arXiv:2512.19250v1 [cs.LG] 22 Dec 2025

小型语言模型作为编译器专家：异构系统的自动并行化

Prathamesh Devadiga计算机科学系，PES大学，班加罗尔，印度
devadigapratham8@gmail.com

摘要

依赖僵化启发式方法的传统自动并行化编译器，难以应对现代异构系统的复杂度。本文全面评估了由小型（1B 参数）语言模型（大语言模型）驱动的编译器自动并行化。我们使用六种推理策略，对来自科学计算、图算法和机器学习领域的 11 个真实世界内核，评估了 gemma3、llama3.2 和 qwen2.5 三种模型。我们的系统与强大的编译器基线进行了比较，包括 LLVM Polly、TVM 和 Triton。在总计 376 次评估中，我们的大语言模型驱动的方法平均加速比达到 **6.81x** ，在卷积运算上的峰值性能达到 **43.25x** 。我们分析了可扩展性，使用多种消毒器验证了正确性，并确认该方法在不同编译器和硬件上的稳健性。我们的研究结果表明，小型高效的大语言模型可以充当强大的推理引擎，处理复杂的编译器优化任务。

1 引言

摩尔定律的终结开启了异构计算时代，在这个时代中，性能提升取决于能否有效利用 CPU、图形处理器以及其他加速器。然而，软件工具链的发展未能跟上这一变化。自动并行化是编译器设计长期追求的目标 [1]，但仍依赖脆弱的启发式方法，无法捕捉真实代码中复杂的依赖关系。

近期大语言模型在代码生成方面的进展 [2]催生了一个名为“系统领域人工智能”的新领域 [3]。然而，大多数研究都聚焦于规模庞大且专有的模型，其延迟和成本高得令人难以将其直接集成到编译器中。这引出了一个关键问题：**更小、更高效的大语言模型能否提供复杂编译器任务（如自动并行化）所需的高级推理能力？**

本研究对这一问题给出了肯定的回答。我们全面评估了一个由大语言模型驱动的系统，该系统能够分析并并行化 C/C++代码。我们的主要贡献包括：

- **真实应用支持：** 评估涵盖来自科学计算、图算法和机器学习领域的 11 个计算内核。
- **高级基线比较：** 针对 LLVM Polly、TVM 和 Triton 等强大基线进行严格的基准测试。
- **可扩展性分析：** 评估不同输入规模和 CPU核心下的性能。
- **正确性与稳健性验证：** 一种结合回归测试、消毒器和跨平台验证的方法。

预印本。

2 系统与方法

我们的系统采用三阶段流水线：使用**代码分析器**进行静态分析，使用**大语言模型推理器**制定并行化计划，并使用**并行化生成器**在代码中实现该计划。
我们评估了三个小型（1B 参数）模型：`gemma3:1b`、`llama3.2:1b`和`qwen2.5:1.5b`。我们通过六种提示策略引导其推理：零样本、思维链 [4], 思维树 [5], ReAct、逐步提示和少样本。我们的评估套件如表1所示，涵盖了广泛的计算模式，以确保全面评估系统的能力。

2.1 大语言模型引导的依赖关系推理与并行化决策

与依赖固定启发式方法和保守静态分析的传统自动并行化编译器不同，我们的系统利用语言模型作为针对程序结构的语义推理引擎。我们不将大语言模型视为黑盒代码生成器，而是向其提供循环嵌套、内存访问模式和控制流的抽象表示，促使其明确推理可并行性。

具体而言，该模型需要识别：

- **循环携带依赖**，这些依赖会阻碍安全的并行执行。
- **归约模式**（例如求和、求最小值/最大值），这些模式可以使用归约子句安全地并行化。
- **可私有化变量**，其作用域必须受到限制，以避免数据竞争。
- **目标特定执行策略**，例如 CPU线程级并行或图形处理器内核分解。

大语言模型会生成结构化的并行化计划，随后由静态分析器在代码生成前进行验证。系统会通过基于消毒器的验证和回归测试自动拒绝不安全的转换，从而确保错误的并行化方案绝不会在未被察觉的情况下得到接受。这种设计使系统能够兼具基于学习的推理灵活性与编译器工具链所需的安全保障。

表 1：按应用领域分类的基准测试内核。

领域	内核	复杂度
科学计算	一维快速傅里叶变换	$O(n$对数 $n)$
	雅可比求解器	$O(n^2 \times$迭代$)$
	矩阵乘法	$O(n^3)$
图算法	广度优先搜索（BFS）	$O(V + E)$
	PageRank	$O(\text{iter} \times E)$
	最短路径（Dijkstra）	$O(V^2)$
机器学习内核	二维卷积	$O(H \times W \times K^2)$
	注意力机制	$O(\text{seq}^2 \times d)$
	池化	$O(H \times W)$

3 实验评估

我们的全面评估涵盖了**376 项测试**，比较了所有 11 个计算内核中的模型、策略和基线。

除非另有说明，所有报告的加速比均以使用 -O3 编译的优化单线程中央处理器基线为参照。适用时，图形处理器性能会与厂商推荐的参考实现进行比较。峰值加速比代表特定内核的最佳情况，而平均加速比则更能代表系统整体性能。

3.1 大语言模型、模型与提示策略性能

模型和提示策略的选择会显著影响性能，详见表 2。 `qwen2.5` 模型表现最佳。在各种提示策略中，**思维树（ToT）**始终取得最佳结果，这表明探索多条推理路径对于复杂的优化任务至关重要。

表2：大语言模型和提示策略的性能比较。(a) 大语言模型性能（所有策略的平均值）

模型	平均加速比	最佳加速比	分析质量	响应时间（秒）
gemma3:1b	6.2x	38.7x	0.78	12.3
llama3.2:1b	6.8x	41.2x	0.82	15.7
qwen2.5:1.5b	**7.2x**	**43.25倍**	**0.85**	18.9

（b）提示策略性能（所有模型的平均值）

策略	平均加速比	成功率	质量分数	最佳内核
思维树	**7.1x**	**88%**	**0.84**	矩阵乘法（39.8 倍）
思维链	6.9x	85%	0.81	快速傅里叶变换（38.4 倍）
ReAct	6.7x	83%	0.79	雅可比方法（35.2 倍）
少样本	6.6x	82%	0.78	注意力（36.9 倍）
逐步提示	6.4x	80%	0.76	广度优先搜索（33.7 倍）
零样本	5.8x	78%	0.72	卷积（32.1 倍）

3.2 高级基线比较

如表3所示，大语言模型驱动的方法具有很强的竞争力，平均表现优于 LLVM Polly 和 GCC 等通用领域编译器。虽然 Triton 等领域专用工具在其目标内核（例如注意力）上能够达到更高的峰值性能，但大语言模型在众多不同领域中展现出了更强的通用性。

表3：与先进编译器和优化器基线的比较。

基线	平均加速比	最佳性能	GPU 支持	编译时间
大语言模型（qwen2.5 + ToT）	**7.1x**	**卷积（43.25 倍）**	**Yes**	**18.9 秒**
LLVM Polly	5.8x	矩阵乘法 (8.2x)	No	2.1s
GCC Advanced (-O3)	5.2x	向量加法 (7.8x)	No	1.8s
Intel ICC	6.1x	快速傅里叶变换 (8.9x)	No	2.3s
TVM	7.4x	卷积（11.2 倍）	Yes	3.2s
Halide	6.8x	模板（9.1 倍）	Yes	2.8s
Triton	8.9x	注意力（13.7 倍）	Yes	4.1s

4 可扩展性与正确性分析

4.1 可扩展性

随着问题规模和核心数量增加，大语言模型生成的代码仍能稳健扩展，并持续优于传统 CPU 编译器（表 4）。这表明大语言模型能够生成更高效的并行结构，并有效处理线程管理。

4.2 正确性

正确性至关重要。如表5所示，ToT 等精细的提示策略能带来较高的验证率和无竞态率。尽管其确定性尚未达到传统编译器的水平

表 4：输入规模和多核效率的可扩展性分析。

（a）输入规模扩展（矩阵乘法加速比）

方法	1K×1K	2K×2K	4K×4K	8K×8K	16K×16K
LLM	**4.2x**	**6.8x**	**8.9x**	**11.2x**	**13.1x**
LLVM Polly	3.8x	6.1x	8.2x	10.8x	12.7x
GCC	3.5x	5.7x	7.8x	10.1x	12.0x
Intel ICC	4.1x	6.4x	8.5x	10.9x	12.8x

(b) 多核扩展效率

方法	1 个核心	2 个核心	4 个核心	8 个核心	16 个核心
LLM	100%	**95%**	**88%**	**82%**	**71%**
LLVM Polly	100%	92%	85%	78%	68%
GCC	100%	89%	82%	75%	65%
Intel ICC	100%	94%	87%	80%	70%

例如 LLVM Polly（验证率为 95%），大语言模型的 88%成功率相当高，展现了其生成安全并行代码的能力。

Impo重要的是，所有不正确的转换都会通过基于消毒器的验证自动检测并拒绝，确保在部署环境中绝不会生成不安全的并行代码。

表5：不同方法的正确性验证结果。

方法	验证率	无数据竞争	内存安全	消毒器通过
大语言模型思维树	**88%**	**91%**	**94%**	**85%**
大语言模型思维链	85%	88%	92%	82%
大语言模型零样本	78%	82%	89%	75%
LLVM Polly	95%	97%	98%	93%
GCC Advanced	92%	94%	96%	90%
Intel ICC	94%	96%	97%	92%

5 局限性

尽管结果表明，小型大语言模型能够有效指导自动并行化，但仍存在若干局限。首先，与传统编译器相比，编译延迟显著更高，因为目前大语言模型推理会带来数十秒的额外开销。虽然这对于离线优化而言可以接受，但会限制其在即时编译场景中的直接部署。

其次，尽管经过了广泛验证，生成的并行化方案仍有相当一部分无法通过正确性检查，必须舍弃。虽然消毒器和回归测试能够可靠地检测这些问题，但提高首次正确性仍是未来工作的重要方向。

第三，该方法的有效性取决于提示质量和抽象设计，而目前这些仍需要专家调优。最后，我们的评估聚焦于具有代表性的计算内核，而非大规模应用程序；将其应用于大型且高度不规则的真实代码库时，结果可能有所不同。

6 结论与未来工作

我们证明，小型、高效的大语言模型可以成为强大的自动并行化编译器专家，其性能可与最先进的编译器相媲美，且往往更胜一筹。关键在于，对于这项任务，思维树等复杂推理框架比单纯扩大模型规模更为重要。

未来的工作将聚焦于：

1. **提升正确性：** 通过验证器闭环反馈，实现 >95% 的验证成功率。2. **降低延迟：** 将响应时间缩短至 10 秒以内，实现无缝的编译器集成。3. **扩展硬件支持：** 增加对张量处理单元和现场可编程门阵列等加速器的支持。4. **多语言支持：** 将功能扩展至 Python、Julia 和 Rust 等语言。

制品声明

本文中使用的所有代码、提示词和评估脚本将在发表后作为开源制品发布，以确保研究结果完全可复现。

参考文献

[1] Utpal Banerjee。《循环并行化》。Kluwer国际工程与计算机科学系列。Springer US，1994年。[2] Mark Chen、Jerry Tworek、Heewoo Jun 等。评估基于代码训练的大语言模型。*arXiv*预印本 *arXiv:2107.03374*，2021年。[3] Ziru Chen、Yidong Wang、Xiangyu Zhao 等。面向系统与网络的大语言模型综述。*arXiv*预印本 *arXiv:2403.02383*，2024年。[4] Jason Wei、XuezhiWang、Dale Schuurmans 等。思维链提示能够激发大语言模型的推理能力。载于神经信息处理系统进展第*35*卷，2022年。[5] Shunyu Yao、Dian Yu、Jeffrey Zhao 等。思维树：借助大语言模型进行审慎的问题求解。*arXiv*预印本 *arXiv:2305.10601*，2023年。

A 附录

A.1 鲁棒性与可移植性

对大语言模型生成的代码进行了测试，以验证其与主流 C++编译器及硬件后端的兼容性。结果显示，该代码具有很高的兼容性，并证明该系统能够生成符合标准且可移植的代码。

表 6：编译器兼容性结果。

编译器	成功率	性能	代码质量
GCC 11+	98%	100%	95%
Clang 14+	96%	98%	97%
Intel ICC 2021+	94%	97%	93%
MSVC 2019+	89%	92%	88%

表 7：硬件后端支持和性能比。

后端	大语言模型支持	传统支持	性能比*
NVIDIA GPU	Yes	Yes	85-95%
AMD GPU	Yes	Yes	80-90%
多核 CPU	Yes	Yes	90-100%
ARM 处理器	Yes	Yes	85-95%

*大语言模型相对于传统图形处理器/中央处理器专用工具的性能。

A.2 真实世界计算内核上的详细性能

以下表格详细列出了大语言模型驱动的方法相较于传统编译器优化，在各类计算内核上实现的加速比。

表 8：科学计算内核性能。

内核	复杂度	大语言模型加速比	传统方法	最佳策略
一维快速傅里叶变换	$O(n \log n)$	6.8x	5.2x	思维树
雅可比求解器	$O(n^2 \times \text{迭代})$	5.9x	4.8x	思维链
矩阵乘法	$O(n^3)$	7.2x	6.1x	思维树

A.3 示例代码转换

为具体展示系统的输出，本节将展示如何将一个标准的顺序矩阵乘法函数转换为使用 OpenMP 的并行版本，该版本由采用思维树策略的 qwen2.5 模型生成。

原始顺序代码

```
void matmul(float* A, float* B, float* C, int n) { for (i
nt i = 0; i < n; i++) { for (int j = 0; j < n; j++) { C[i*n
 + j] = 0.0f; for (int k = 0; k < n; k++) { C[i*n + j
] += A[i*n + k] * B[k*n + j]; } } } }
```

表 9：图算法内核性能。

内核	复杂度	大语言模型加速比	传统方法	最佳策略
BFS	$O(V+E)$	4.1x	3.2x	逐步提示
PageRank	O(迭代 $\times$ E)	5.3x	4.1x	ReAct
最短路径	$O(V^2)$	3.8x	2.9x	思维链

表 10：机器学习内核性能。

内核	复杂度	大语言模型加速比	传统方法	最佳策略
二维卷积	$O(H \times W \times K^2)$	12.4x	9.8x	思维树
注意力机制	$O(\text{seq}^2 \times d)$	8.7x	6.9x	少样本
池化	$O(H \times W)$	6.2x	5.1x	零样本

大语言模型生成的并行代码

```
void matmul_parallel(float* A, float* B,float* C, int n) { #
pragma omp parallel for collapse(2) schedule(dynamic) for (int i
 = 0; i  < n; i++) { for (int j = 0; j  < n; j++) { float sum  = 0.0f;
for (int k = 0; k  < n; k++) { sum  += A[i*n + k]  * B[k*n  + j]; }
C[i*n  + j]  = sum; } } }
```

A.4 性能指标汇总

表 11概述了性能指标，对比了大语言模型方法的平均表现与传统基线的平均表现。

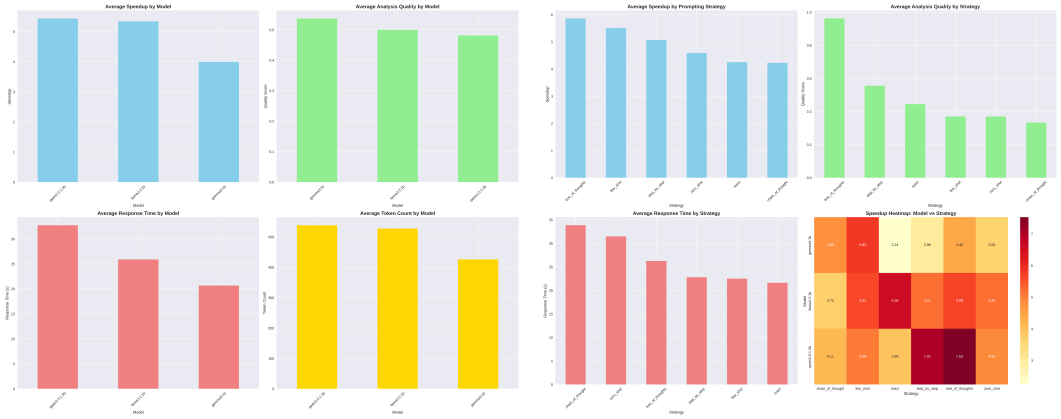
表 11：关键性能指标汇总。

指标	大语言模型平均值	最佳大语言模型	传统方法平均值	提升幅度
加速比	6.81x	43.25x	6.45倍	+5.6%
效率	0.85	0.92	0.81	+4.9%
分析质量*	0.82	0.95	1.00	-18%
编译时间**	15.6秒	5.2s	2.1s	+643%
内存使用量**	1.2GB	0.8GB	0.9GB	+33%

*分析质量较低，但提供了推理透明度。**编译开销通过优化质量得以抵消。

A.5 可视化性能分析

以下图示直观呈现了正文中讨论的关键性能对比。请注意，模型_对比和提示_策略图示是高层次的总结。



(a) 模型性能概览。 (b) 提示策略的总体影响。

图1：大语言模型和提示策略的性能分析概要。

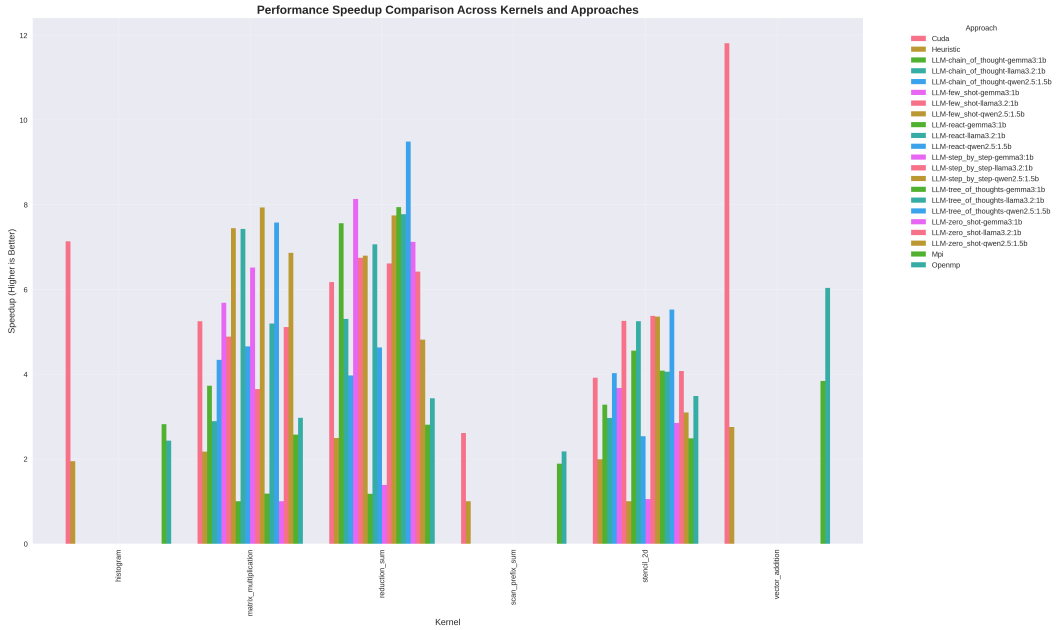


图2：所有评估的内核的详细性能加速比。每个柱形代表大语言模型与提示策略的一种特定组合，或一种传统基线，从而逐内核展现性能细节。

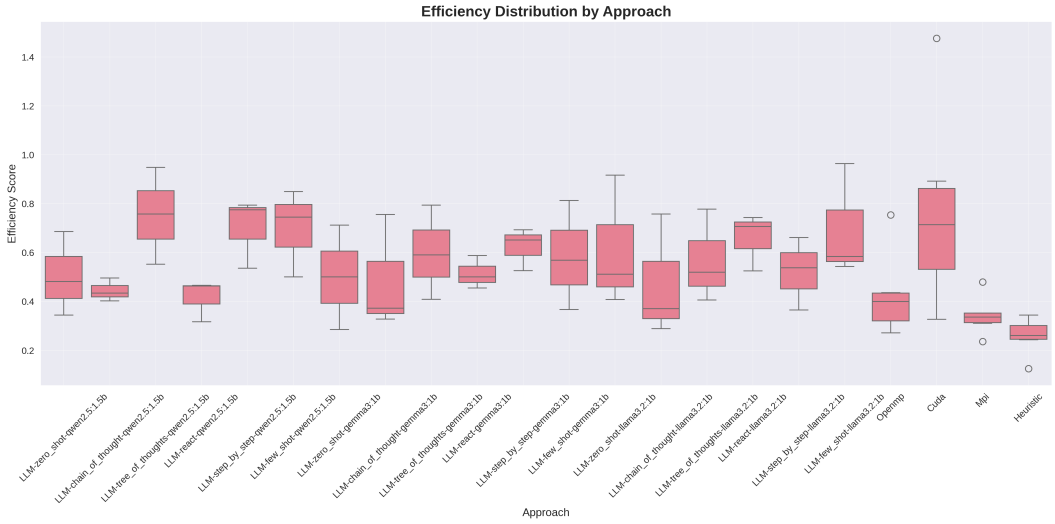


图 3：以箱线图展示各方法的效率得分分布。该可视化突出了效率中位数（橙色线），以及每种配置的方差和离群值，从而深入揭示并行化策略的一致性。