arXiv:2602.22240v1 [cs.PL] 24 Feb 2026

从提示到性能：评估大语言模型生成基于任务的并行代码

Linus Bantel[0009−0009−0247−4748]、Moritz Strack、Alexander Strack[0000−0002−9939−9044] 和 Dirk Pflüger[0000−0002−4360−0212]

斯图加特大学并行与分布式系统研究所，德国斯图加特
st166660@stud.uni-stuttgart.de {linus.bantel, alexander.strack, dirk.pflueger}@ipvs.uni-stuttgart.de

摘要。大语言模型（LLM）在代码生成方面展现出强大的能力，但其生成高效并行程序的能力尚未得到充分研究。本文探讨大语言模型如何根据三种输入提示生成基于任务的并行代码：自然语言问题描述、顺序参考实现和并行伪代码。我们聚焦于三种编程框架：OpenMP 任务并行、C++ 标准并行以及异步多任务运行时 HPX。每种框架在任务执行方面都提供了不同程度的抽象和控制。我们评估大语言模型生成的解决方案的正确性和可扩展性。结果揭示了大语言模型在面对不同问题复杂度和框架时的优势与不足。最后，我们探讨这些发现对未来高性能计算和科学计算领域的大语言模型辅助开发意味着什么。

关键词： 大语言模型 · 代码生成 · 异步多任务系统 · OpenMP · C++ 标准并行 · HPX

1 引言

大语言模型的快速发展通过实现自动化代码生成，改变了软件工程。然而，它们生成正确且高性能的并行程序的能力在很大程度上仍未被探索。并行代码生成需要对数据依赖、工作负载分解和同步进行推理。近期的实证研究表明，当前模型在并行工作负载上的表现明显差于串行工作负载，常常生成不可扩展或低效的实现 [1]。

异步任务并行编程提供了一个有用的框架来审视这些局限性，因为它相比传统的分叉-汇合方法提出了更困难的并行化问题。我们重点关注 OpenMP 任务并行、C++ 标准并行以及异步多任务运行时 HPX，它们涵盖了广泛的抽象层次和执行模型。我们还研究提示输入模态对模型性能的影响，因为先前的工作表明，零样本并行化在很大程度上仍然不可靠，并且容易

出现并发错误 [2]。具体而言，我们将不同的输入提示建立在自然语言、伪代码和顺序实现之上。

此外，我们评估不同类别的大语言模型如何生成异步任务型代码，并分析提示复杂度和抽象如何影响正确性与性能。本文其余部分安排如下。第2节综述高性能计算领域代码生成能力的近期研究。第3节介绍评估方法。第4节呈现结果，第5节给出结论。

2 相关工作

大语言模型在自动代码生成领域的应用已迅速从通用编程辅助发展到面向特定领域的高性能计算（HPC）代码。近年来，Chat-GPT [3], DeepSeek [4], 以及 LLaMA-2 [5] 等开源替代方案所代表的基础模型取得了显著进展，展现出强大的能力，能够使用多种编程语言生成语法正确、语义有意义的代码 [6–9]。这些能力引发了人们对利用大语言模型提升高性能计算应用开发者生产力的日益浓厚的兴趣；在此类应用中，并行编程仍然复杂且容易出错。

近期基于机器学习的方法尝试弥合并行性检测与代码合成之间的差距。AutoParLLM [10] 代表了早期的一项尝试，将基于图神经网络的并行性检测与大语言模型引导的代码生成相结合，既凸显了混合方法的潜力，也揭示了正确性和泛化方面的挑战。然而，近期并行代码生成领域最显著的进展来自基于大语言模型的方法。对 OpenAI Codex 的早期评估表明，大语言模型能够根据自然语言提示词或部分代码直接生成使用 OpenMP、OpenACC、CUDA 和 MPI 构造的高性能计算内核，但成功程度因训练数据的可用性而异 [11]。后续研究扩展了这些评估，对比了用于高性能计算内核生成的专有模型与开源模型，结果表明，尽管两类模型都能生成有效的并行代码，但其性能和正确性在不同编程模型与硬件目标之间存在显著差异 [12]。两项研究都指出了一些持续存在的挑战，包括缺少同步、数据作用域不正确以及性能可移植性问题。HPC-Coder [13]研究了基于大语言模型的并行程序建模，报告称生成正确的 OpenMP 编译指示和 MPI 调用的效果喜忧参半，凸显了合成语义正确的并行代码之难。Godoy 等人 [14] 将早期评估扩展至对大语言模型生成的并行内核进行评估，涵盖 C++、Fortran、Python 和 Julia，结果表明，大语言模型能够辅助自动并行化，但仍需专家验证以确保正确性和性能。

现有研究聚焦于针对不同编程语言、GPU 内核和分叉-汇合并行化的代码生成，而异步任务型代码生成仍鲜少得到探索。我们的工作填补了这一研究

空白：我们使用多个框架和多种大语言模型评估基于任务的代码生成，不仅考察编译和正确性，也考察并行扩展。尽管并行扩展至关重要，先前的研究却未考察强扩展和弱扩展。

3 方法

我们使用一组基准测试问题来评估大语言模型生成基于任务的并行代码的能力。这些问题在算法结构、依赖复杂度以及对计算与内存带宽限制的敏感度方面各不相同。所选基准测试涵盖易并行工作负载、分治算法、常规计算内核，以及带有全局同步屏障的迭代方法：

1. **随机近似** π，代表同步需求极少的独立任务执行。2. **归并排序**，展现递归任务创建和显式依赖约束。3. **矩阵乘法**，一种具有高算术强度和结构化数据访问的稠密线性代数内核。4. **共轭梯度法**，一种涉及稀疏计算、归约和严格迭代顺序的迭代求解器。

这些基准测试共同考验基于任务的并行化的不同方面，包括任务粒度、依赖推断和同步位置。对于每个基准测试，我们根据三种具体程度逐步提高的提示模态生成并行实现。这些模态用于评估输入结构如何影响模型推断并行性的能力：

1. **问题描述**，仅提供函数签名、高层自然语言规范以及目标并行框架。2. **顺序代码**，提供待并行化的完整顺序 C++ 实现。3. **并行伪代码**，用结构化的并行算法大纲来补充该描述。

这一递进过程反映了大语言模型的常见使用场景，从算法设计辅助到对现有代码的自动重构。每个问题–提示组合均在三个大语言模型上进行评估，以便进行比较分析：

1. **ChatGPT-5**2.
Qwen-Coder 3 (30B) [1]3.
Gemini-3

[1] https://huggingface.co/Qwen/Qwen3-Coder-30B-A3B-Instruct

我们的模型选择涵盖不同的参数规模、训练目标和可用性，既包括开源系统，也包括专有系统。

我们使用三种代表不同抽象层次和执行模型的并行编程框架，评估大语言模型生成的基于任务的并行代码：

1. **OpenMP 任务并行** 是一种广泛使用的、基于指令的共享内存并行框架。其显式的任务创建和同步指令，使其成为评估大语言模型推理依赖关系和任务粒度能力的自然基准。2. **C++ 标准并行性** 提供基于库的高级并行抽象，例如并行算法和执行策略。借此，我们可以评估大语言模型是否能够利用声明式并行性。3. **HPX** 是一种完全异步、基于任务的运行时，利用期值和数据流语义。我们纳入 HPX，以评估大语言模型在细粒度、依赖驱动型并行方面的表现。

这些框架共同涵盖了从基于指令到由运行时驱动的并行模式，使我们能够比较分析大语言模型如何适应不同的基于任务的编程模型。对于每种基准问题、提示模态、大语言模型和并行化框架的组合，我们都会生成多个代码片段。我们从三个方面评估生成的实现。正确性通过针对特定问题的验证和参考输出进行评估。复杂度依据生成程序的代码行数、注释密度和控制流模式进行量化。通过改变可用并行资源并分析加速比表现来评估可扩展性。

该方法将问题复杂度、提示模态和模型选择对大语言模型生成的基于任务的并行代码质量的影响进行了隔离。

3.1 评估指标

我们采用 $pass@k$ 来分析生成代码的正确性，采用 scc 评估复杂度，并采用 $PCGQS$ 对生成的并行代码进行基准测试。

正确性 $pass@k$ 指标广泛用于评估大语言模型及其生成的代码 [15]。它量化了在 k 个生成样本中至少有一个样本正确的概率。$pass@k$ 的计算公式如公式（1）所示：

$$\text{pass@k} = 1 - \frac{\binom{n-c}{k}}{\binom{n}{k}} \tag{1}$$

其中， n 表示生成样本的总数， c 表示正确样本的数量。为了评估生成的代码的质量，我们引入了一组等级，用于描述修正代码错误的难度。我们不明确按照错误类型进行划分，即编译错误、运行时错误或一般算法错误。不过，通常编译错误比

运行时错误或算法错误更容易修正。通过按照成功使用代码所需的修正工作量进行分类，该框架能够更全面地评估大语言模型生成的程序。该等级体系包括以下级别：

$$C(c) = \begin{cases} 1 \\ \text{，无需修复——完全正确，无需修改 0。75，简单——只需进行少量调整即可确保正确性 0。5，中等——} \\ \\ \text{需要适度调试和代码修改 0。25，困难——需要进行大量干预才能修复代码，0，。不可行——无法切实地修正代码} \end{cases} \tag{2}$$

复杂度除了代码的可执行性和性能之外，我们还评估生成代码示例的内在属性。我们使用工具scc [2] 来量化有效代码行数以及注释行所占的比例。此外，scc 还提供一种复杂度指标，用于近似衡量圈复杂度 [16]，可作为代码预期维护工作量的指标。

扩展性 为了评估大语言模型生成高效并行代码的能力，我们定义了并行代码生成质量分数（*PCGQS*）。该指标结合了功能正确性与经验扩展行为。对于生成的实现 c，PCGQS 定义如下：

$$\mathrm{PCGQS}(c) = \frac{1}{2}C(c) + \frac{1}{2}S(c), \tag{3}$$

其中， $C(c)$ 表示公式（2）中定义的功能正确性，而 $S(c)$ 则反映并行可扩展性。

根据公式（4）和公式（5），分别通过强扩展和弱扩展实验评估扩展行为。T_1 是单核运行时间， T_p 是使用 p 个处理单元时的运行时间， $\mathcal{P}$ 是评估的核心数量集合。我们按照公式（6）将强扩展和弱扩展合并为一个分数。

$$S_{\text{strong}}(c) = \frac{1}{|\mathcal{P}|} \sum_{p \in \mathcal{P}} \frac{T_1}{p \cdot T_p} \tag{4}$$

$$S_{\text{weak}}(c) = \frac{1}{|\mathcal{P}|} \sum_{p \in \mathcal{P}} \frac{T_1}{T_p} \tag{5}$$

$$S(c) = \frac{1}{2}S_{\text{strong}}(c) + \frac{1}{2}S_{\text{weak}}(c) \tag{6}$$

[2] https://github.com/boyter/scc

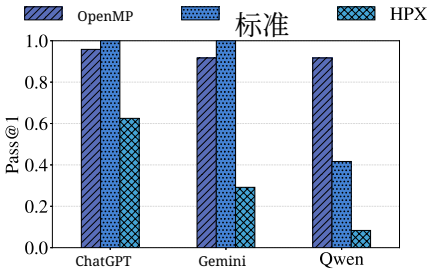
4 结果

为确保实验可复现，所有实验均在固定的测试环境中进行。实验环境配备了一台双路 AMD EPYC 7742 处理器。扩展性实验中，我们评估了最多使用 128 个线程时的代码表现。我们将线程绑定到 CPU 核心，以消除多线程可能引入的潜在隐性影响。根据初步的任务扩展分析，所有实验的任务数均固定为 128。关于关键软件规格，我们使用 GCC 13.3.0、OpenMP 4.5 和 HPX 1.11.0。我们首先评估代码正确性，接着分析复杂度，最后考察扩展行为。

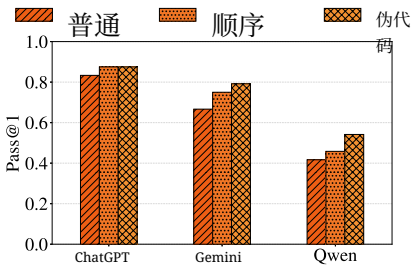
4.1 正确性

我们首先分析这三个大语言模型开箱即用的性能。对于每个大语言模型，我们计算其在所有其余实验组中的 pass@1 指标。因此，图 1 初步展示了模型性能的概况，虽是初步结果，却颇具意义。即使在这些初步结果中，模型之间也已显现出显著差异。在生成正确代码方面，ChatGPT 的开箱即用性能最强。Google Gemini 排名第二，而开源模型 Qwen-Coder 则明显落后。如图 1a 所示，OpenMP 在所有模型上都表现稳健。C++ 标准并行化在 ChatGPT 和 Gemini 上取得了近乎完美的结果，但在 Qwen-Coder 上表现很差。相比之下，HPX 的 pass@1 起始率较低，且在所有模型上都迅速下降。图 1b 显示，不同提示词形式总体上也呈现出类似的模式。性能差异并不能归因于提示类型，且在统计上并不显著。包含并行伪代码的提示词仅带来微乎其微的提升。考察图 1c 中的各个基准问题，可以发现几个值得注意的现象。尽管直觉上， π 的并行近似似乎比并行归并排序更简单，但 ChatGPT 和 Gemini 为归并排序生成的代码都更可靠。这很可能是因为归并排序是一种经典的编程练习，在计算机科学导论课程中十分常见，因此有大量现成的实现。最后，共轭梯度（CG）基准测试的难度相当高，其 pass@1 指标明显低于整体表现较好的矩阵乘法基准测试。

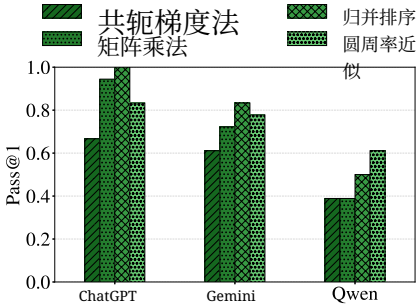
在图2中，我们分析了允许进行有限人工修正时所获得的性能提升。为获得正确代码所需的修改被归入第3.1节介绍的五个修复级别。作为基线，图2a报告了未进行任何人工干预时的 pass@1 指标。该图与图1c完全相同。如图2b所示，仅允许进行简单修复，就已经使可运行代码的比例显著提高。少量修正足以解决所有模型和基准测试问题中相当一部分失败案例。当允许进行中等程度修复时，



(a) 按框架分组。



(b) 按提示类型分组。

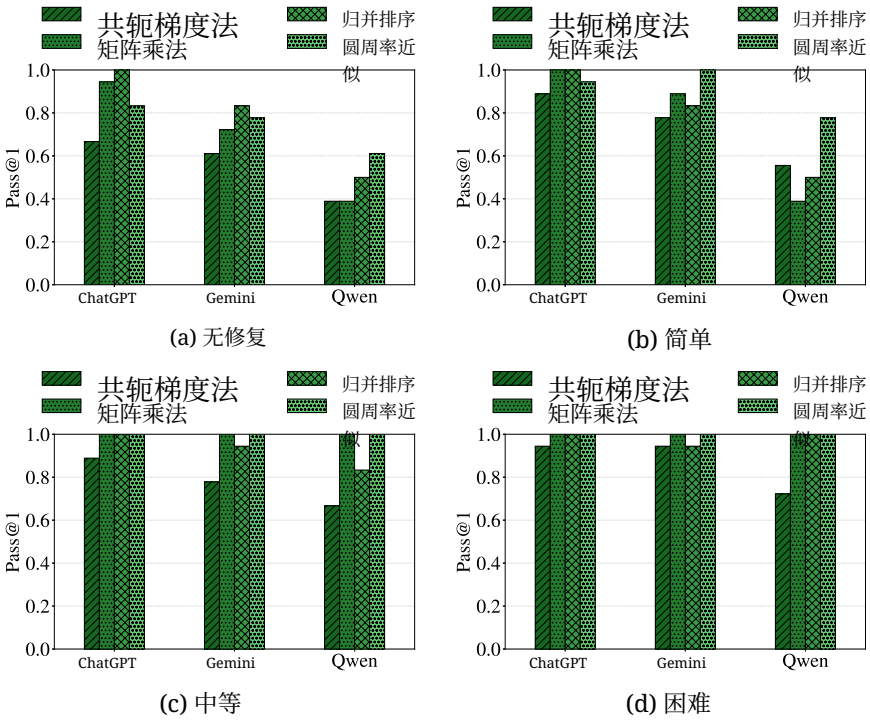


(c) 按问题分组。

图 1：各大语言模型未经任何修正时的 Pass@1 指标。

对于多个基准测试，所有模型的 pass@1 指标均达到100%。值得注意的是，在这一设置下，Qwen-Coder 在矩阵乘法和对 π的近似这两项任务上都取得了完美结果，超过了 Gemini。相比之下，归并排序在未进行修正时的 pass@1 比率最高，但在中等程度修复级别下，其表现落后于矩阵乘法和对 pi 的近似。在进行困难修复后，几乎所有模型—基准测试组合的 pass@1 比率都接近100%。唯一显著的例外是 CG 基准测试：对于 Qwen-Coder，其 pass@1 值仍明显较低。下面，我们将介绍所应用的具体代码修改，并说明将其归入相应修复类别的依据。

在错误类型方面，最常见的头文件相关错误包括缺失头文件、过时头文件或使用不存在的头文件。我们将其归类为简单修复。对于 HPX 和 C++ 标准并行，lambda 表达式经常出现捕获子句错误。这类问题需要中等程度的修复。生成的 HPX-Code 中还有一类反复出现的错误，即命名空间过时或不正确，例如使用 hpx::parallel::for_loop，而不是 hpx::experimental::for_loop。在真正的编码错误中，HPX 中期值处理不当尤为常见，通常是由于 .get() 调用位置不正确或缺失所致。更严重的错误源于在并行区域之间错误地共享数组，这往往会导致段错误或悄无声息的



(a) 无修复 (b) 简单

(c) 中等 (d) 困难

图 2：修复难度等级对应的 Pass@1 指标，即程序正确且能够运行

数据损坏。我们还观察到，在奇数核心数上执行生成的代码时会失败，具体来说是 24、48 和 96 个核心。如果代码在任何扩展性实验中失败，且所需修复不可行，我们就将相应输出归类为不可执行。

4.2 复杂度

表1展示了不同大语言模型、编程框架和提示策略对生成代码的结构与复杂度的影响对比。所有类别的代码行数均保持不变。代码文档的详尽程度存在显著差异：Gemini 生成的代码包含更多注释，而 ChatGPT 生成的注释最少。此外，提示词中的抽象程度与生成代码的复杂度直接相关。包含伪代码的提示词能提供清晰的结构，促进代码生成。相比之下，提供顺序实现的提示词会促使模型严格遵循该结构，并在其周围添加并行结构。这种方法会增加生成代码的结构复杂度。

类别	#行数	#代码	#注释	复杂度
LLM				
ChatGPT	67.25	51.27	4.66	8.65
Gemini	75.97	51.07	12.80	9.06
Qwen	71.89	50.78	8.63	9.66
框架				
OpenMP	70.11	51.14	8.16	9.06
标准 C++	70.93	50.26	8.71	10.26
HPX	74.12	51.75	9.25	7.99
提示级别				
普通	73.46	50.43	10.35	9.01
顺序	73.86	52.70	8.86	10.23
伪代码	67.69	49.97	6.87	8.04

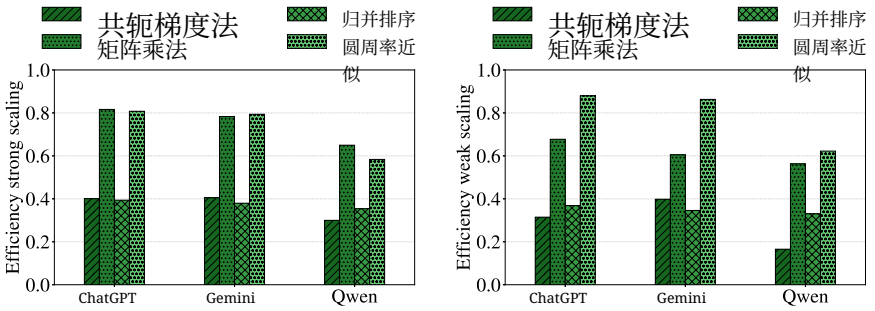
表1：生成代码在长度和复杂度方面的比较。

4.3 扩展性

接下来，我们使用公式（4）和（5）中给出的强扩展和弱扩展定义，分析生成代码的扩展行为。为此，我们修正了所有错误，包括归类为严重错误的情况。不可执行代码按零扩展处理。

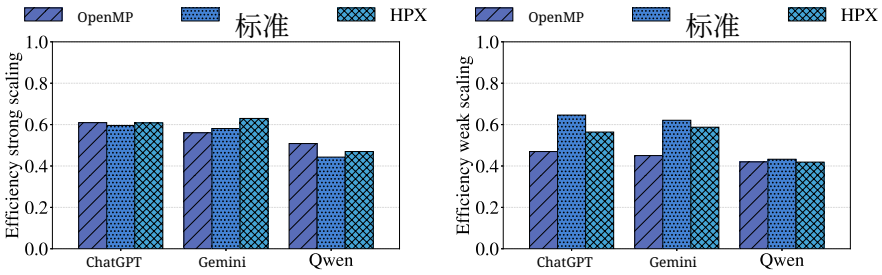
图3总结了最终的扩展性能。从这项分析中可以得出几个值得注意的观察结果。正如预期，π 近似和矩阵乘法的扩展性始终优于归并排序和共轭梯度算法。这种差异源于 π 近似和矩阵乘法都具有规律的内存访问模式，而且并行任务之间所需的同步极少。相比之下，归并排序和共轭梯度算法涉及不规则的控制流、递归或迭代依赖关系，以及频繁的归约或同步点。随着核心数量增加，这些特征会限制可实现的加速比。ChatGPT 和 Gemini 生成的代码呈现出非常相似的扩展趋势，并且总体上能在核心数量增加时保持性能；而 Qwen-Coder 的实现扩展性则明显较差。这种扩展性不佳在 π近似中尤为明显，理论上，由于其结构本身具有并行性，随着核心数量增加，它应能实现接近线性的加速比。这一限制背后的一个主要因素是，Qwen-Coder 经常在 HPX 变体中使用单个全局随机数生成器。因此，所有并发任务（包括我们实验中最多同时运行的 128 个线程）都必须访问同一个生成器来获取随机值。

如果按并行化框架而非基准问题对结果进行分类，我们就能找出框架与模型的最佳组合。图4中的数据表明，C++ 标准并行在所有模型中始终实现最高的弱扩展效率。相比之下，OpenMP 的效率最低。生成的 OpenMP 代码经常会混淆



(a) 强扩展 (b) 弱扩展

图 3：按基准问题分组的生成代码的扩展。



（a）强扩展 （b）弱扩展

图4：按框架分组的生成代码的扩展性。

基于任务的并行与经典的基于循环的并行，例如，错误地将 single 或并行区域与 parallel for 构造混用。

尽管在弱扩展方面各框架之间的差异更大，但所有框架的强扩展性能非常相似。只有Qwen-Coder生成的代码效率略低。结果表明，对于所评估的大语言模型生成的代码而言，C++ 标准并行是最稳健的选择；而如果模型能够正确处理框架特定细节，HPX也可以具有竞争力。最后，

最后，我们使用公式（3）中的 PCGQS 分数评估受测大语言模型。表 2 展示了所有实验、框架和提示级别的综合得分。ChatGPT 的整体性能最强，而开源模型通义千问在响应正确性方面存在明显困难，显著拉低了其综合得分。

	ChatGPT	Gemini	通义千问
PCGQS 分数	0.7425	0.7008	0.5702

表 2：评估的大语言模型的 PCGQS 分数。

5 结论与展望

在本研究中，我们系统地评估了大语言模型在多个基准测试中生成任务并行代码的能力，这些基准测试涵盖了算法结构和依赖复杂度各异的问题。结果表明，大语言模型能够可靠地处理易于并行化的问题和简单的任务图，说明它们对基本并行抽象有扎实的理解。对于更复杂的模式，例如具有内在同步需求的迭代算法，这些模型往往会生成次优或错误的任务依赖。这些缺陷表现为缺少同步或潜在的数据竞争，尽管代码在语法上正确。所提出的评估指标（包括 PCGQS）能够超越功能正确性来区分模型，同时衡量结构并行性和可扩展性潜力。

关于未来工作，我们计划将基准测试套件扩展至规模更大、结构更不规则且包含多个阶段的并行应用。通过改进提示、明确的依赖表示以及工具辅助生成工作流，或可进一步提升效果。总体而言，这些方向旨在提高大语言模型在异步、基于任务的并行编程中的可靠性和实用价值。最后，智能体人工智能的近期进展显著提升了代码生成性能，因此有必要在异步基于任务的并行化框架中对此进行系统研究。

人工智能使用披露

我们使用了 Grammarly [17],DeepL [18], Gemini [19], 和 ChatGPT [20], 等生成式人工智能工具，以提升稿件的清晰度、语法和整体连贯性。所有技术内容、数据分析和研究发现均由作者独立构思并完成。作者仔细审阅、核实并编辑了人工智能辅助生成的内容，以确保事实准确、诠释严谨且符合学术诚信。最终稿件体现了作者原创的智力贡献和分析工作。

参考文献

1. D. Nichols, J. H. Davis, Z. Xie, A. Rajaram, and A. Bhatele，“大语言模型能编写并行代码吗？”，载于第*33*届高性能并行与分布式计算国际研讨会论文集，HPDC ’24，第281–294页，ACM，2024年6月。

2. D. Yadav and S. Mondal，“评估预训练大语言模型在用于源代码并行化的零样本提示上的表现”，《系统与软件杂志》，第230卷，第112543页，2025年。 3. OpenAI，“GPT-4技术报告”，2023年。 4. DeepSeek-AI，“DeepSeek-V3技术报告”，2025年。 5. H. Touvron, L. Martin, 等，“Llama 2：开放式基础模型与微调聊天模型”，2023年。 6. M. Chen, J. Tworek, 等，“ 评估基于代码训练的大语言模型 ”，2021 年。 7. P. Diehl, N. Nader, M. Moraru, and S. R. Brandt，“使用Llama 2进行大语言模型基准测试：评估其在多种编程语言中的代码开发性能”，《建模与计算机器学习杂志》，第6卷，第3期，第95–129页，2025年。 8. P. Diehl, N. Nader, S. Brandt, and H. Kaiser，评估人工智能生成的*C*代码++、*Fortran*、*Go*、*Java*、*Julia*、*MATLAB*、*Python*、*R*和*Rust*代码，第243–254页。施普林格·自然瑞士，2025年。 9. N. Nader, P. Diehl, S. Brandt, and H. Kaiser，大语言模型与高性能计算：*DeepSeek*在高性能计算任务中的性能基准测试，第626–638页。施普林格·自然瑞士，2025年11月。 10. Q. I. Mahmud, A. TehraniJamsaz, H. Phan, L. Chen, M. Capotă, T. Willke, N. K. Ahmed, and A. Jannesari，“AutoParLLM：使用图神经网络引导的上下文生成，通过大语言模型实现零样本代码并行化”，2025年。 11. W. Godoy, P. Valero-Lara, K. Teranishi, P. Balaprakash, and J. Vetter，“评估OpenAI Codex在高性能计算并行编程模型内核生成中的表现”，载于第*52*届国际并行处理会议研讨会论文集，ICPP-W 2023，第136–144页，ACM，2023年8月。 12. P. Valero-Lara, A. Huante, M. A. Lail, W. F. Godoy, K. Teranishi, P. Balaprakash, and J. S. Vetter，“比较Llama-2和GPT-3大语言模型在高性能计算内核生成方面的表现”，2023年。 13. D. Nichols, A. Marathe, H. Menon, T. Gamblin, and A. Bhatele，“HPC-Coder：使用大语言模型为并行程序建模”，载于*ISC 2024*高性能计算研究论文集（第*39*届国际会议），第1–12页，IEEE，2024年5月。 14. W. F. Godoy, P. Valero-Lara, K. Teranishi, P. Balaprakash, and J. S. Vetter，“面向高性能计算软件开发的大语言模型评估”，《并发与计算：实践与经验》，第36卷，2024年9月。 15. M. Chen, J. Tworek, H. Jun, 等，“评估基于代码训练的大语言模型”，2021年。 16. T. McCabe，“ 复杂度度量 ”，《 IEEE 软件工程汇刊》，第 SE-2 卷，第 4 期，第 308–320页，1976年。 17. Grammarly, Inc.，“Grammarly。” https://www.grammarly.com/，2026年。访问日期：2026年1月14日。 18. DeepL SE，“DeepL 翻译。” https://www.deepl.com/translator，2026年。访问日期：2026年1月14日。 19. Google LLC，“Google Gemini 3。” https://gemini.google.com/，2026年。访问日期：2026年1月14日。 20. OpenAI，“ChatGPT 5。” https://openai.com/chatgpt，2025年。访问日期：2026年1月14日。