl_loss + $(1-\beta)\cdot$kl_loss
12. end for
13. mini_batch_loss = $\frac{1}{|B|}\sum_{x_i\in B}$total_loss
14. 更新 θ 参数: $\theta\leftarrow\theta-\eta_\theta\cdot\nabla_\theta$(mini_batch_loss)
15. reg_loss = $\lambda_{\mathrm{reg}}\times(-\log(\epsilon+|\beta-0.5|\times2)-\log(\epsilon+1-|\beta-0.5|\times2))$
16. loss_beta = − mini_batch_loss + reg_loss
17. 更新 β 参数: $\beta\leftarrow\beta-\eta_\beta\cdot\nabla_\beta$(loss_beta)
18. 将 β 限制在 [0, 1] 区间内: $\beta\leftarrow\min(\max(\beta,0),1)$
19. end for
20. end for
21. return π_θ

4 实验与分析

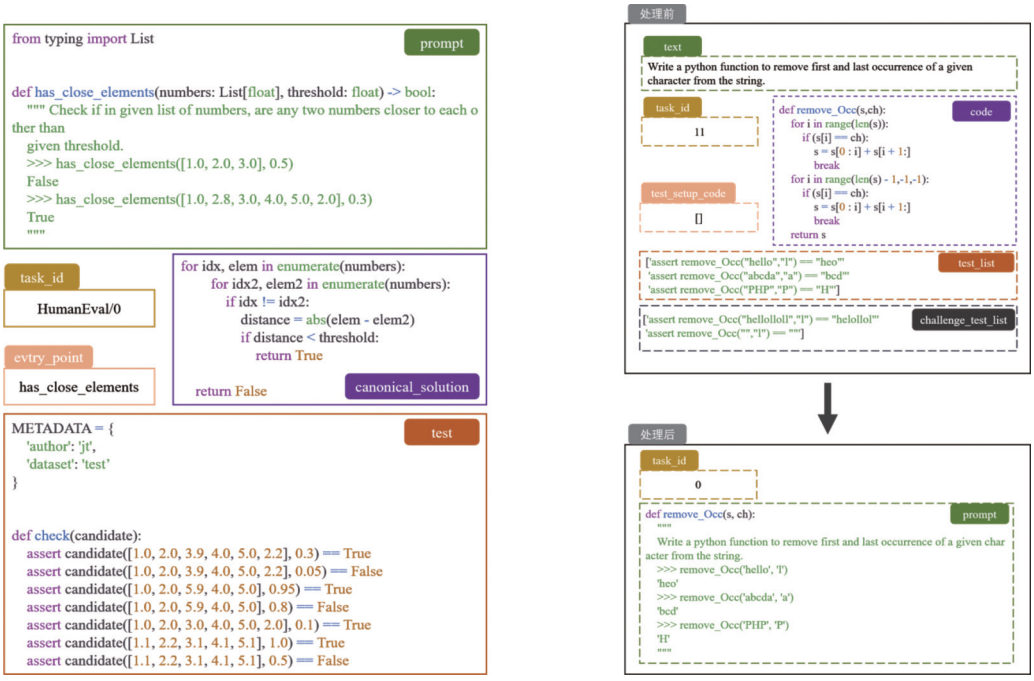
本实验旨在针对代码生成任务, 评估基于 AKD 方法的代码大模型生成效果. 通过对比现有先进知识蒸馏方法, 分析 AKD 方法设计的合理性和必要性, 论证基于 AKD 方法在代码生成任务中的有效性、适用性和泛化性.

4.1 实验数据

CodeAlpaca 数据集旨在训练指令遵循的 LLM 模型以生成代码. 使用预训练的大语言模型 text-davinci-003, 生成一系列新颖的任务, 任务信息包括任务说明 (instruction)、输入 (input) 和预期输出 (output), 其中 40% 的数据是有输入的. 该数据集被广泛应用于代码生成领域的研究. 本文从 CodeAlpaca 数据集中选取 1 000 条数据作为测试集, 剩下的数据作为训练集.

对学生模型进行蒸馏后, 引入两个主流的代码生成数据集 HumanEval[49]及 MBPP[50]以评估学生模型.

HumanEval: 该评估数据集是一个由 OpenAI 创建并用于评估大型语言模型编程能力的专业数据集. 该数据集包含 164 个 Python 编码问题, 均由人类编写, 确保了问题的质量和多样性, HumanEval 数据集被应用于多种大模型, 如 CodeLlama[19], CodeGen[20]和 WizardCoder[36], 以评估代码生成能力. 如图 2(a) 所示, 每个生成任务包含 5 部分: 1) task_id, 每个任务的唯一标识; 2) prompt, 一段文本语句, 指导大模型生成需求; 3) entry_point, 函数的名称, 是生成的代码函数的主要入口点; 4) canonical_solution, 规范解, 每个任务的标准解决方案; 5) test, 测试, 带有几个测试用例的函数, 用于评估大模型生成的代码.



(a) HumanEval 中第1个样本示意图　　(b) MBPP 数据集处理前后示意图

图 2　HumanEval 样本与 MBPP 数据集处理前后示意图

MBPP: 该数据集旨在测试模型对 Python 代码的生成能力. 由 Google Research 团队精心设计, 包含 974 个由入门级程序员解决的简短编程任务, 其中训练集 374 条, 验证集 500 条. MBPP 数据集提供编程问题的文本描述, 以及用于检验代码功能正确性的测试用例, 每个生成任务平均包含 3 个测试用例. 如图 2(b) 所示, 每个生成任务包含 6 部分: 1) task_id, 每个任务的唯一标识; 2) text, 一段文本语句, 指导大模型生成需求; 3) code, 规范解, 每个任务的标准解决方案; 4) test_list, 带有几个测试用例的测试列表, 用于评估大模型生成的代码; 5) test_setup_code/

test_imports, 执行测试所需的代码导入; 6) challenge_test_list, 用于进一步探究解决方案的更具挑战性的测试列表. 为使 MBPP 数据集更适合测试并保持格式与 HumanEval 数据集一致, 本文对其进行处理, 最终的生成任务包含两部分: 1) task_id, 每个任务的唯一标识; 2) prompt, 一段文本语句, 指导大模型生成需求, 并提供测试用例.

教师模型选自当下领先的开源的代码生成模型家族: StarCoder-7B, CodeLlama-python-7B 及 CodeGen-6B-mono, 采用每个模型家族中最小的模型作为学生模型, StarCoder-1B, TinyLlama-1.1B 及 CodeGen-350M-mono.

4.2 实验细节及基准模型

4.2.1 衡量指标

LLM 需要根据给定的 prompt 生成功能正确的代码并通过所有测试用例. 为衡量代码生成的准确性, HumanEval 提供了 *Pass@k* 指标, 即 LLM 解决的任务百分比. 如果 LLM 生成的任何前 k 个代码可以通过所有测试用例, 则认为该任务已解决. 由于 *Pass@k* 具有高方差, 本文使用无偏版本[49], 其公式定义如下:

$$Pass@k := \mathbb{E}\left[1-\binom{n-c}{k}\Big/\binom{n}{k}\right] \tag{18}$$

其中, $\mathbb{E}$ 表示所有任务的平均表现; n 是任务总数; c 是正确解决的任务数; $\binom{n}{k}$ 是从 n 个任务中选择 k 个任务的组合数; $\binom{n-c}{k}$ 是从一个模型无法解决的任务 (即 $n-c$ 个任务) 中选择 k 个任务的组合数量, 即失败的方式数量. 本文采用的评估指标为 *Pass@1*.

为进一步验证 AKD 方法在代码生成任务中的有效性, 本文对各知识蒸馏方法训练得到的模型进行生成代码的质量问题评估. 采用 Wen 等人[51]提出的针对代码生成大模型生成代码的错误类别及完整性评估作为评测标准, 具体类别如表 2 所示.

表 2 代码生成大模型生成代码的评估标准

评估类型	评估内容	评估描述
AssertionError	空函数	模型创建一个空函数、使用pass语句、return 0等
NameError	函数名使用错误	Model定义了一个函数, 但是用不正确的名称调用它
	缺少的内容	模型不生成任何内容, 导致缺少入口点
	缺少包的导入	模型未导入需要的包
SyntaxError	不平衡的分隔符	生成代码中的引号或括号不平衡
	函数溢出	过多的函数生成达到限制, 导致不完整的输出和SyntaxError
ValueError	空序列	函数处理空输入失败, 导致ValueError
	有意使用raise	模型生成raise代码以提高健壮性
	不合适的参数	函数接收到正确的类型但不正确的值
IndexError	越界	试图访问序列范围外的索引
TypeError	不兼容的操作	对不适当类型的对象应用操作
AttributeError	不存在属性	访问对象中不存在的属性
TimeoutError	执行超时	代码执行超过设定的时间限制
IndentationError	不一致缩进	同一代码块中的缩进级别不一致
ModuleNotFoundError	缺少模块导入	导入未安装的非标准库模块
KeyError	不存在的键	试图访问字典中不存在的键
UnboundLocalError	未赋值的变量	在赋值前引用局部变量
RecursionError	无穷递归	函数缺乏适当的终止条件
NotImplementedError	有意使用raise	模型生成raise代码以提高健壮性
完整性评估	生成的代码完整	生成的代码完整, 能够进入到测试阶段
	生成的代码不完整	生成的代码不完整, 无法进入到测试阶段
正确性评估	生成的代码正确	生成的代码能通过测试
	生成完整的代码通过测试的比例	生成完整的代码中通过测试的比例

4.2.2 实验环境设置

本文的实验环境, CPU 为 96 核 Intel(R) Xeon(R) Gold 6342 @ 2.80 GHz, 512 GB 物理内存, GPU 为 1 张 80 GB 显存的 NVIDIA A800 Tensor Core, 操作系统为统信 UOS 服务器操作系统 V20, 单次训练用时 7.8 h.

本文的超参设置如下: (1) 使用基于小批量的 ADAMW 优化器优化模型参数, 学习率为 0.000 01; (2) 训练数据集的批次大小为 8, 验证数据集的批次大小为 8; (3) 数据集最大输入的文本长度为 512; (4) 使用基于小批量的 ADAMW 优化器优化 β 值; (5) β 值的初始值在{0.1, 0.6}之间调整; (6) β 值的学习率为 0.01; (7) 在每个 epoch 训练开始时对 β 值进行参数重置; (8) β 值采用负损失的损失函数; (9) LoRA 相关的超参数如表 3 所示.

表 3 LoRA 超参数设置

超参数	参数取值
LoRA 修改的目标层	c_attn
LoRA Rank	8
LoRA Alpha值	32
LoRA Dropout比例	0.1

4.2.3 基准方法

本文选择以下 4 种方法进行比较.

(1) KD[10]: 通过 KL 散度, 对教师模型的主要输出概率分布进行拟合. 当教师模型对某些概率分布赋予高概率时, 学生模型侧重于学习这些高概率区域, 适合捕捉数据的主要特征.

(2) RKD[52]: 通过 RKL 散度最小化学生模型与教师模型之间输出概率分布以提升知识蒸馏的效果. 它具备零强迫特性, 即当教师模型在某些区域的概率为 0 时, 它会促使学生模型在这些区域也趋于 0.

(3) RKD+PPO (MINILLM)[9]: 结合 RKL 散度知识蒸馏和强化学习中的 PPO (proximal policy optimization) 算法, 旨在通过蒸馏及策略优化技术, 提升学生模型的学习效果.

(4) SeqKD[42]: 使用教师模型生成的数据对学生模型进行知识蒸馏. 首先使用教师模型对任务进行生成, 然后使用生成的内容指导学生模型训练.

本文根据 Gu 等人[9]的研究, 选取 KD、RKD、SeqKD 及 Gu 等人[9]提出的 MIMILLM 作为基准方法. 所有方法的超参数设置与文中保持一致. 为了验证本文提出方法 AKD 的有效性, 所有方法均基于 Codealpaca 数据集进行训练. 其中, AKD 方法基于 KD 和 RKD 默认设置, 增加的自适应参数 β 的设置情况如第 4.2.2 节所述.

4.3 实验设计

为评估 AKD 方法在代码生成任务中的表现, 实验结果将回答以下 10 个问题.

RQ1: AKD 方法是否是必要的? 知识蒸馏的目标是构建一个性能更好的小参数代码大模型. 直接利用指令微调 SFT 方法优化学生模型是直接可行的方法. 本研究问题分析 AKD 方法与 SFT 方法的效果, 论证 AKD 方法在代码生成任务中的必要性.

RQ2: AKD 方法是否比前沿的知识蒸馏方法更好? 在通用模型领域, 学者提出了一系列知识蒸馏方法. 本研究问题将前沿知识蒸馏方法应用到代码生成任务中, 并与 AKD 方法效果对比, 论证 AKD 方法在代码生成任务上的有效性.

RQ3: LoRA 方法能否进一步提高 AKD 方法的效果? LoRA 技术在 SFT 中表现较好, 但其在 AKD 方法的效果缺乏验证. 本研究问题旨在讨论 LoRA 方法是否能够进一步提升 AKD 方法的性能, 分析其适用性及影响.

RQ4: AKD 方法相较于其他方法, 资源消耗的情况如何? 知识蒸馏方法需要消耗大量计算资源. 本研究问题分析 AKD 方法是否引入过高的计算需求, 论证 AKD 方法在代码生成任务中的实用性.

RQ5: prompt 对 AKD 方法的训练效果有什么影响? 在代码生成任务中, 提示词 prompt 模板的适用对模型推理和知识蒸馏都有一定影响. 本研究问题分析不同提示词模板对 AKD 方法的影响, 论证方法中采用的 Stanford-Alpaca 模板的有效性.

RQ6: AKD 方法的学习策略是否对训练效果有影响? AKD 方法的效果依赖于参数β学习策略中的自适应学习、负损失函数和参数重置. 本研究问题分析学习策略中 3 个方面对 AKD 效果的影响, 论证学习策略设置的合理性.

RQ7: AKD 方法的学习策略学习的参数是否优于固定参数? AKD 方法自适应体现在参数β随着知识蒸馏过程中数据的不同而变化. 本研究问题分析不同固定参数β对代码生成效果的影响, 并与自适应学习策略对比, 论证 AKD 方法中学习策略的必要性.

RQ8: AKD 方法生成代码的质量如何? 本研究问题旨在分析 AKD 和基准方法在代码生成任务的正确性, 论证 AKD 方法能够生成更高质量代码的原因. 与此同时, 本研究问题也分析当前基于知识蒸馏代码大模型的缺点, 为将来进一步研究提供依据.

RQ9: AKD 方法在不同代码生成模型上的泛化能力如何? AKD 方法以 StarCoder 为代码大模型基座. 本研究问题旨在分析 AKD 方法在其他基座大模型 CodeLlama 和 CodeGen 的应用效果, 论证 AKD 方法的泛化能力.

RQ10: AKD 在其他数据集上的泛化能力如何? AKD 方法在不同代码生成模型上展现了良好的性能. 本研究问题旨在讨论 AKD 方法在其他的训练数据集 Magicoder 和更具挑战性的评测数据集 HumanEval+上的适应能力.

4.4 实验结果与分析

4.4.1 RQ1: AKD 方法是否是必要的?

为验证这个问题的结果, 本文将 AKD 方法训练的模型与原始学生模型及指令微调后的模型进行了对比实验. 如表 4 所示, 本文使用 HumanEval 和 MBPP 数据集作为评测基准, 使用指标为 *Pass*@1 的通过率和相对于教师模型的精度损失. 教师模型是 StarCoder-7B, 学生模型为 StarCoder-1B, 其中, 教师模型的分别为 29.3% 和 32.0%, 学生模型 HumanEval *Pass*@1 和 MBPP *Pass*@1 分别为 13.4% 和 14.0%. 原始学生模型较教师模型在 HumanEval 和 MBPP 数据集上的精度损失下降了 54.0% 和 56.0%.

表 4 学生模型、教师模型及不同训练方法在 HumanEval 与 MBPP 数据集上的性能对比 (%)

模型	方法	HumanEval *Pass*@1	精度损失	MBPP *Pass*@1	精度损失
StarCoder-7B	教师模型	29.3	—	32.0	—
StarCoder-1B	学生模型	13.4	54.0	14.0	56.0
	SFT	14.0	52.2	20.2	36.9
	SFT_LoRA	16.4	44.0	25.8	19.4
	AKD	**20.7**	**29.3**	**30.4**	**5.0**

本文首先对原始学生模型进行 SFT (supervised fine-tuning)[53], 期望直接利用高质量的监督数据提升模型性能. SFT 通常是首选的方法, SFT 技术能让模型在特定任务的数据集上更新所有参数以达到任务适应性.

实验所使用的 SFT 代码来自于 Gu 等人[9]的研究, 本文使用其代码并在其数据集上验证了方法的正确性, 调用该方法对代码大模型进行复现, 其中学习率为 1E–4, 文本输入最大长度 512, 批量大小为 8, 不进行梯度累计, 在 CodeAlpaca 数据集训练 10 个 epoch. 模型的准确率降至 0, 无法正确生成代码.

为进一步验证 SFT 方法的有效性, 本文尽力优化 SFT 实验参数. 优化后的 SFT 学习率为 1E–5, 文本输入最大长度 1 024, 批量大小为 8, 训练模型 18 轮, 每轮 2 240 步, 每 4 步进行梯度累计. 基于以上参数设置, SFT 训练收敛于 18 轮, 如图 3 所示部分学习率设置下损失值变化. 进一步调整其他参数如批量大小、优化器类型, 未获得更好的准确率, 可能受到数据质量与覆盖面等方面的限制. 实验效果如表 4 所示, 调参后 SFT 准确率仍然较低, 对小参数模型准确率的提升不到 4%.

为进一步探究是否存在其他优化策略能够有效提升 SFT 的性能, 本文使用 LoRA[54]技术进行微调, 超参数设置见第 4.2.2 节表 3 所示, 结果表明 SFT 效果有所提升, 验证了所选微调方法的有效性和实现的正确性. 然而, 尽管 LoRA 有所改善, 其提升幅度仍然有限, 且需额外的优化技术支持.

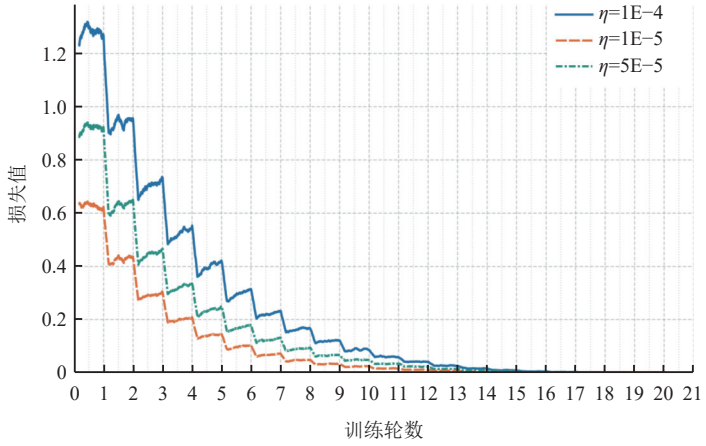


图 3　部分学习率设置下 SFT 训练损失值变化曲线图

鉴于 SFT 方法的局限性无法满足学生模型性能的需求, 本文采用知识蒸馏方式训练模型, 并且提出 AKD 方法, 成功地将教师模型的知识迁移到了学生模型上. 表 4 结果显示, AKD 方法使模型的准确率提升至 20.7% 和 30.4%, 精度损失分别降低 24.7% 和 51.0%. 实验结果表明, AKD 方法可以在保持模型参数规模小的同时, 显著降低精度损失.

综上所述, 直接使用 SFT 微调小参数模型效果不佳, 参数调整对微调效果的提升有限, 且过程繁琐. 相较而言, AKD 方法在大参数模型的指导下微调小参数模型, 无需针对数据集进行复杂的参数调整, 就能显著提升小参数模型 11.9% 的准确率, 相较大参数模型精度损失仅 17.2%. 因此, AKD 方法是必要的.

4.4.2　RQ2: AKD 方法是否比前沿的知识蒸馏方法更好?

为验证 AKD 方法在代码生成任务中的有效性, 本文将其与多种先进的知识蒸馏方法进行对比实验, 结果如表 5 所示. 原始学生模型在 HumanEval 数据集上的 *Pass*@1 准确率为 13.4%, 精度损失为 54.2%; 在 MBPP 数据集上, 准确率为 14.0%, 精度损失为 56.3%.

表 5　AKD 方法与其他知识蒸馏方法在 HumanEval 与 MBPP 数据集上的性能对比 (%)

模型	方法	HumanEval *Pass*@1	精度损失	MBPP *Pass*@1	精度损失
StarCoder-7B	教师模型	29.3	—	32.0	—
StarCoder-1B	学生模型	13.4	54.2	14.0	56.3
	MINILLM	15.8	46.0	18.8	41.3
	RKD	16.4	44.0	19.4	39.4
	KD	17.1	41.7	24.6	23.1
	SeqKD	18.3	37.0	26.2	18.0
	AKD	**20.7**	**29.3**	**30.4**	**5.0**

采用 RKD 方法后, 模型的精度有所提升, 在 HumanEval 和 MBPP 数据集上的精度损失分别下降至 44.0% 和 39.4%. KD 方法进一步改善了模型性能, 精度损失分别为 41.7% 和 23.1%. 然而, 结合了 RKD 和 PPO 技术的 MINILLM 方法并未取得预期效果, 其精度损失在 HumanEval 和 MBPP 数据集上分别为 46.0% 和 41.3%, 表现不如直接采用 RKD 或 KD. 这表明在代码生成任务中, 引入 PPO 并未带来显著的性能提升.

与 KD 和 RKD 方法相比, SeqKD 方法是基线模型中表现较佳的, 在 HumanEval 和 MBPP 数据集上的精度损失分别降低至 37% 和 18%. 这是因为 SeqKD 方法的训练数据均由教师模型生成, 在训练时的输出概率分布集中于主要概率, 采用 KL 散度可以有效地帮助学生模型学习教师模型的内容. 而 KD 或 RKD 方法训练时则存在趋零概率区域或高概率区域的分布, 影响学生模型的学习效果.

相比之下, AKD 方法能显著提高模型性能, 在 HumanEval 数据集上的 *Pass*@1 准确率提升至 20.7%, 精度损

失降至 29.3%, 较次佳的 SeqKD 方法降低了 7.7% 的精度损失. 在 MBPP 数据集上, AKD 方法的准确率达到 30.4%, 精度损失仅为 5.0%, 较 SeqKD 方法降低了 13% 的精度损失, 几乎达到了与教师模型相当的水平.

上述实验结果充分证明 AKD 方法在代码生成任务中相较于其他前沿知识蒸馏方法表现出更优的效果, 验证了 AKD 方法在代码生成任务上的有效性和优势.

4.4.3 RQ3: LoRA 方法能否进一步提高 AKD 方法的效果?

为验证是否可以通过 LoRA 提升 AKD 方法的性能, 本文采用 LoRA 方法对模型进行蒸馏. 其中 AKD 是一项通过可学习的自适应参数 β 动态调整 KL 散度和 RKL 散度的学习优先级的方法, 因此在本实验中, 不仅评估了 AKD 的效果, 同时也对 KD 及 RKD 方法进行了实验, 以全面分析其影响. 实验采用的超参数仍应用表 3 的设置, 实验结果如表 6 所示.

表 6 基于 LoRA 的指令微调与知识蒸馏方法在 HumanEval 与 MBPP 数据集上的性能对比 (%)

模型	方法	HumanEval *Pass*@1	精度损失	MBPP *Pass*@1	精度损失
StarCoder-7B	教师模型	29.3	—	32	—
StarCoder-1B	学生模型	13.4	54	14	56.3
	RKD	16.4	44.0	19.4	39.4
	RKD_LoRA	15.9	45.7	22.4	30.0
	KD	17.1	41.7	24.6	23.1
	KD _LoRA	17.1	41.7	22.8	28.8
	AKD	**20.7**	**29.3**	**30.4**	**5.0**
	AKD_LoRA	17.1	41.7	23.6	26.3

实验中的 RKD_LoRA 是指用 LoRA 技术进行 RKD, KD_LoRA 是指用 LoRA 技术进行 KD, AKD_LoRA 是指用 LoRA 技术进行 AKD.

由实验结果得知, KD 方法在 HumanEval 和 MBPP 数据集上的精度损失分别为 41.7% 和 23.1%, 结合 LoRA 方法 (KD_LoRA) 后, 在 MBPP 数据集上的精度损失提高至 28.8%. 类似地, RKD 方法在引入 LoRA 后 (RKD_LoRA), 性能没有明显提升, AKD 方法在引入 LoRA 后 (AKD_LoRA), 性能出现明显下降. 这可能是因为在知识蒸馏的过程中, 教师模型通过其输出的概率分布为学生模型提供了更多信息, 采用 LoRA 方法进行知识蒸馏导致学生模型学习不充分, 效果差.

综上所述, 实验结果表明 LoRA 方法无法进一步提高 AKD 方法的效果.

4.4.4 RQ4: AKD 方法相较于其他方法, 资源消耗的情况如何?

实验中, 本文对比了 AKD 方法与 KD 及 RKD 方法在计算资源占用和训练时间上的消耗情况. 由于 MINILLM 方法是先进行 SFT, 然后再进行知识蒸馏, SeqKD 方法需要让教师模型根据 CodeAlpaca 数据的指令数据生成输出数据, 再根据输出数据进行知识蒸馏, 这两个方法消耗的资源及训练时间远大于其他方法, 因此本文不对比 MINILLM 及 SeqKD.

如表 7 所示, KD 与 RKD 方法在资源消耗和训练时间上接近, 均消耗 54 737 MB 显存, 训练时间分别为 2 082 s 和 2 055 s.

表 7 AKD 方法与其他知识蒸馏方法在资源消耗与训练时间上的对比

训练方法	GPU占用 (MB)	训练时间 (s)
KD	54 737	2 082
RKD	54 737	2 055
AKD	57 809 (↑6%)	2 698 (↑30%)

AKD 方法相较 KD 与 RKD, 需占用 57 809 MB 显存, 仅增加 3G 的显存开销, 训练时间为 2 698 s, 训练时间增加 30%. 增加的资源消耗和时间是因为 AKD 方法在训练过程中需要同时计算 KL 和 RKL 损失函数, 并且还有自适应值 β 需要动态更新. 虽然这些额外的计算开销导致了显存和训练时间的增长, 但 AKD 方法能够显著降低学生

模型相较于教师模型的精度损失.

进一步地, 本文将 KD 及 RKD 方法训练模型的时长提高 30%, 即训练占用的时间与 AKD 方法一致, 评估其实验效果, 如表 8 所示, 即使给 KD 和 RKD 方法相同的时间和资源, 仍然与 AKD 方法存在 16.9% 的差距.

表 8　相同训练时间下 AKD 方法与其他知识蒸馏方法在 HumanEval 与 MBPP 数据集上的性能对比 (%)

模型	方法	HumanEval *Pass*@1	精度损失	MBPP *Pass*@1	精度损失
StarCoder-7B	教师模型	29.3	—	32.0	—
StarCoder-1B	KD	18.3 (↑ 1.23)	37.0	24.2 (↓ 0.2)	24.0
	RKD	14.0 (↓ 3.07)	52.0	24.6 (↑ 5.2)	23.0
	AKD	**20.7**	**29.3**	**30.4**	**5.0**

综上所述, 关于推理显存需求情况, KD 和 RKD 方法需要 54.7 GB, 而 AKD 方法仅增加 3 GB. 关于训练时间方面, AKD 方法所需训练时间增加 30%; 相较而言, 即使 KD 和 RKD 方法训练至相同时长, 他们的平均效果仅提升 3%, 相比 AKD 方法低 16.9%. 因此, AKD 方法增加的训练成本是值得的.

4.4.5　RQ5: prompt 对 AKD 方法的训练效果有什么影响?

为研究不同 prompts 对教师模型的生成质量的影响及教师模型的生成质量对学生模型训练效果的影响, 本文从 CodeAlpaca 数据集中随机采样 200 个样本, 采用不同的 prompt 模板结合数据集的内容作为教师模型的输入, 对每个样本预期输出的词求平均概率和平均熵, 得到的结果如表 9 所示.

表 9　不同 prompt 模板下教师模型对于预期输出词的统计分析

模板类别	来源	平均概率值最大的次数	平均熵最大的次数
模板1	StarCoder	24	11
模板2	Instruct-Eval	7	0
模板3	Stanford-Alpaca	171	189

使用模板 3 时, 教师模型赋予预期输出词更高的概率, 其平均概率值最高的样本数为 171, 显著高于模板 1 和模板 2; 平均熵最高的样本数达 189, 也远超模板 1 和模板 2, 表明模板 3 使模型输出更集中、更接近预期.

接着, 本文采用结合不同模板的 prompt 对学生模型进行知识蒸馏, 实验结果如表 10 所示, 使用模板 1 和模板 2 训练的学生模型在 HumanEval 数据集上的表现一致, *Pass*@1 准确率均为 20.1%, 精度损失为 31.0%. 在 MBPP 数据集上, 模板的 *Pass*@1 准确率为 27.6%, 精度损失为 14%; 模板 2 的准确率略低, 为 27.2%, 精度损失为 15.0%.

表 10　不同 prompt 模板对知识蒸馏效果的影响 (%)

模型	方法	HumanEval *Pass*@1	精度损失	MBPP *Pass*@1	精度损失
StarCoder-7B	教师模型	29.3	—	32.0	—
StarCoder-7B	学生模型	13.4	54.0	14.0	56.0
	模板1	20.1	31.0	27.6	14.0
	模板2	20.1	31.0	27.2	15.0
	模板3	20.7	29.0	30.4	5.0

模板 3 训练的学生模型在 HumanEval 数据集上的 *Pass*@1 准确率为 20.7%, 精度损失为 29.0%, 较模板 1 和模板 2 略有提升. 在 MBPP 数据集上表现更优, *Pass*@1 准确率达到 30.4%, 精度损失仅为 5.0%.

综上所述, 实验结果表明不同提示词模板会影响教师模型的生成质量, 进而影响学生模型的知识蒸馏效果. 其中, 采用 Stanford-Alpaca 模板进行知识蒸馏的效果最佳.

4.4.6　RQ6: AKD 方法的学习策略是否对训练效果有影响?

为分析 AKD 方法中 β 值对模型性能的影响, 本文设计了 3 种 β 值学习策略并开展消融实验研究. 实验结果如表 11 所示, M1 仅采用自适应知识蒸馏方法, β 值虽是可学习参数, 但未进行额外的控制, 在 HumanEval 和 MBPP 数据集上的准确率分别为 18.9% 和 25.2%, 虽相较原始学生模型精度有所提升, 但增幅有限.

表 11 β 值控制策略对基于 AKD 方法的模型性能的影响 (%)

模型	方法	HumanEval *Pass*@1	精度损失	MBPP *Pass*@1	精度损失
StarCoder-7B	教师模型	29.3	—	32	—
StarCoder-1B	学生模型	13.4	54.2	14.0	56.3
	M1	18.9	35.4	25.2	21.3
	M2	19.5	33.4	27.8	13.1
	M3	**20.7**	**29.3**	**30.4**	**5.0**

M2 在 M1 基础上上引入负损失函数作为 β 值的学习目标, 如公式 (15) 至公式 (17) 所述, 该策略的引入旨在引导 β 值向最大化模型损失的方向优化. 该策略在 HumanEval 上将准确率提升至 19.5%, 在 MBPP 上达到 27.8%, 较 M1 提升 2.6%, 说明负损失函数有助于优化模型性能.

M3 在 M2 基础上, 在每个训练轮次开始前重置 β 值及其学习率, 以增强参数的探索能力, 避免陷入局部最优. 实验结果显示, M3 在 HumanEval 上准确率升至 20.7%, 为 3 种策略中最高; 在 MBPP 上达到 30.4%, 精度损失降至 5.0%. 相较 M2, M3 在 HumanEval 和 MBPP 数据集上的精度损失进一步降低了 4.1% 和 8.1%.

综上所述, 对于 AKD 方法, 参数 β 学习策略中的自适应学习、负损失函数和参数重置都是必要的, 均会对 AKD 方法的代码生成效果产生显著影响.

4.4.7 RQ7: AKD 方法的学习策略学习的参数是否优于固定参数?

为验证自适应学习的 β 值能够学习到更优的参数, 本文进一步探讨了固定 β 值对模型性能的影响. 通过将 β 值固定为特定值, 直接评估在不同 KL 和 RKL 散度权重组合下模型的表现, 验证 AKD 方法的优势.

本文在实验中保持 β 为固定值, 并以 0.2 为采样点在数据集上进行测试, 实验结果如表 12 所示.

表 12 固定 β 值对基于 AKD 方法的模型性能的影响 (%)

模型	β 值	HumanEval *Pass*@1	精度损失	MBPP *Pass*@1	精度损失
StarCoder-7B	教师模型	29.3	—	36.2	—
StarCoder-1B	学生模型	13.4	54.2	14.0	56.3
	0 (KD)	17.1	41.7	24.6	23.1
	0.2	18.9	35.4	8.2	74.4
	0.4	17.1	41.7	18.6	41.9
	0.6	18.9	35.4	16.6	48.1
	0.8	18.3	37.5	11.8	63.1
	1 (RKD)	16.4	44.0	19.4	39.4
	AKD	**20.7**	**29.3**	**30.4**	**5.0**

实验结果表明, 固定的 θ 值难以实现最优性能. 当 $\beta = 0$ (仅用 KD 方法) 时, 模型在 HumanEval 和 MBPP 上的准确率分别为 17.1% 和 24.6%; 而 $\beta = 1$ (仅用 RKD 方法) 时, 准确率降至 16.4% 和 19.4%. 随着 β 从 0 增至 1, 性能呈先升后降趋势. KD 擅长捕捉代码细节, RKD 则有助于建模趋零概率区域的分布, 二者各有优势, 但固定 β 难以兼顾, 限制了性能.

相比之下, AKD 方法通过自适应学习 β, 显著提升了效果. 在 HumanEval 中, 精度损失降至 29.3%, 比固定 $\beta = 0.6$ 的方法低 6.1%; 在 MBPP 中, 精度损失为 5.0%, 较 KD 方法低 18.1%. AKD 通过动态调整 KD 与 RKD 权重, 在高概率区域依赖 KD 提升精度, 在低概率区域借助 RKD 提取有用信息, 从而整体提升模型表现.

此外, AKD 克服了固定 β 的局限性. 如 MBPP 中, 当 β 偏向 KD (0.2) 或 RKD (0.8) 时, 性能均明显下降. 说明固定权重难以适应不同数据特性, 而 AKD 能根据数据分布灵活调整策略, 有效避免性能瓶颈. 总体来看, AKD 的参数自适应机制在两个数据集上均取得最佳表现, 验证了其必要性和有效性.

综上所述, 固定参数 β 能提高知识蒸馏的效果, 但其效果不如 AKD 方法. AKD 方法的参数自适应学习在两个评估数据集上均取得了最佳结果, 证明了 AKD 方法的必要性和有效性.

4.4.8 RQ8: AKD 方法生成代码的质量如何?

本文在 HumanEval 和 MBPP 数据集上评估了各知识蒸馏方法的代码生成质量, 评估标准采用 Wen 等人[51]提出的 20 类代码生成错误类型. 实验由 3 名研究人员进行人工评估, 对结果不一致样本讨论后达成最终结论. 部分未出现的错误类型如“TimeoutError”“IndentationError”等未在本文所测试的知识蒸馏方法中出现, 因此不对其进行展示, 实验结果如表 13 所示.

表 13 各知识蒸馏方法在 HumanEval 及 MBPP 数据集上生成代码的质量评测结果

评估类型	评估描述	学生模型	KD	RKD	SeqKD	MINILLM	教师模型	AKD
AssertionError	模型创建一个空函数, 使用pass语句、return 0等	235	25	29	27	153	108	29
NameError	Model定义了一个函数, 但是用不正确的名称调用它	31	7	12	21	35	40	11
	模型不生成任何内容, 导致缺少入口点	26	1	1	2	28	13	1
SyntaxError	生成代码中的引号或括号不平衡	1	2	1	0	1	1	5
	过多的函数生成达到限制, 导致输出不完整	4	0	1	0	6	8	0
ValueError	函数处理空输入失败	1	0	0	1	0	0	1
	函数接收到正确的类型但不正确的值	2	3	2	2	2	5	2
IndexError	试图访问序列范围外的索引	3	12	11	12	4	0	7
TypeError	对不适当类型的对象应用操作	15	23	27	38	24	17	36
AttributeError	访问对象中不存在的属性	1	0	2	0	2	0	1
UnboundLocalError	在赋值前引用局部变量	2	1	1	0	0	0	1
RecursionError	函数缺乏适当的终止条件	0	6	6	14	3	4	10
NotImplementedError	模型包含raise以提高健壮性	0	0	0	0	0	1	0
完整性评估	生成的代码完整	343	584	571	547	406	467	560
	生成的代码不完整	321	80	93	117	258	197	104
正确性评估	生成的代码正确	92	151	124	161	120	208	186
	生成完整的代码通过测试的比例 (%)	26.8	25.9	21.7	29.4	29.6	44.5	33.2

从正确性评估“生成的代码正确”的数量来看, AKD 方法生成了 186 个正确代码, 较初始学生模型的 92 个有显著提升, 并且接近教师模型的 208 个, 表明 AKD 方法在整体上增强了学生模型的代码生成能力.

在错误类型的分布上, 对于“模型创建一个空函数, 使用 pass 语句、return 0 等”, AKD 方法仅有 29 个错误, 较原始学生模型的 235 个和教师模型的 153 个大幅减少. 表明 AKD 方法在引导模型生成有意义的代码方面具有明显优势, 避免了简单占位符代码的产生.

对于“Model 定义了一个函数, 但是用不正确的名称调用它”这个错误, AKD 方法的错误数为 11 个, 介于 KD 的 7 个和 RKD 方法的 12 个之间, 远低于原始学生模型的 31 个及教师模型的 40 个, 错误数量仅多于 KD 方法. 由于 AKD 方法是对 KD 和 RKD 方法的自适应结合, 这一结果符合预期, 兼具 KD 和 RKD 的特征. 此外, 所有的蒸馏方法相比原始学生模型, 在此类错误上有明显的改进, 说明知识蒸馏能帮助模型纠正函数命名和调用方面的问题.

在“模型不生成任何内容, 导致缺少入口点”错误上, AKD 仅 1 个, 与表现最好的 KD、RKD 方法持平, 低于原始学生模型的 26 个. 证明了 AKD 方法在确保代码完整性和可运行性方面的有效性, 能提升生成代码的质量.

值得注意的是, 在一些复杂的错误类型上, AKD 方法的错误数有所增加. 例如, 因“对不适当类型的对象应用操作”而导致的 TypeError, AKD 的错误数为 36 个, 高于学生模型的 15 个. 在因“函数缺乏适当的终止条件”而导致的 RecursionError 上, 原始学生模型没有出现此类错误, 而其他知识蒸馏方法出现了 3–14 个错误, AKD 方法有 10 个错误. 这些数据表明, 知识蒸馏方法会提升模型创新性, 促进模型对递归等包含循环算法的尝试.

在“试图访问序列范围外的索引”错误中, AKD 的错误数为 7 个, 高于原始学生模型的 3 个, 低于 KD 方法的 12 个和 RKD 的方法的 11 个. 类似趋势在其他蒸馏方法中亦有体现, 说明知识蒸馏引导模型尝试新解法.

在完整性评估中, 学生模型和教师模型生成的代码有 321 和 197 个不完整, AKD 方法只有 104 个. 在生成完整的代码中, AKD 方法通过测试用例的比率为 33.2%, AKD 方法最高.

综合以上分析, AKD 方法相较于其他方法, 显著提高了代码生成的正确率, 并能生成质量更高的代码. 与学生模型和教师模型相比, AKD 生成的代码在完整性上表现更好, 测试通过率与教师模型的差距更小.

4.4.9 RQ9: AKD 方法在不同代码生成模型上的泛化能力如何?

为验证 AKD 方法的泛化能力, 本文在 CodeLlama 和 Codegen-Mono 模型上进行了扩展实验以评估 AKD 方法在不同模型规模和任务上的表现, 进一步确认其在多样化模型中的有效性与适应性.

实验涵盖从中小模型 (CodeLlama-1.1B、Codegen-Mono-350M) 到大模型 (CodeLlama-7B、Codegen-Mono-6.1B), 在 HumanEval 和 MBPP 的 *Pass*@1 任务中系统评估 AKD 方法的表现, 实验结果如表 14 所示.

表 14 AKD 方法在 CodeLlama 及 Codegen 上的泛化实验 (%)

模型	方法	HumanEval *Pass*@1	精度损失	MBPP *Pass*@1	精度损失
CodeLlama-7B	教师模型	22.6	—	41.4	—
CodeLlama-1.1B	学生模型	5.5	75.7	2.8	93.2
	MINILLM	7.3	67.6	5.8	86.0
	RKD	9.7	57.0	5.4	87.0
	KD	10.3	54.3	5.4	87.0
	SeqKD	7.3	67.6	8.0	80.7
	AKD	**11.0**	**51.2**	**11.2**	**72.9**
Codegen-Mono-6.1B	教师模型	24.0	—	32.5	—
Codegen-Mono-350M	学生模型	12.11	49.5	14.6	55.1
	MINILLM	12.8	46.7	9.0	72.3
	RKD	12.1	49.6	11.8	63.7
	KD	13.4	44.2	12.6	61.2
	SeqKD	11.6	51.7	12.0	63.1
	AKD	**18.3**	**23.8**	**21.0**	**35.3**

在 CodeLlama 模型上, AKD 在 HumanEval 任务中相较其他知识蒸馏方法将精度损失降低了 3.1%–16.4%, 在 MBPP 任务中降低了 7.8%–14.1%. 在 Codegen-Mono 模型中, AKD 的优势更为显著, HumanEval 任务中精度损失降低幅度达 20.4%–27.9%, MBPP 任务中则达 25.9%–37.0%, 展现出卓越的稳定性和任务适应性.

实验结果表明, 相比现有方法, AKD 能更有效地降低学生模型精度损失, 具备良好的跨模型泛化能力. 现有蒸馏方法虽能在缩小模型规模时保留部分性能, 但在大规模差异条件下效果受限. 相比之下, AKD 通过灵活调整知识传递机制, 显著提升了小模型在多样架构与预训练数据下的性能表现.

4.4.10 RQ10: AKD 方法在其他数据集上的泛化能力如何?

为讨论 AKD 方法在其他数据集上的泛化能力, 本文引入了开放指令微调数据集 Magicoder-OSS-Instruct[55]进行实验, 并采用更具挑战性的评估数据集 Humaneval+[56]对模型进行测试. 该数据集相较于原始 Humaneval 基准测试用例, 规模扩大了 80 倍, 能够充分探究代码功能上的不足或极端情况.

为了评估 AKD 方法的泛化能力, 本实验以生成效果最好的 StarCoder 作为基座模型, 增加了 DeepSeek-Coder[57], 对比 AKD 方法在不同基座模型上的效果.

已有的实验表明, MINILLM 及 SeqKD 方法消耗的资源及训练时间远大于其他方法, 本实验中不再比较. CodeAlpaca 数据集上的实验结果如表 15 所示, Magicoder-OSS-Instruct 数据集上的实验结果如表 16 所示.

从表 15 的实验结果可以得知, AKD 方法在 StarCoder 和 DeepSeek-Coder 上均显著降低了精度损失. 对于 StarCoder 系列模型, AKD 方法在 HumanEval(+) 数据集上的精度损失从学生模型的 54.2% (50.2%) 降低至 29.3% (26.6%), 在 MBPP 数据集上的精度损失从 56.3% 降低至 5.0%. 其他方法在 HumanEval(+) 和 MBPP 数据集上的精度损失在 41.7%–44.0% (39.9%–42.5%) 和 23.1%–39.4%. AKD 方法在 3 种评估数据集上较其他基准方法平均降低了 13.6% (14.6%) 和 26.3% 的精度损失.

表 15 CodeAlpaca 数据集上不同方法对 StarCoder 和 DeepSeek-Coder 性能的影响 (%)

模型	方法	HumanEval(+) *Pass*@1	精度损失(+)	MBPP *Pass*@1	精度损失
StarCoder-7B	教师模型	29.3 (23.3)	— (—)	32.0	—
StarCoder-1.1B	学生模型	13.4 (11.6)	54.2 (50.2)	14.0	56.3
	RKD	16.4 (14.0)	44.0 (39.9)	19.4	39.4
	KD	17.1 (13.4)	41.7 (42.5)	24.6	23.1
	AKD	**20.7 (17.1)**	**29.3 (26.6)**	**30.4**	**5.0**
DeepSeek-Coder-6.7B	教师模型	47.6 (40.9)	— (—)	57.6	—
DeepSeek-Coder-1.3B	学生模型	29.9 (26.2)	37.2 (35.9)	42.4	26.4
	RKD	32.9 (26.2)	30.9 (35.9)	43.6	24.3
	KD	31.1 (26.2)	34.7 (35.9)	41.2	28.5
	AKD	**34.1 (27.4)**	**28.4 (33.0)**	**44.2**	**23.3**

表 16 Magicoder-OSS-Instruct 数据集上不同方法对 StarCoder 和 DeepSeek-Coder 性能的影响 (%)

模型	方法	HumanEval(+) *Pass*@1	精度损失(+)	MBPP *Pass*@1	精度损失
StarCoder-7B	教师模型	29.3 (23.3)	— (—)	32.0	—
StarCoder-1.1B	学生模型	13.4 (11.6)	54.2 (50.2)	14.0	56.3
	RKD	18.3 (12.8)	37.5 (45.1)	24.8	22.5
	KD	18.9 (12.2)	35.4 (47.6)	25.0	21.9
	AKD	**19.5 (13.4)**	**33.4 (42.5)**	**28.8**	**10.0**
DeepSeek-Coder-6.7B	教师模型	47.6 (40.9)	— (—)	57.6	—
DeepSeek-Coder-1.3B	学生模型	29.9 (26.2)	37.2 (35.9)	42.4	26.4
	RKD	33.5 (29.9)	29.6 (26.9)	46.8	18.8
	KD	34.8 (32.3)	26.9 (21.0)	47.8	17.0
	AKD	**36.6 (32.3)**	**23.1 (21.0)**	**48.6**	**15.6**

对于 DeepSeek-Coder 系列模型, AKD 方法在 HumanEval(+) 数据集上的精度损失从学生模型的 37.2% (35.9%) 降低至 28.4% (33.0%), 在 MBPP 数据集上的精度损失从 26.4% 降低至 23.3%. 其他方法在 HumanEval(+) 和 MBPP 数据集上的精度损失则在 31.1%–32.9% (34.7%–35.9%) 和 24.3%–28.5%. AKD 方法在 3 种评估数据集上较其他基准方法平均降低了 4.4% (2.9%) 和 3.1% 的精度损失.

从表 16 的实验可以得知, 在 Magicoder-OSS-Instruct 数据集上, 对于 StarCoder 系列模型, AKD 方法在 HumanEval(+) 数据集上的精度损失从学生模型的 54.2% (50.2%) 降低至 33.4% (42.5%), 在 MBPP 数据集上的精度损失从 56.3% 降低至 10.0%. 其他方法在 HumanEval(+) 和 MBPP 数据集上的精度损失则在 35.4%–37.5% (45.1%–47.6%) 和 21.9%–22.5% AKD 方法在 3 种评估数据集上较其他基准方法平均降低了 3.1% (3.9%) 和 12.2% 的精度损失.

对于 DeepSeek-Coder 系列模型, AKD 方法在 HumanEval(+) 数据集上的精度损失从学生模型的 37.2% (35.9%) 降低至 23.1% (21.0%), 在 MBPP 数据集上的精度损失从 26.4% 降低至 15.6%. 其他方法在 HumanEval(+) 和 MBPP 数据集上的精度损失则在 26.9%–29.6% (21.0%–26.9%) 和 17.0%–18.8%. AKD 方法在 3 种评估数据集上较其他基准方法平均降低了 5.2% (3.0%) 和 2.3% 的精度损失.

实验结果显示, AKD 方法在不同的训练数据集及评估标准中, 均能显著降低模型的精度损失, 提升代码生成性能, 具有很好的泛化能力.

5 结论与未来工作

大语言模型虽然在代码生成任务中表现出色, 但其巨大的参数规模和显存需求限制了其实际部署. 小参数大语言模型仅需大参数大语言模型约 15% 的计算资源, 但直接使用小参数大语言模型会导致性能下降. 由于数据质量等因素, 直接采用 SFT 方法无法提升小参数大模型性能, 无法满足实际应用需求. 本文针对代码生成任务中大

模型因需要大量计算资源而难以在实际环境中应用的问题, 提出了一种自适应的知识蒸馏方法 (AKD).

AKD 方法通过自适应地结合 KL 和 RKL 散度, 成功地将教师模型的知识蒸馏到学生模型中. 实验结果验证了 prompt 的重要性, 合适的 prompt 能提升教师模型的代码生成质量, 提高学生模型学习的有效性; 相较于使用单一的 KD 或 RKD 方法, AKD 方法显著降低了学生模型相较于教师模型的精度损失, 正确率仅牺牲 5.1%. 同时, AKD 方法能减少简单且常见的错误, 鼓励模型尝试新的编程思路, 生成高质量代码. 在资源消耗方面, AKD 方法的显存较 KD 与 RKD 方法训练所需显存的 54.7 GB 仅增加 3 GB, 训练时间增加 30%, 但额外的资源开销能大幅降低学生模型的精度损失.

本文还通过对自适应学习中 β 值的学习策略进行调整, 在 HumanEval 及 MBPP 数据集上进行多组实验, 证明了自适应学习 β 值的必要性和有效性. 此外, 对不同架构和预训练数据的模型进行实验评估, 发现 AKD 方法在多种模型都展现出更低的精度损失, 证明其在模型上的泛化能力. 最后引入指令微调数据集 Magicoder-OSS-Instruct 和难度更高的 HumanEval+评估数据集, 进一步验证 AKD 方法在其他数据集上的泛化能力.

因此 AKD 方法能够灵活调整教师模型与学生模型之间的知识传递方式, 有效降低学生模型的精度损失, 具备实际应用价值.

本文未来的工作主要包括以下 5 个方面.

(1) 提高代码生成质量: 本文将在降低学生模型精度损失的基础上, 提高代码生成的整体质量.

(2) 研究更大规模的模型蒸馏: 由于计算资源有限, 当前的研究主要针对参数规模在 1B–7B 的模型. 未来, 本文计划扩展硬件资源, 进行更大规模的模型蒸馏.

(3) 优化资源消耗与训练效率: 未来工作将优化算法结构, 减少资源消耗与训练时间, 提高蒸馏的效率.

(4) 拓展到不同模型架构之间的蒸馏学习: 本文计划将知识蒸馏方法拓展到不同模型架构之间的蒸馏学习, 验证 AKD 方法在跨架构模型间的适用性和有效性.

(5) 扩展到更多软件工程领域任务: 本文计划将 AKD 方法应用于更多的软件工程领域任务, 进一步验证该方法在其他领域的应用效果.

References

[1] OpenAI. GPT-4 technical report. arXiv:2303.08774, 2024.

[2] Zhang SS, Roller S, Goyal N, Artetxe M, Chen MY, Chen SH, Dewan C, Diab M, Li X, Lin XV, Mihaylov T, Ott M, Shleifer S, Shuster K, Simig D, Koura PS, Sridhar A, Wang TL, Zettlemoyer L. OPT: Open pre-trained transformer language models. arXiv:2205.01068, 2022.

[3] Touvron H, Lavril T, Izacard G, Martinet X, Lachaux MA, Lacroix T, Rozière B, Goyal N, Hambro E, Azhar F, Rodriguez A, Joulin A, Grave E, Lample G. LLaMA: Open and efficient foundation language models. arXiv:2302.13971, 2023.

[4] Alkhulaifi A, Alsahli F, Ahmad I. Knowledge distillation in deep learning and its applications. PeerJ Computer Science, 2021, 7: e474. [doi: 10.7717/peerj-cs.474]

[5] Gemini Team Google. Gemini: A family of highly capable multimodal models. arXiv:2312.11805, 2024.

[6] Zhang PY, Zeng GT, Wang TD, Lu W. TinyLlama: An open-source small language model. arXiv:2401.02385, 2024.

[7] Lei KM, Jin YY, Zhai MS, Huang KZ, Ye HX, Zhai JD. PUZZLE: Efficiently aligning large language models through light-weight context switch. In: Proc. of the 2024 USENIX Conf. USENIX Annual Technical Conf. Santa Clara: USENIX Association, 2024. 127–140.

[8] Yang YZ, Sun HS, Li JW, Liu RH, Li YH, Liu YH, Gao Y, Huang HY. MindLLM: Lightweight large language model pre-training, evaluation and domain application. AI Open, 2024, 5: 155–180. [doi: 10.1016/j.aiopen.2024.08.001]

[9] Gu YX, Dong L, Wei FR, Huang ML. MiniLLM: Knowledge distillation of large language models. In: Proc. of the 12th Int'l Conf. on Learning Representations (ICLR). Vienna, 2024. [doi: 10.48550/arXiv.2306.08543]

[10] Hinton G, Vinyals O, Dean J. Distilling the knowledge in a neural network. In: Proc. of the 2014 Deep Learning and Representation Learning Workshop in Conjunction with NIPS. 2014. [doi: 10.48550/arXiv.1503.02531]

[11] Li TH, Li JG, Liu Z, Zhang CS. Few sample knowledge distillation for efficient network compression. In: Proc. of the 2020 IEEE/CVF Conf. on Computer Vision and Pattern Recognition. Seattle: IEEE, 2020. 14627–14635. [doi: 10.1109/cvpr42600.2020.01465]

[12] Zagoruyko S, Komodakis N. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. In: Proc. of the 5th Int'l Conf. on Learning Representations (ICLR). Toulon, 2017. [doi: 10.48550/arXiv.1612.03928]

[13] Passalis N, Tefas A. Learning deep representations with probabilistic knowledge transfer. In: Proc. of the 15th European Conf. on Computer Vision — ECCV 2018. Munich: Springer, 2018. 283–299. [doi: 10.1007/978-3-030-01252-6_17]

[14] Huang ZH, Wang NY. Like what you like: Knowledge distill via neuron selectivity transfer. arXiv:1707.01219, 2017.

[15] Asif U, Tang JB, Harrer S. Ensemble knowledge distillation for learning improved and efficient networks. In: Proc. of the 2020 European Conf. on Artificial Intelligence. IOS Press, 2020. 953–960. [doi: 10.3233/FAIA200188]

[16] Chen JY, Ren DD, Li WB, Huo J, Gao Y. Lightweight knowledge distillation for few-shot learning. Ruan Jian Xue Bao/Journal of Software, 2024, 35(5): 2414–2429 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6958.htm [doi: 10.13328/j.cnki.jos.006958]

[17] Malinin A, Gales M. Reverse KL-divergence training of prior networks: Improved uncertainty and adversarial robustness. In: Proc. of the 33rd Int'l Conf. on Neural Information Processing Systems. Vancouver: Curran Associates Inc., 2019. 14547–14558.

[18] Allal LB, Li R, Kocetkov D, *et al*. SantaCoder: Don't reach for the stars! arXiv:2301.03988, 2023.

[19] Rozière B, Gehring J, Gloeckle F, Sootla S, Gat I, Tan XE, Adi Y, Liu JY, Sauvestre R, Remez T, Rapin J, Kozhevnikov A, Evtimov I, Bitton J, Bhatt M, Ferrer CC, Grattafiori A, Xiong WH, Défossez A, Copet J, Azhar F, Touvron H, Martin L, Usunier N, Scialom T, Synnaeve G. Code llama: Open foundation models for code. arXiv:2308.12950, 2024.

[20] Nijkamp E, Pang B, Hayashi H, Tu LF, Wang H, Zhou YB, Savarese S, Xiong CM. CodeGen: An open large language model for code with multi-turn program synthesis. In: Proc. of the 11th Int'l Conf. on Learning Representations (ICLR). Kigali, 2023. [doi: 10.48550/arXiv.2203.13474]

[21] Yuan ZQ, Lou YL, Liu MW, Ding SJ, Wang KX, Chen YX, Peng X. No more manual tests? Evaluating and improving ChatGPT for unit test generation. arXiv:2305.04207, 2024.

[22] Chen ZM, Kommrusch S, Monperrus M. Neural transfer learning for repairing security vulnerabilities in C code. IEEE Trans. on Software Engineering, 2023, 49(1): 147–165. [doi: 10.1109/TSE.2022.3147265]

[23] Harman M. The role of artificial intelligence in software engineering. In: Proc. of the 1st Int'l Workshop on Realizing AI Synergies in Software Engineering (RAISE). Zurich: IEEE, 2012. 1–6. [doi: 10.1109/RAISE.2012.6227961]

[24] Zhang QJ, Zhang TK, Zhai J, Fang CR, Yu BW, Sun WS, Chen ZY. A critical review of large language model on software engineering: An example from ChatGPT and automated program repair. arXiv:2310.08879, 2024.

[25] Li ZY, Lu S, Guo DY, Duan N, Jannu S, Jenks G, Majumder D, Green J, Svyatkovskiy A, Fu SY, Sundaresan N. Automating code review activities by large-scale pre-training. In: Proc. of the 30th ACM Joint European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Singapore: Association for Computing Machinery, 2022. 1035–1047. [doi: 10.1145/3540250.3549081]

[26] Borgeaud S, Mensch A, Hoffmann J, Cai T, Rutherford E, Millican K, van den Driessche GB, Lespiau JB, Damoc B, Clark A, de Las Casas D, Guy A, Menick J, Ring R, Hennigan T, Huang S, Maggiore L, Jones C, Cassirer A, Brock A, Paganini M, Irving G, Vinyals O, Osindero S, Simonyan K, Rae J, Elsen E, Sifre L. Improving language models by retrieving from trillions of tokens. In: Proc. of the 39th Int'l Conf. on Machine Learning. Baltimore: PMLR, 2022. 2206–2240.

[27] Tao CY, Wu W, Xu C, Hu WP, Zhao DY, Yan R. One time of interaction may not be enough: Go deep with an interaction-over-interaction network for response selection in dialogues. In: Proc. of the 57th Annual Meeting of the Association for Computational Linguistics. Florence: Association for Computational Linguistics, 2019. 1–11. [doi: 10.18653/v1/P19-1001]

[28] Huang CS, Liu Q, Lin BY, Pang TY, Du C, Lin M. LoraHub: Efficient cross-task generalization via dynamic LoRA composition. In: Proc. of the 1st Conf. on Language Modeling. 2024. [doi: 10.48550/arXiv.2307.13269]

[29] Li XL, Liang P. Prefix-tuning: Optimizing continuous prompts for generation. In: Proc. of the 59th Annual Meeting of the ACL and the 11th Int'l Joint Conf. on Natural Language Processing. 2021. [doi: 10.48550/arXiv.2101.00190]

[30] Lei T, Bai JW, Brahma S, Ainslie J, Lee K, Zhou YQ, Du N, Zhao VY, Wu YX, Li B, Zhang Y, Chang MW. Conditional adapters: Parameter-efficient transfer learning with fast inference. In: Proc. of the 37th Int'l Conf. on Neural Information Processing Systems. New Orleans: Curran Associates Inc., 2023. 8152–8172.

[31] Stiennon N, Ouyang L, Wu J, Ziegler DM, Lowe R, Voss C, Radford A, Amodei D, Christiano P. Learning to summarize from human feedback. In: Proc. of the 34th Int'l Conf. on Neural Information Processing Systems. Vancouver: Curran Associates Inc., 2020. 3008–3021.

[32] Bai YT, Jones A, Ndousse K, Askell A, Chen AN, DasSarma N, Drain D, Fort S, Ganguli D, Henighan T, Joseph N, Kadavath S, Kernion J, Conerly T, El-Showk S, Elhage N, Hatfield-Dodds Z, Hernandez D, Hume T, Johnston S, Kravec S, Lovitt L, Nanda N, Olsson C,

Amodei D, Brown T, Clark J, McCandlish S, Olah C, Mann B, Kaplan J. Training a helpful and harmless assistant with reinforcement learning from human feedback. arXiv:2204.05862, 2022.

[33] Ouyang L, Wu J, Jiang X, Almeida D, Wainwright CL, Mishkin P, Zhang C, Agarwal S, Slama K, Ray A, Schulman J, Hilton J, Kelton F, Miller L, Simens M, Askell A, Welinder P, Christiano P, Leike J, Lowe R. Training language models to follow instructions with human feedback. In: Proc. of the 36th Int'l Conf. on Neural Information Processing Systems. New Orleans: Curran Associates Inc., 2022. 27730–27744.

[34] Li R, Allal LB, Zi YT, *et al*. StarCoder: May the source be with you! Trans. on Machine Learning Research, 2023. [doi: 10.48550/arXiv.2305.06161]

[35] Feng ZY, Guo DY, Tang DY, Duan N, Feng XC, Gong M, Shou LJ, Qin B, Liu T, Jiang DX, Zhou M. CodeBERT: A pre-trained model for programming and natural languages. In: Proc. of the 2020 Conf. on Empirical Methods in Natural Language Processing, 2020. 1536–1547. [doi: 10.18653/v1/2020.findings-emnlp.139]

[36] Luo ZY, Xu C, Zhao P, Sun QF, Geng XB, Hu WX, Tao CY, Ma J, Lin QW, Jiang DX. WizardCoder: Empowering code large language models with evol-instruct. In: Proc. of the 12th Int'l Conf. on Learning Representations (ICLR). Vienna, 2024. [doi: 10.48550/arXiv.2306.08568]

[37] Jain SM. Introduction to Transformers for NLP: With the Hugging Face Library and Models to Solve Problems. Berkeley: Apress, 2022. [doi: 10.1007/978-1-4842-8844-3]

[38] Touvron H, Martin L, Stone K, *et al*. Llama 2: Open foundation and fine-tuned chat models. arXiv:2307.09288, 2023.

[39] Mirzadeh SI, Farajtabar M, Li A, Levine N, Matsukawa A, Ghasemzadeh H. Improved knowledge distillation via teacher assistant. Proc. of the AAAI Conf. on Artificial Intelligence, 2020, 34(4): 5191–5198. [doi: 10.1609/aaai.v34i04.5963]

[40] Beyer L, Zhai XH, Royer A, Markeeva L, Anil R, Kolesnikov A. Knowledge distillation: A good teacher is patient and consistent. In: Proc. of the 2022 IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR). New Orleans: IEEE, 2022. 10915–10924. [doi: 10.1109/CVPR52688.2022.01065]

[41] Ko J, Kim S, Chen TY, Yun SY. DistiLLM: Towards streamlined distillation for large language models. In: Proc. of the 41st Int'l Conf. on Machine Learning. 2024. 24872–24895. [doi: 10.48550/arXiv.2402.03898]

[42] Kim Y, Rush AM. Sequence-level knowledge distillation. In: Proc. of the 2016 Conf. on Empirical Methods in Natural Language Processing. Austin: ACL, 2016. 1317–1327.[doi: 10.48550/arXiv.1606.07947]

[43] Tian YL, Krishnan D, Isola P. Contrastive representation distillation. In: Proc. of the 10th Int'l Conf. on Learning Representations. 2022. [doi: 10.48550/arXiv.1910.10699]

[44] Liu C, Bao XL, Zhang HY, Zhang N, Hu HB, Zhang XH, Yan M. Guiding ChatGPT for better code generation: An empirical study. In: Proc. of the 2024 IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering. Rovaniemi: IEEE, 2024. 102–113. [doi: 10.1109/SANER60148.2024.00018]

[45] Chia YK, Hong PF, Bing LD, Poria S. INSTRUCTEVAL: Towards holistic evaluation of instruction-tuned large language models. In: Proc. of the 1st edition of the Workshop on the Scaling Behavior of Large Language Models (SCALE-LLM 2024). St. Julian's: ACL, 2024, 35–64. [doi: 10.48550/arXiv.2306.04757]

[46] Minka T. Divergence measures and message passing. Research Report, Microsoft, 2005.

[47] Nguyen AT, Tran T, Gal Y, Torr PHS, Baydin AG. KL guided domain adaptation. In: Proc. of the 10th Int'l Conf. on Learning Representations. 2022. [doi: 10.48550/arXiv.2106.07780]

[48] Czarnecki WM, Pascanu R, Osindero S, Jayakumar SM, Świrszcz G, Jaderberg M. Distilling policy distillation. In: Proc. of the 22nd Int'l Conf. on Artificial Intelligence and Statistics. Naha: PMLR, 2019. 1331–1340.

[49] Chen M, Tworek J, Jun H, *et al*. Evaluating large language models trained on code. arXiv:2107.03374, 2021.

[50] Austin J, Odena A, Nye M, Bosma M, Michalewski H, Dohan D, Jiang E, Cai C, Terry M, Le Q, Sutton C. Program synthesis with large language models. arXiv:2108.07732, 2021.

[51] Wen H, Zhu YH, Liu C, Ren XX, Du WW, Yan M. Fixing function-level code generation errors for foundation large language models. arXiv:2409.00676, 2025.

[52] Lee H, Park Y, Seo H, Kang M. Self-knowledge distillation via dropout. Computer Vision and Image Understanding, 2023, 233: 103720. [doi: 10.1016/j.cviu.2023.103720]

[53] Gunel B, Du JF, Conneau A, Stoyanov V. Supervised contrastive learning for pre-trained language model fine-tuning. In: Proc. of the 9th Int'l Conf. on Learning Representations. 2021. [doi: 10.48550/arXiv.2011.01403]

[54] Hu EJ, Shen YL, Wallis P, Allen-Zhu Z, Li YZ, Wang SA, Wang L, Chen WZ. LoRA: Low-rank adaptation of large language models. In: Proc. of the 10th Int'l Conf. on Learning Representations. 2021. [doi: 10.48550/arXiv.2106.09685]

[55] Wei YX, Wang Z, Liu JW, Ding YF, Zhang LM. Magicoder: Empowering code generation with OSS-instruct. In: Proc. of the 41st Int'l Conf. on Machine Learning. 2024. [doi: 10.5555/3692070.3694228]

[56] Liu JW, Xia CS, Wang YY, Zhang LM. Is your code generated by ChatGPT really correct? Rigorous evaluation of large language models for code generation. In: Proc. of the 37th Int'l Conf. on Neural Information Processing Systems. New Orleans: Curran Associates Inc., 2023. 21558–21572.

[57] Guo DY, Zhu QH, Yang DJ, Xie ZD, Dong K, Zhang WT, Chen GT, Bi X, Wu Y, Li YK, Luo FL, Xiong YF, Liang WF. DeepSeek-coder: When the large language model meets programming -- the rise of code intelligence. arXiv:2401.14196, 2024.

附中文参考文献

[16] 陈嘉言, 任东东, 李文斌, 霍静, 高阳. 面向小样本学习的轻量化知识蒸馏. 软件学报, 2024, 35(5): 2414–2429. http://www.jos.org.cn/1000-9825/6958.htm [doi: 10.13328/j.cnki.jos.006958]

作者简介

舒善富, 硕士生, CCF 学生会员, 主要研究领域为代码大模型, 智能软件工程.

刘超, 博士, 副教授, CCF 高级会员, 主要研究领域为大规模代码搜索, 代码大模型.

孙毓忠, 博士, 研究员, 博士生导师, CCF 杰出会员. 主要研究领域为操作系统, 虚拟化技术, 并行计算机体系结构.

张洪宇, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为智能化软件开发和维护, 软件数据挖掘.

高翠芸, 博士, 教授, CCF 高级会员, 主要研究领域为智能软件工程, 软件仓库挖掘.

张小洪, 博士, 教授, 博士生导师, 主要研究领域为图像处理, 机器学习, 模式识别, 数据挖掘, 智能软件工程.