arXiv:2601.02045v1 [cs.PL] 5 Jan 2026

新编译器栈：大语言模型与编译器协同作用综述

Shuoming Zhang[1,2], Jiacheng Zhao[1, 2*], Qiuchu Yu[1,2], Chunwei Xia[3], Zheng Wang[3], Xiaobing Feng[1,2], Huimin Cui[1,2]

[1*]SKLP，中国科学院计算技术研究所，中国北京科学院南路6号。[2]中国科学院大学，中国北京。[3]英国利兹大学，英国。

*通讯作者。电子邮箱：zhaojiacheng@ict.ac.cn；贡献作者：zhangshuoming21s@ict.ac.cn；yuqiuchu19@mails.ucas.ac.cn；C.Xia@leeds.ac.uk；Z.Wang5@leeds.ac.uk；fxb@ict.ac.cn；cuihm@ict.ac.cn；

摘要

本综述通过回答若干关键研究问题，对新兴的大语言模型赋能的编译领域进行了系统概述。我们首先提出了一个全面的多维分类体系，根据设计理念（选择器、翻译器、生成器）、大语言模型方法、其运行所处的代码抽象层级，以及所处理的具体任务类型，对相关工作进行分类，从而回答了大语言模型是如何被集成的问题。在回答这些方法带来了哪些进展时，我们确定了三项主要优势：编译器开发的民主化、新型优化策略的发现，以及编译器传统适用范围的拓展。最后，在探讨该领域的挑战与机遇时，我们强调了确保正确性和实现可扩展性这两项关键难题，同时指出混合系统的开发是最具前景的前进方向。通过给出这些问题的答案，本综述为研究人员和从业者提供了一份基础性路线图，描绘了新一代由大语言模型驱动的智能、自适应且协同的编译工具的发展方向。

关键词：综述，大语言模型，编译器，代码翻译，代码优化

1 引言

几十年来，编译器一直是计算栈的基石，承担着一项关键而复杂的任务：将人类可读的源代码转换为高效的机器可执行程序。在这一过程中，主要挑战之一是优化。传统上，这一领域由人类专家设计的复杂手工启发式方法主导，用以探索庞大而复杂的决策空间。机器学习的出现（Krizhevsky 等，2012；He 等，2016）引入了一种新范式，利用数据驱动模型处理阶段排序和标志选择等任务（Wang 和 O'Boyle，2018）。然而，这些传统机器学习方法通常依赖繁重的特征工程流程：专家必须精心设计并提取程序特征来训练模型，而编译器设计的核心组件基本保持不变。

大语言模型（LLM）的近期兴起代表了这一领域的根本性转变。它们在海量文本和代码语料上进行预训练，大语言模型已展现出理解、生成和转换编程语言原始文本的非凡能力，基本消除了对显式特征工程的需求。通过在规模比任何人能够研究的代码库大几个数量级的数据上进行训练，大语言模型掌握了对众多语言中的编程模式、语法和语义的深层理解。这些能力已通过大语言模型驱动的聊天机器人（OpenAI 2024；Gem-ini Team and Google 2023；Anthropic 2024）、代码助手（Chen 等，2021b；Tabnine 2022）和半自动化智能体（Anysphere, Inc. 2023；google-gemini2025；anthropics 2025）迅速融入软件开发工作流，在每个阶段提升开发者效能。反过来，这股创新浪潮也催生了更广泛编译器领域内的新研究。任务范围已从编译优化遍选择等狭窄问题，扩展到代码转译和自动化程序修复等雄心勃勃的端到端目标。因此，编译器的角色正从静态工具重新构想为软件开发中动态且可交互的合作伙伴。

这一快速演进促成了编译器相关任务各个领域中新研究的激增。研究人员如今正将大语言模型应用于一系列雄心勃勃的目标，包括利用源到源代码转译在不同语言之间迁移代码（Roziere 等人（2020）；Jana 等人（2024）；Zhang 等人，2025；Ibrahimzada 等人，2025）以及在不同架构之间迁移代码（Wen 等，2022；Tehrani-Jamsaz 等人，2024；Palkowski 和 Gruzewski，2024；Dong 等，2025）；利用高级代码优化超越传统启发式方法（Gao 等人，2025；Peng 等人，2025；Purschke 等人，2025；Xu 等人，2025；Lin 等人，2025）；利用低级中间表示优化微调性能和代码大小（Deng 等，2024；Cummins 等，2025；Cui 等，2025）；甚至研究使用大语言模型充当编译器（Gao 等，2024；Zhang 等，2024；Zhong 等人，2025）或反编译器这一极其困难的问题（Xu 等人，2023；Armengol-Estape 等人，2024；Tan 等人，2024；Hu 等人，2024；Wong 等人，2025）。尽管前景广阔，但这些研究的数量和多样性之大，可能会妨碍人们清晰理解当前的研究格局。为弥补这一空白，本文对大语言模型赋能的编译器研究近期进展进行了系统综述。我们提出了一个关于该领域最先进成果的全面多维分类体系，并清晰分析其核心进展与挑战，为编译器研究这一激动人心的方向确立路线图。

本文的主要贡献如下：

- 我们开展了系统性文献综述，识别并整理出由 159 项主要研究构成的语料库，代表了大语言模型赋能编译领域的最新进展。
- 我们提出了一种新颖的多维分类体系，用于梳理该领域。该分类体系依据大语言模型的**设计理念**（选择器、翻译器、生成器）、**大语言模型方法**、其运作的**代码抽象层级**以及所针对的具体**任务类型**，对现有研究进行分类。
- 我们全面分析了基于大语言模型的方法带来的主要进展，重点阐述其在推动编译器开发普及、发现新型优化以及拓宽编译范围方面的作用。
- 我们归纳了该领域面临的共同挑战，包括正确性、可扩展性和可解释性，并探讨了有望取得进展的未来研究方向，例如开发混合系统和自我改进编译器。

本文其余部分的组织如下。§ 2 详细介绍我们旨在回答的研究问题，以及用于文献筛选的系统性方法论。§ 3 和§ 4 详细呈现我们的多维分类体系，对现有研究进行分类。§ 5系统介绍该领域具有代表性的数据集和基准测试，以及相关的最新进展。§ 6深入讨论该领域的进展、挑战和未来机遇。最后，§ 7总结全文。

2 方法论

本节详细介绍了我们为调查大语言模型赋能的编译器研究现状而采用的系统性方法。严格且透明的方案对于确保综述全面且可复现至关重要。为此，我们首先在§§ 2.1中介绍指导我们开展研究的问题。随后，我们将在§§2.2中描述用于整理分析所需主要研究最终语料的结构化三阶段文献检索与筛选方案。

本节介绍开展文献综述的详细步骤。为确保对该领域的综述全面且公正，我们遵循标准实践，采用了系统性文献综述（SLR）方案（Kitchenham 等，2007）。该方案明确了指导综述的研究问题、用于识别相关研究的搜索流程，以及纳入或排除论文的标准。

2.1 研究问题（RQs）

为指导文献综述并为分析提供清晰的框架，我们围绕大语言模型在编译器领域的应用提出了以下四个关键研究问题：

- **研究问题 1：大语言模型如何集成到编译过程中？**我们从两个互补的视角探讨这一问题：首先是**设计理念**，它定义了大语言模型在系统中的架构角色（任务侧）；其次是**大语言模型方法**，它详细说明了模型如何在技术层面针对该任务进行开发和应用（大语言模型侧）。
- **研究问题 2：大语言模型主要处理哪些编译器相关任务？**这一问题旨在识别并归类近期研究关注的具体编译器相关任务，例如代码优化、转译和代码生成。
- **研究问题3：基于大语言模型的方法取得了哪些主要进展？**这一问题着重总结这些技术为编译器和机器学习领域带来的创新贡献与突破。
- **研究问题4：这一新兴领域面临哪些共同挑战，又有哪些未来机遇？**该问题旨在综合文献中报告的主要障碍，并确定未来研究的潜在方向。

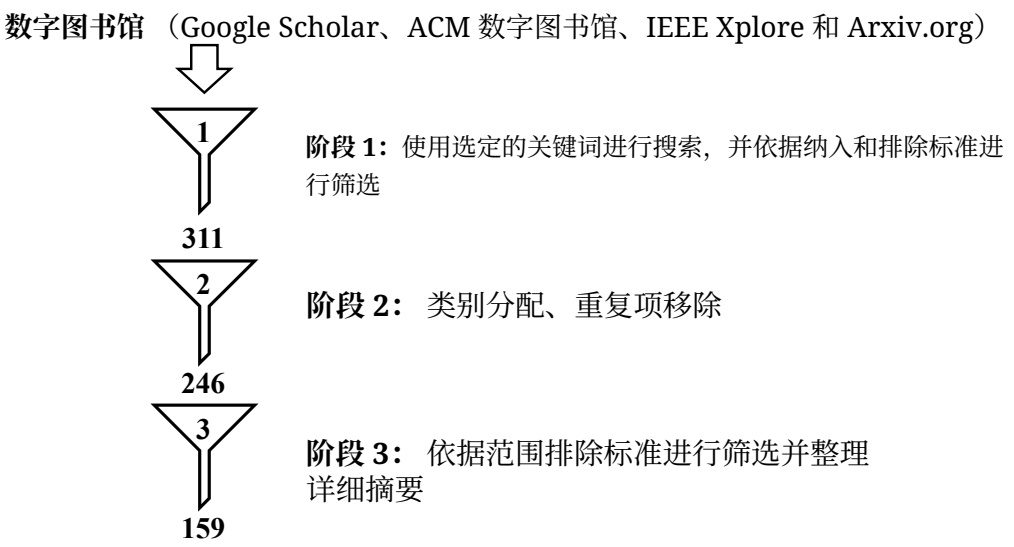

图 1：文献检索与筛选方案概览

2.2 文献检索与筛选方案

为系统地回答上述研究问题，我们设计并执行了一套全面的文献搜索与筛选方案。该流程分为三个不同阶段，逐步从大量初始候选论文中筛选出一个聚焦且相关的文献集，共计159篇论文。

1. **阶段1：初步搜索与候选论文收集**。第一阶段是在四个主要数字图书馆中广泛搜索候选论文：arXiv、Google Scholar、ACM Digital Library 和 IEEE Xplore。我们采用了一系列广泛的检索查询，包括“LLM compiler”（大语言模型编译器）、“large language model for code optimization”（用于代码优化的大语言模型）、“AI in compilers”（编译器中的人工智能）、“transformer for compilation”（用于编译的 Transformer）和“LLM for code generation”（用于代码生成的大语言模型）等关键词。为补充自动化搜索，我们还手动纳入了领域专家指出的几篇开创性论文。随后，我们采用滚雪球法，查阅这些核心论文的前向引用链，以确保覆盖全面，并纳入其他引用了这些论文的密切相关研究。

2. **阶段 2：去重与初步筛选**。在第二阶段，我们对初始论文集进行处理，删除重复项。随后，我们对剩余文章进行初步筛选。此步骤包括审阅标题和摘要，剔除明显不在本综述范围内的研究，例如与编译器无关或未使用大语言模型的研究。

3. **第三阶段：全文审查与最终筛选**。 最后阶段对通过初步筛选的每篇论文进行了全文审查。通过这项深入分析，我们得以确定研究是否最终符合资格，依据的是一套严格的纳入与排除标准。

纳入标准：

- 研究必须主要聚焦于使用大语言模型（具体而言，基于 Transformer 的模型）执行与编译或代码优化相关的任务。
- 研究必须是同行评审会议论文、期刊文章，或具有重要意义且相关的预印本手稿（以纳入通常发表于 arXiv 的最新进展）。
- 研究全文必须可通过数字图书馆或开放获取网站公开获取。

排除标准：

- 未使用语言模型（LM）的传统机器学习研究（例如，支持向量机、决策树）。
- 非技术研究论文类文章，例如社论、主题演讲、教程、海报或细节不足的扩展摘要。
- 模型的核心应用与更广泛的编译领域研究兴趣无关的研究。

通过这一严格的三阶段流程，我们最终筛选出 159 项主要研究。这一整理后的语料构成了本文后续章节所述系统综述、分类和分析的基础。

2.3 分类体系的范围

在介绍详细的分类体系之前，有必要先界定本综述的范围。“由大语言模型赋能的编译器”这一术语涵盖了广泛的新兴研究。对于本综述，我们将范围界定为：任何利用大型预训练 Transformer 语言模型来执行或增强传统上与

编译工作流，或更广泛的代码开发与优化生命周期相关的任务的研究。

该定义有意区分了基于大语言模型的方法这一新范式与以往工作。具体而言，**我们的综述不包括**：

- **传统机器学习技术**：依赖经典机器学习模型（例如支持向量机、决策树、随机森林）的研究。这些模型需要从源代码中进行大量手工特征工程（例如提取代码质量指标），且相关方法已经得到充分研究和综述。
- **纯基于自然语言处理的软件工程任务**：将语言模型应用于软件工程制品，但不直接分析、转换或生成代码的研究不在本文范围内。例如，错误报告分类或开发者注释摘要虽然与之相关，但超出了我们以编译器为中心的研究重点。

通过确立这一范围，我们旨在对将大语言模型应用于编译科学与工程这一具体且变革性范式的最新研究成果进行聚焦而连贯的综述。任何编译器的核心功能从根本上都植根于两个过程：翻译 和优化。因此，这两个领域构成了我们综述的主要重点。我们纳入了大量关于**代码优化**的工作，其中大语言模型被用于重写程序，以提升性能或减小代码规模，处理对象既可以是高级源代码，也可以是低级中间表示（IR）。

同样重要的是**代码转译**，我们将其广义地定义为不同程序表示之间的翻译。这包括狭义定义的编译（例如，从编程语言到汇编语言）、反编译、源到源转译以及二进制翻译。此类别还涵盖一些独特任务，例如将编程语言翻译为硬件描述语言（HDL），甚至翻译为神经网络权重。

需要注意的是，这些任务类别并非总是互斥的；事实上，它们往往彼此重叠。一个突出的例子是将 C 代码转换为 CUDA，这既可以视为用于语言迁移的代码转译任务，也可以视为旨在释放特定硬件并行性能的代码优化任务。

除了这些核心功能，我们的综述很大一部分内容专门讨论与正确性和验证相关的任务。这一关注点直接源于大语言模型的一个基本特征：它们是强大的生成模型，但并非完全可靠。与传统的确定性编译器不同，任何将大语言模型用作直接操作工具来修改代码的工作流程，都需要配套的验证流程，以确保其输出的正确性。

因此，我们全面回顾了**自动程序修复**和**错误修复**，即利用大语言模型修正缺陷。此外，我们还涵盖了**程序合成与代码生成**，并特别关注能够强化编译器相关验证生态的任务，例如为编译器模糊测试生成有效的测试用例，或协助开发编译器自身的源代码。

我们纳入范围的159项主要研究具有很高的多样性。为了梳理这一领域，我们提出一个多维分类体系，沿着以下四个关键维度对研究进行分类：

- **设计理念**，描述大语言模型如何在架构层面集成到编译工作流中。
- **大语言模型方法**，详述模型如何针对该任务进行技术开发与应用。
- **代码抽象层级**，刻画所处理代码的表示层级（例如，源代码、中间表示）。
- **任务类型**，定义所要完成的具体编译器相关目标（例如，优化、转译）。

为了清晰地分析，我们根据这四个维度所回答的研究问题对其进行分组。

§3 将分析前两个维度：设计理念（“任务侧”）和大语言模型方法（“大语言模型侧”），以全面回答**研究问题 1：大语言模型如何集成？**。随后，§ 4 将分析最后两个维度：代码抽象级别和任务类型。我们发现，这两个维度高度相关（例如，源到源转译与中间表示优化处于不同的抽象级别）。因此，我们将结合分析，以提供连贯的概述并回答**研究问题 2：涉及哪些任务？**。

3 维度 1&2：设计理念与大语言模型方法

本节将通过分析两个互补维度——大语言模型如何集成到编译过程中？——来回答我们的第一个研究问题：**设计理念**和**大语言模型方法**。前者描述大语言模型在架构中的角色，后者则详细说明将模型应用于该角色的技术方法。

第一个维度——**设计理念**——根据大语言模型在更广泛的编译系统中所扮演的概念角色，对各种方法进行分类。这一选择是一项关键的架构决策，因为它从根本上决定了如何利用和约束大语言模型的能力。它会显著影响系统的自主性程度、可信度，以及验证其输出的复杂性。正如我们将详细介绍的，我们确定了三种主导性理念：作为选择器的大语言模型（§§ 3.1）、作为翻译器的大语言模型（§§ 3.2）以及作为生成器的大语言模型（§§ 3.3）。

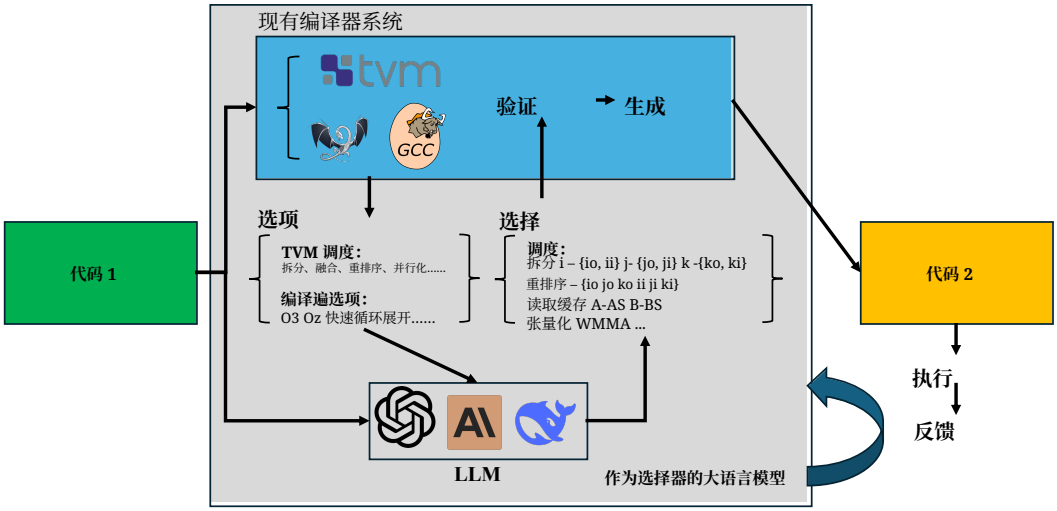

图 2：大语言模型作为选择器的方法论概览

与这种“任务侧”视角相补充的是，第二个维度——**大语言模型方法**——考察集成的“大语言模型侧”。这一维度关注使大语言模型能够胜任指定任务所采用的技术方法。这些方法分布在一个连续谱上，从“无需训练”的方法（例如上下文学习、提示工程、检索增强生成）到“需要训练”的适配方法（例如微调、领域特定预训练、强化学习）。我们将在§§ 3.4中详细分析这些方法论及其权衡。

3.1 作为选择器的大语言模型

在这一设计中，大语言模型充当复杂的策略引擎或“超优化器”。如图 2所示，其主要作用不是生成新代码，而是从编译系统提供的预定义有限选项或操作集合中选择最佳行动方案。系统会向大语言模型提供源代码上下文和一组有效选项，它则利用对上下文的深刻理解做出明智决策。随后，编译系统本身会根据大语言模型的选择执行相应的转换。

这种方法是传统编译器中机器学习技术最直接的演进，关键区别在于：它用一个强大的大语言模型取代了基于手工设计特征训练的模型，使其能够直接对原始源代码进行推理（也可以进行微调）。常见应用包括选择最优编译器标志，或为给定程序确定最有效的优化遍序列。这种模型的主要优势在于其固有的安全性和可控性：由于大语言模型只能从一系列由人定义的有效操作中进行选择，因此只要大语言模型生成了有效选择，最终的转换就能保证有效。不过，这种方法的创造潜力受限于预定义的搜索空间，无法发现可用选项中尚未编码的全新优化方法。

这一类别的研究工作可根据其主要目标和方法论大致分为以下几类：

替换或增强编译器启发式方法：这是最常见的应用，其目标是找到优于默认启发式方法（例如 -O2、-O3）的编译器标志或通道序列。这一领域有着悠久的历史，早期研究曾使用神经进化（Heckel 2023年）和Mammadli 等人（2020年）等技术来调优编译器启发式方法；Tavarageri 等人（2021年）使用深度神经网络优化多面体循环生成策略，或使用基于图的算法（Sajjadinasab 等人 2024年）优化 GCC 标志设置。如今，现代方法开始使用大语言模型，将这一原则应用于更复杂的特定领域，例如为高效模型服务选择优化策略（Tang 等人 2025年）、优化量子编译（Ren 等人 2024年），或指导高性能张量程序的生成（Zhai等人 2024年）。

提高搜索效率：一些研究不再仅仅选择最终配置，而是利用大语言模型让搜索优良配置的过程更加智能、高效。例如，CompilerDream（Deng 等，2024）利用大语言模型帮助构建优化空间的“世界模型”，进而指导更有效的搜索。另一些研究则利用大语言模型进行优先级采样（Grubisic 等，2024），以决定哪些编译器选项最值得测试；或者通过根据先前编译尝试的反馈智能地变异标志，来增强黑盒模糊测试（Wang 等人，2024年）。

智能体与强化学习（RL）框架：最先进的选择器系统会将大语言模型用作自主智能体的核心，使其能够与编译环境交互。这些智能体可以执行一系列选择操作以达成目标。多项研究将强化学习用于编译器自动调优，例如 Compiler-R1（Pan 等人，2025年），这是一个用于探索选项空间的智能体框架；以及 DeCOS（Cui 等，2025），它使用大语言模型“点燃”强化学习流程，从而提高学习的数据效率。更进一步，CompileAgent（Huet al.2025）展示了一种高级智能体，它能够选择并

协调一系列命令行工具（Git、Make 等），以自动化编译整个真实世界软件仓库这一复杂任务。

3.2 作为翻译器的大语言模型

将大语言模型用作翻译器，是最直接也最具雄心的应用方式。在这种方式下，大语言模型本身充当部分或完整的翻译器，直接对给定范围内的程序或代码片段执行一项或多项转换，例如翻译、优化或重构。这种理念将代码转换视为生成式的序列到序列任务，充分发挥大语言模型的能力，将输入程序改写为新版本。

图 3 展示了理想的大语言模型翻译器系统。首先，翻译器必须能够处理足够规模的代码输入，才能应用于真实场景。得益于编程语言的可组合性，大型项目可以分解为多个函数级代码片段，甚至进一步细化为基本块或语句；同时，通过适当的上下文管理，确保分解后的翻译结果能够重新组合。随后，将每个代码片段提交给大语言模型，提示其执行代码翻译。在此阶段，可以运用多种技术来提升翻译质量，例如思维链提示（Wei 等，2022），将大型翻译任务分解为若干连续的子任务，以降低翻译复杂度。少样本示例或检索增强生成（RAG）（Lewis 等，2020）可以通过提供翻译所需的信息来减少大语言模型的幻觉，而大语言模型的输出格式（Macedo 等，2024）也会影响其生成质量。翻译过程中，大语言模型还可以生成其他有用的代码，例如测试代码（如果未提供黄金测试），以及验证所需的断言和前置条件。

大语言模型翻译后，需要进行验证，以检查一些对质量至关重要的属性。例如，最基本的属性是检查代码能否编译；而对于内存安全检查、溢出检查和 for 循环索引检查等更具体的属性，则需要采用 SMT 求解器（De Moura 和 Bjørner，2008）等复杂验证方法，或使用 Alive2（Lopes 等，2021）等变换验证器。未能通过验证的翻译结果，需要借助编译器、运行时或行为反馈，通过自动程序修复流程进行修复。

这是多种源到源任务的主流方法，这些任务可分为以下几类：

- **语言转译：**这涉及将代码从一种高级语言翻译成另一种高级语言。文献中的案例丰富多样，包括：
  - 将 Java 等高资源语言翻译为 OCaml 等低资源语言（Cassano 等人，2024年）。
  - 将顺序执行的 C语言代码转换为并行 CUDA 代码，以充分利用图形处理器架构（Wen 等，2022年；TehraniJamsaz 等，2024年）。– 在不同的并行范式之间迁移，例如从基于单指令多线程并行的 CUDA 迁移到基于单指令多数据并行的 BANG C（Dong 等，2025年）。
- **代码优化：**重点是重写特定代码段，以提升性能或改善其他非功能属性。一个常见示例是自动重写 `for` 循环指令，以便在给定的目标硬件上实现更高性能（Taneja 等，2025）。
- **自动程序修复：**在此上下文中，错误修复被视为一个翻译问题。目标是利用大语言模型将有缺陷的程序翻译为语义正确的版本（Wei 等，2023；Xia 等，2023）。

这种方法论的主要优势在于，它具有巨大的潜力，能够学习并生成复杂且非平凡的转换，甚至可能超越人工设计的基于规则的翻译器。然而，其主要缺点是难以确保正确性。大语言模型具有概率性，因此可能会产生“幻觉”，生成语法正确但语义有误的代码，使得严格验证成为这一方法中至关重要且颇具挑战的一环。

值得注意的是，我们仅列举了这一类别中一些重要的研究，其余内容将在§ 4中详细介绍，因为代码抽象层级有助于更好地对本综述中的主要研究进行分类。

3.3 作为生成器的大语言模型

这是一种更间接的元层级方法，大语言模型的作用是生成程序的源代码，而该程序随后执行所需的代码转换。如图 4所示，其工作流程通常分为两个步骤：首先，大语言模型编写实现转换逻辑的脚本或程序；如果将其应用于编译器系统开发，这可以视为一个编译器开发过程。其次，编译并执行生成的程序，以修改目标代码；这也可以视为一个编译器测试过程。

目前，这一设计理念最常见的实现形式是人工智能驱动的代码助手和智能体：它们根据自然语言提示生成可运行的代码片段、脚本或完整应用程序。prominent 的示例包括 GitHub Copilot（Chen 等，2021b）、Cursor（Any-sphere,Inc.2023）以及命令行界面应用程序，如 gemini-cli（google-gemini 2025）和 claude- code（anthropics 2025）。

同一原则也可以扩展到编译器领域，理论上，大语言模型可以生成新的优化遍或其他编译器组件。然而，由于编译器内部结构、数据结构和应用程序接口极其复杂，且需要深厚的专业知识，应用这种方法

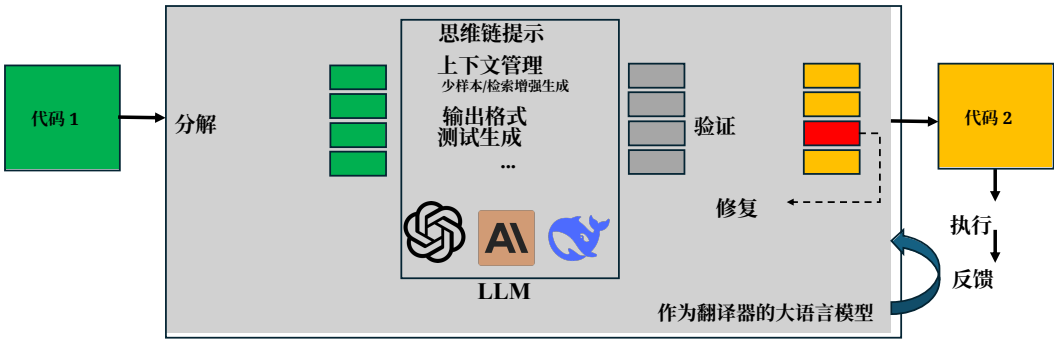

图 3：LLM 作为翻译器方法概述

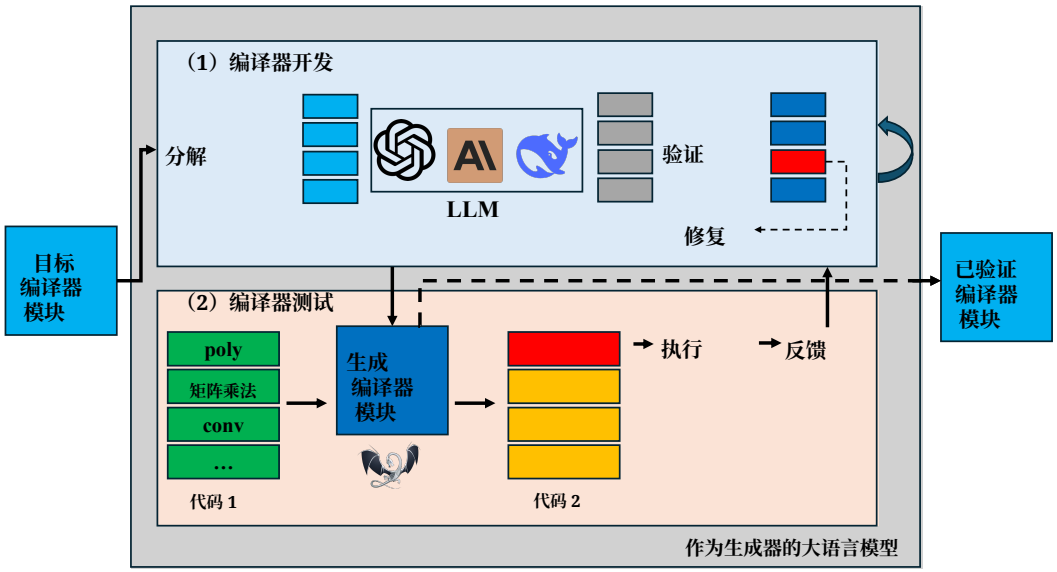

图 4：LLM 作为生成器方法概述

确实是一项重大挑战。因此，能够成功将“大语言模型作为生成器”范式用于核心编译器任务的精心设计研究仍然十分有限。尽管如此，一些开创性工作已开始探索这一前沿领域。例如，VEGA（Zhong 等人，2025年）展示了一种利用预训练 Transformer 模型自动生成编译器后端的方法，而 ComBack（Zhong 等人，2024年）则提供了一个专门设计的数据集，以促进编译器后端代码生成与增强方面的研究。在源代码方面，CodeTrans-form（Cumminset al. 2024）也初步探索了代码重写的潜力。简而言之：别重写代码，而要编写变换。

这种混合方法将大语言模型的模式识别能力与传统确定性程序的严谨性相结合。其主要优势在于，生成的产物——转换脚本——可以被检查、验证并重复使用，相较于直接生成式模型，具有更高的可信度。其主要挑战在于工作流程更加复杂，并且要求大语言模型对其需要在生成代码中使用的特定编译器 API 和框架具备深入理解。

总而言之，设计理念决定了 LLM 驱动的编译器的基本架构。文中讨论的三种范式——**选择器**、**转换器**和**生成器**——代表了一系列集成策略，每种策略都在控制力与创造潜力之间呈现出独特的权衡。选择器模型在预定义的动作空间内运行，因此能够提供最高程度的安全性；转换器模型则释放大语言模型完整的生成能力，以发现新颖的转换，但也带来了显著的验证挑战。生成器提供了一种混合方法，在生成式灵活性与传统代码系统的确定性之间实现平衡。

3.4 大语言模型方法

作为架构层面设计理念的补充，大语言模型方法详细介绍了让模型胜任指定任务所采用的技术方法。这一维度对大语言模型的技术应用方式进行了重要的比较分析，区分了“模型侧”的开发与“任务侧”的作用。这些方法大致可分为两种主要方法：**需要训练**的方法会调整模型的权重，而**无需训练**的方法则会引导预训练模型的推理

方面	需要训练	无需训练
核心原理	调整参数	引导推理
主要成本	数据整理 训练计算	提示词工作量 借助额外上下文进行推理
性能	高（专用）	各不相同（通用化）
灵活性	低（任务锁定）	高（可适应）
知识来源	内部（权重）	外部（上下文）
典型示例	• 领域预训练 LLMCompiler（Cummins 等，2025） • 监督微调 VirtualCompiler（Gao 等，2024年） • 强化学习 CUDA-L1（Li 等人，2025年）	• 提示工程 LEGOCompiler（Zhang 等人，2025） • 检索增强生成 CoCoGen（Bi 等人，2024年） • 智能体工作流 Compiler-R1（Pan 等人，2025年）

表 1：大语言模型方法论的定性比较 方法

过程。两者之间的选择，涉及数据整理与训练成本，以及提示工程和推理时系统复杂度之间的根本权衡。

3.4.1 需要训练：调整模型权重

这些方法会修改大语言模型的参数，以注入领域专属知识并使其行为专业化。当任务高度专业化，或目标表示（如编译器中间表示）在模型原始预训练数据中较为稀缺时，通常就有必要采用这些方法。

•**领域特定预训练：** 这是资源消耗最高的方法，即从头训练模型，或使用大规模领域专属语料进行持续预训练。对于处理通用模型从未见过的表示的任务而言，这种方法必不可少。例如，**Meta LLMCompiler**（Cummins 等，2025) 创建了专门针对 LLVM 中间表示进行预训练的基础模型，而 Zhai等人 (2024年) 则在 TVM 的大规模搜索空间语料上训练了其**TLM** 模型，使其充当选择器。**TransCoder**系列（Roziere 等人（2020）；anne Lachaux 等人（2021）； Roziere 等人（2022）； Szafraniec 等人（2023））也采用这种方法，在着手翻译之前学习跨语言表示。

•**监督微调（SFT）：** 这是最常见的适配方法。它以通用的预训练基础模型（例如 CodeLlama、GPT）为起点，并使用针对特定任务整理的“输入—输出”示例数据集对其进行微调。这种适配可以通过全量微调实现，也可以更常见地采用 LoRA 等参数高效微调（PEFT）技术。这种方法已在各种任务中证明了其卓越效果，例如针对 C 到 x86 神经编译（Zhang 等，2024）、Verilog 生成（Thakur 等，2024）和源代码级优化（Xu 等人，2025）对模型进行微调。**Virtu-alCompiler** (Gao 等，2024) 也利用 SFT，通过匹配语义将源代码转换为汇编语言。**CUDA-L1** 研究（ Li 等人，2025 年）也从 SFT 入手，教会模型 CUDA 优化的基础知识。

• **强化学习（RL）与基于反馈的调优：** 当任务具有明确且不可微的成功指标（例如性能加速、代码大小缩减或通过测试套件）时，可以使用强化学习或其他反馈机制，直接针对该目标优化模型。这通常在监督微调之后进行。例如，**PerfRL** (Duan 等，2025)使用带有指标反馈的强化学习进行代码优化。同样，**VerilogLLM** (Wang 等人，2025)利用来自测试平台的反馈，**CoTran** (Jana 等人（2024）)利用编译器和符号执行反馈，而**CUDA-L1** (Li 等人，2025年)采用 GRPO（Shao 等人，2024年），以超越其初始监督微调性能。**DeCOS**(Cui 等，2025)还利用大语言模型“点燃”强化学习过程，以实现数据高效学习。

3.4.2 无需训练：引导模型推理

这些方法利用强大的通用基础模型已有的能力，而不改变其权重。重点从数据整理和训练转向设计复杂的推理时系统，为模型提供必要的上下文和引导。

• **提示工程：** 这涉及设计详细的提示，指示模型执行任务（零样本），或提供一些任务的上下文示例（少样本学习）。对于更复杂的推理任务，通常会进一步采用思维链（CoT）等多步提示策略。例如，**LEGOCom-piler** (Zhang 等人，2025)结合少样本学习和思维链提示来执行神经编译，**CodeOptCoT** (Xuet al. 2024)和**SBLLM**(Gao 等人，2025)都明确利用思维链，通过迫使模型“逐步思考”来改进代码优化。类似地，**RTLLM**(Lu 等人，2024)采用“自我规划”提示策略，而**BuiltRome**(Nakkab 等人，2024)也发现，分层提示

结构对于基于大语言模型的硬件设计至关重要。

•**检索增强生成（RAG）：**为应对幻觉问题，并向模型提供其训练过程中可能未接触过的专业知识（例如晦涩难懂的应用程序接口、特定项目的上下文或优化规则），可以使用检索增强生成（RAG）系统。该系统首先从外部知识库中检索相关文档，并将其添加到模型的上下文中。Autoiot 使用该方法获取背景知识，**Autoiot**（Shen 等，2025）使用该方法进行项目级上下文检索，**CoCoGen**（Bi 等人，2024）使用该方法进行项目级上下文检索，而**SBLLM**（Gao 等人，2025）使用该方法检索优化知识。

• **智能体与迭代工作流：**这种先进的方法论将大语言模型视为更大循环中的推理引擎。智能体可以进行规划、使用外部工具（如编译器、验证器或 SMT 求解器），并根据这些工具提供的反馈迭代改进其输出。这种方法是**Compiler-R1**（Pan 等人，2025年）和**CompileAgent**（Hu等人，2025）等系统的核心。它也是迭代修复循环背后的核心原理，例如**DecLLM**（Wong 等人，2025）和**面向问题**（Ye 等，2024）中的迭代修复循环，以及**WhiteFox**（Yang 等，2024）等多智能体框架所采用的循环。Qimeng-Xpiler（Dong 等，2025）还采用了基于蒙特卡洛树搜索的搜索，可将其视为引导式迭代生成的一种形式。

3.4.3 方法论比较

需要训练的方法与无需训练的方法之间的选择，体现了一项根本性的设计权衡，我们在表1中进行了定性总结。

需要训练的方法着重于**调整模型参数（权重）**，这会产生大量前期**成本**，用于数据整理和训练计算。不过，这种方法将领域知识内化，因此在专门或狭窄的任务上表现出色（例如，Meta LLMCompiler（Cummins 等，2025）专门针对 LLVM IR 代码尺寸优化）。其代价是灵活性较低，因为最终模型会被“锁定”在特定的训练任务上。

相反，无需训练的方法会**引导固定模型的推理**过程。这使得**成本**转移到提示工程和推理时资源上。这些方法灵活性较高，因为提示和工具可以快速调整。其性能更具通用性，且高度依赖基础模型的能力；知识则通过上下文在运行时外部化并注入（例如，Xu等人(2024年)、Gao 等人(2025年)、Hu等人(2025年)）。

在实践中，这些方法论并非互斥。常见的模式是先采用需要训练的方法（如监督微调）创建专用模型，再将其部署于无需训练的方法论中（如结合检索增强生成的智能体工作流）。这种组合既能利用模型的专门知识，又能通过实时上下文和验证为其输出提供依据。

最后，结合分析过的这两个维度，我们现在可以全面回答第一个研究问题（**RQ1**），如下所述。

> **RQ1**：大语言模型如何融入编译过程？
>
> **答案**：
> 在任务方面，大语言模型通过三种主要的设计理念融入编译系统，这些理念定义了其角色：(1) 作为**选择器**，从预定义的编译器操作中进行选择（§§ 3.1）；(2) 作为直接的**翻译器**，重写代码（§§ 3.2）；或 (3) 作为**生成器**，创建新的编译器工具和组件（§§ 3.3）。
>
> 在大语言模型方面，这些角色通过两种主要的**大语言模型方法**得以实现：(1) **需要训练**的方法（例如微调、强化学习），通过调整模型权重来适应专门任务；以及 (2) **无需训练**的方法（例如提示工程、检索增强生成、智能体工作流），用于引导通用模型的推理。

4 维度 3&4：代码抽象层级与任务类型

本节详细介绍分类体系的第三和第四个维度：代码抽象层级和具体的任务类型。我们将这两个轴结合起来分析，因为它们密切交织；程序表示的层级从根本上决定了能够执行的任务性质。如图 5所示，我们将这些表示分为三个主要层次：

• 自然语言（NL），人类意图的最高层级；

•高级编程语言（PL），开发者编写的源代码；

•低级中间表示和汇编语言（ASM），以机器为中心的表示形式，由编译器后端使用并由计算机执行。

抽象层级的选择从根本上决定了可执行任务的性质，这也构成了我们分类法的第三个维度。本节将首先详细介绍在单一抽象层级内进行的任务（层内转换），然后讨论连接不同层级的更复杂任务（跨层级转换）。

4.1 层内转换

层内转换是指处理过程的输入和输出保持在同一抽象层级的转换。这些任务通常侧重于在给定表示形式内进行细化、优化或迁移。

首字母缩略词	引文	任务	代码级别
Qimeng-Xpiler	Dong等人(2025年)	转译	VNNI、CUDA、HIP、BANG
G-TransEval	Jiao 等(2023年)	转译	C语言++、Java
CoTran	Jana等人(2024年)	转译	Java、Python
Oxidizer	Zhang等人(2025年)	转译	Go-Rust
OpenCLGen	Palkowski 和 Gruzewski (2024年)	转译	PolyC-OpenCL
LLMLift	Bhatia 等人(2024年)	转译	C语言,C语言++,Java领域特定语言
Rectifier	Yin 等人(2024年)	转译	C语言++、Java、Python
AlphaTrans	Ibrahimzada 等人（2025年）	转译	Java-Python
输出格式	Macedo等人(2024年)	转译	C语言,C语言++,Go,Java,Python
LostInTranslation	Pan等人(2024年)	转译	C语言、C语言++、Go、Java、Python
KnowTransfer	Cassano 等（2024年）	转译	Java、Python、JS-Ocaml、Racket
CanLLMParallel	Nichols 等人（2024年）	转译	C语言++-MPI+OpenMP
SALLM	Siddiq 等人（2024年）	转译	Python+测试
TransCoder	Roziere 等人(2020年)	转译	C语言++,Java,Python
DOBF	Anne Lachaux 等人(2021年)	转译	C语言++,Java,Python
TransCoder-ST	Roziere 等人(2022年)	转译	C语言++、Java、Python
TransCoder-IR	Szafraniec 等人（2023年）	转译	LLVM IR-C++、Java、Rust、Go
BabelTower	Wen 等人(2024年)	转译	C-CUDA
CodeRosseta	TehraniJamsaz 等人(2024年)	转译	C-CUDA
HPCTrans	Lv 等人（2025年）	数据集生成器	C-CUDA
UnsuperBinTrans	Ahmad 和 Luo (2023年)	二进制转译	ARM-x86
TFix	Berabi 等(2021年)	代码修复	JavaScript
LLMAPR	Xia等人（2023年）	代码修复	Java、Python、C语言
ChatGPTRepair	Zhang等人（2023年）	代码修复	Java
EISP	Chen 等人(2024年)	代码修复	Python，JavaScript
MacroConfig	Albuquerque 等人(2024年)	代码修复	C语言/C++/Java
CoCoGen	Bi 等人(2024年)	代码修复	Python
ZS4C	Kabir 等人(2025年)	代码修复	Python
Repilot	Wei 等人 (2023年)	代码修复	Java
RustAssistant	Deligiannis 等人(2024年)	代码修复	Rust
LIBRO	Kang 等人(2023年)	通用漏洞披露 测试生成	Defects4J
SecurityTestGen	Zhang等人(2023年)	CVE 测试生成	Java CVE 测试
GeneticImprove	Brownlee 等人(2023年)	编译器模糊测试	Java
BenchDirect	Tsimpourlas 等人 (2023年)	编译器模糊测试	OpenCL
WhiteFox	Yang 等人 (2024年)	编译器模糊测试	PT-Inductor/XLA/TF-Lite
MetaMut	Ou 等人(2024年)	编译器模糊测试	C语言/C++
ClozeMaster	Gao 等人（2025年）	编译器模糊测试	Rust
FMCSO	Italiano 和 Cummins（2025年）	编译器模糊测试	C语言/C语言++
CORL	Mammadli 等人 （2020年）	编译器遍优化	LLVM 遍
GraphFlagOpt	Sajjadinasab 等人(2024年)	编译器遍优化	GCC 遍
神经进化	Heckel (2023年)	编译器遍优化	LLVM 遍
CompilerR1	Pan等人(2025年)	编译器遍优化	LLVM 遍
TLM	Zhai等人(2024年)	自动调优优化	TVM 中间表示
ReasoningCompiler	Tang等人 (2025年)	自动调优优化	TVM 中间表示
Effi-Learner	Huang 等人（2024年）	代码优化	Python
OMPar	Kadosh 等（2024年）	代码优化	C语言/C语言++ + OpenMP
ProblemOriented	Ye 等 （2024年）	代码优化	C++
CodeOptCoT	Xu等人(2024年)	代码优化	Python
LangProp	Ishida等人(2024年)	代码优化	Python
AutoComp	Hong 等人(2025年)	代码优化	C语言-C语言+内置函数
SBLLM	Gao 等人（2025年）	代码优化	Python、C语言++
PCAOT	Romero Rosas 等人 （2025年）	代码优化	C语言+OpenMP
CompilerGPT	Pirkelbauer 和 Liao (2025年)	代码优化	C语言/C语言++
Perfcodegen	Peng 等人(2025年)	代码优化	Python
SpeedGen	Purschke等人(2025年)	代码优化	Python
CodeOPT	Xu等人(2025年)	代码优化	C语言/C语言++
PerfRL	Duan等人(2025年)	代码优化	C语言++、Java、Python
ECO	Lin等人(2025年)	代码优化	C++
CUDA-L1	Li等人（2025年）	代码优化	CUDA/PyTorch
LLMVectorizer	Taneja等人 （2025年）	代码优化	C语言/C++
RACL	Wang 等人 (2025年)	代码优化	C语言/C++
CodeTransform	Cummins 等（2024年）	代码优化	Python

表 2：用于同层级代码转换的大语言模型总结

首字母缩略词	引文	任务	代码级别
NatGen	Chakraborty 等人(2022年)	代码优化	Java、Python
CodeOptEdu	Rong 等人 (2025年)	代码优化	Python
RTLrewriter	Bhatia 等人 (2024年)	代码优化	RTL
SymRTLO	Wang 等人 (2025年)	代码优化	RTL
InstCombiner	Mannarswamy 和 Das （2022年）	汇编优化	arm
窥孔优化	Fang 和 Mukhanov (2024年)	汇编语言优化	arm
VeriLOCC	Jin 等人(2025年)	汇编语言优化	SASS，RDNA
CompilerDream	Deng 等（2024年）	中间表示优化	LLVM 中间表示
MetaLLMCompiler	Cummins 等（2025年）	中间表示优化	LLVM 中间表示
优先级采样	Grubisic等人 (2024年)	中间表示优化	LLVM 中间表示
DeCOS	Cui等人(2025年)	中间表示优化	LLVM 中间表示

表 2：续

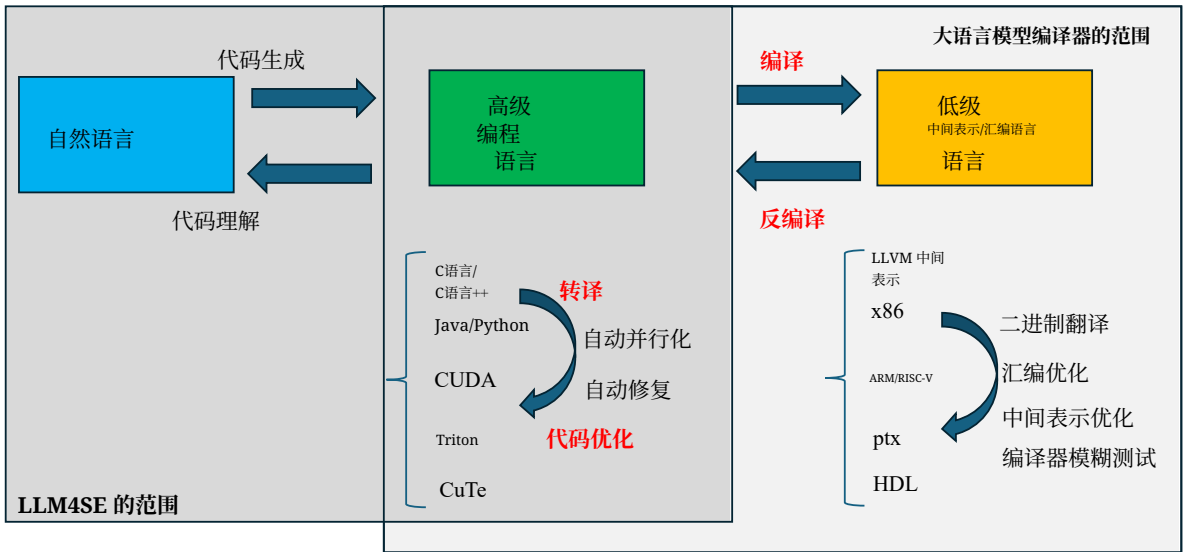

图 5：大语言模型编译器的代码级别视图

•**高级编程语言（编程语言）转换：**目前这是最活跃的研究领域，重点关注源到源转换，也是软件工程、编程语言和大语言模型社区共同关注的热点。大语言模型被应用于各种编程语言，从 C/C++、Java 和 Python 等通用语言，到用于并行计算的 CUDA 、Triton（Tillet 等，2019）以及 CuTe（NVIDIACorporation，2025年）。这一层级的关键任务包括：

– 转译：在不同编程语言之间进行转换，这项任务对于迁移遗留代码库或在不同编程生态系统之间转换程序至关重要。– 代码修复：生产环境中的代码设计以可维护性为一项主要目标，因此，大语言模型生成的代码应接受适当测试，而自动化测试生成或模糊测试是一个重要方向。此外，还应以最少的人工干预自动修复有缺陷的代码。

– 源代码优化：在不改变编程语言的前提下，重写编程语言代码，以提升其质量、可读性或性能。

• **低级中间表示/汇编转换：**在这一层级，大语言模型对面向机器的表示执行转换，这些表示通常对原始程序员不可见，但对最终性能至关重要。这些转换作用于常见的编译器中间表示，例如 LLVM 中间表示，以及 x86、PTX、Arm 和 RISC-V 等指令集架构。主要任务包括：

– 中间表示优化：直接对中间表示应用优化遍，以提高生成代码的效率。– 汇编优化：应用寄存器分配、指令调度和窥孔优化等汇编层级优化，以提升代码性能或缩减代码体积。– 二进制翻译：将低级代码从一种指令集架构（ISA）翻译为另一种，例如从 x86 翻译为 Arm。

如表 2所总结，将大语言模型应用于同层级代码转换已成为一个非常活跃的研究领域。这些任务在同一抽象层级内运行（例如源代码到源代码），主要利用大语言模型的生成能力来重写程序。本节根据研究的主要目标对这些研究进行分类和综述：代码转译，侧重于不同语言之间的迁移；代码修复，旨在纠正缺陷；以及代码优化，旨在提升程序性能。

4.1.1 转译

代码转译，即源到源翻译，是该领域最重要的任务之一。其核心挑战在于准确地将程序从源语言翻译为目标语言，同时保留其语义正确性和功能。此能力对于现代化遗留代码库、提升系统间的互操作性，以及将应用迁移到新的硬件生态系统至关重要，例如将顺序执行的 C 代码转换为适用于 GPU 的并行 CUDA 代码。以下研究展示了这项任务的广度，涵盖了多种语言对和编程范式。

OpenCLGen（Palkowski 和 Gruzewski，2024年）使用大语言模型将多面体 C 内核转译为 OpenCL。Rectifier（Yin 等人，2024年）引入了一种大语言模型纠错器，用于处理不同的转译错误，从而提高代码转译的准确性。OutputFormat（Macedo 等，2024年）研究了大语言模型输出格式对代码翻译任务的影响。LostInTranslation（Pan 等人，2024年）通过实证研究，将大语言模型转译过程中引入的程序错误按多个编程语言的详细类别进行归纳。SALLM（Siddiq 等人（2024年））通过考察与通用弱点枚举（CWE）相关的测试生成，对大语言模型生成安全 Python 代码的能力进行基准测试。

除了直接使用大语言模型，早期研究还聚焦于一个以数据为中心的问题：如何获取并行代码语料库来训练模型？为了解决这个问题，TransCoder 系列（Roziere 等人（2020）；Anne Lachaux 等人（2021）；Roziere 等人（2022）；Szafraniec 等人（2023））确立了无监督学习转译器语言模型的方法：（1）先使用掩码语言建模（MLM）学习跨语言；（2）再使用回译（BT）学习翻译规则。其他技术也逐步提升了所学转译器的能力，例如去噪自编码（Roziere 等人（2020））、反混淆（Anne Lachaux 等人（2021））、单元测试筛选（Roziere 等人（2022））以及利用编译器表示（Szafraniec 等人（2023））。这些研究通常涉及 C语言++、Java 和 Python 等通用编程语言。

除了在通用编程语言之间进行转译外，BabelTower（Wen 等，2022）提出了一种改进的、基于 TransCoder 的方法，以无监督方式学习 C++与 CUDA 之间的转译器，并采用了新的并行化指标。CodeRosseta（TehraniJamsaz 等，2024）进一步改进了 BabelTower，加入了 AST 实体识别和定制化去噪自动编码。如今，这些基于学习的方法更多地被用作数据集生成器，以支持更强大的大语言模型训练。除了基于学习的方法，HPCTrans（Lv 等，2025）修改了人工智能编译器 TVM（Chen 等，2018），使其无需依赖 TVM 即可自动生成语义等价的 CUDA 和 C 代码，从而创建丰富的 CUDA 到 CPU 转译语料库。

CanLLMParallel（Nichols 等，2024）通过使用大语言模型将顺序执行的 C语言++ 代码转换为 MPI、OpenMP 和 CUDA 等并行代码，研究大语言模型的自动并行能力。

CoTran（Jana etal. 2024）提出了一种基于强化学习的反馈机制，利用编译器反馈和符号执行来提高 Python 到 Java 转换中的编译准确率和功能准确率。

AlphaTrans（Ibrahimzada 等人，2025）提出了一种基于大语言模型、集成于 GraalVM 的项目级转译器，以支持部分转译验证；他们还实现了一种分治策略，将源代码和测试拆分，以实现可扩展的 Java 到 Python 转译。Oxidizer（Zhang 等人，2025）也提出了一种面向整个项目、基于大语言模型的 Go 到 Rust 的转译器，其中项目划分模块通过分而治之降低翻译复杂度，类型检查特性映射模块则负责在不同语言之间迁移语义。

KnowTransfer（Cassano 等，2024）提出，可通过将高资源语言（如 Java、Python）中的确定性测试用例转译到低资源语言（如 OCaml、Racket），改进并验证转译，从而有效构建丰富的双语语料库。

LLMLift（Bhatia 等，2024）提出，可以使用大语言模型将顺序程序转译为张量处理框架的领域专用语言（DSL），例如 PyTorch 和 NumPy，并使用 Floyd-Hoare 逻辑（FHL）验证生成的程序。

Qimeng-Xpiler（Dong 等，2025）使用大语言模型在不同架构之间转译并行代码，包括 SIMT GPU（CUDA 和 HIP）、SIMD CPU（C语言+VNNI）和 SIMD NPU（BANG C）；使用 SMT 求解器检查关键转译结果的正确性，并使用蒙特卡洛树搜索（MCTS）在目标架构上探索性能更高的候选方案。

除了源代码转译之外，还有一些有趣的研究，例如 UnsuperBinTrans（Ahmad 和 Luo，2023），它通过将高资源指令集架构（例如 x86）上的二进制代码转换到低资源指令集架构，初步研究漏洞发现问题。然而，由于代码方面的困难，许多二进制层面的工作无法直接通过神经网络进行转译；相反，许多研究关注其他实用任务，例如二进制相似性检测或编译器属性识别，我们将在 §§ 4.3 中详细介绍。

4.1.2 代码修复

除了在不同语言之间进行翻译，另一项关键应用是将程序

从有缺陷的状态转换为正确状态。利用大语言模型进行自动程序修复（APR），将错误修复视为一项专门的翻译任务：把错误的源程序转换为同一语言中的修正版。这种方法旨在利用模型对常见编程错误及其相应修复方案的学习知识，自动完成繁琐且容易出错的调试过程。本文综述的研究展示了利用大语言模型自动识别和修复程序错误的多种策略。此外，生成用于检测程序错误的专门测试（模糊测试）也与此类别密切相关，可视为同层级代码转换（合成测试）任务或基于效用的任务。

TFix（Berabi 等人，2021年）通过训练 Transformer 模型来执行专门的序列到序列代码修复任务。LLMAPR（Xia 等，2023年）首次对直接将大语言模型应用于自动程序修复进行了全面研究，指出增加样本量并结合修复模板信息，都有助于提升基于大语言模型的自动程序修复效果。ChatGPTRepair（Zhang 等，2023年）揭示了自动程序修复（APR）中一个重要但被忽视的数据泄漏问题，并发现，在其提出的去污染基准测试 EvalGPTFix 上，ChatGPT 在自动程序修复方面的能力超过 PLBart 和 CodeT5。EISP（Chen 等，2024年）提出了一种无需测试、基于大语言模型静态分析的语义错误定位框架。MacroConfig（Albuquerque 等，2024年）研究了大语言模型利用编译反馈解决可配置宏错误的能力。CoCoGen（Bi 等人，2024年）通过迭代生成与验证流程，并谨慎检索上下文，开展项目级代码修复。ZS4C（Kabir 等，2025年）提出了一种基于大语言模型的零样本合成器，可将不完整代码补全为可编译代码。

除了使用大语言模型进行自动化代码修复外，大语言模型还可以作为修复助手使用。Repi-Wei 等，2023）通过在修复过程中积极合成有效补丁，将大语言模型用作程序修复的补全引擎。RustAssistant（Deli- giannis 等，2024）是另一种助手工具，可根据编译器反馈提出潜在的 Rust 代码修复方案。

在模糊测试方面，LIBRO（Kang 等，2023）展示了如何利用大语言模型通过根据程序错误报告合成测试程序来复现程序错误。SecurityTest- Gen（Zhang 等，2023）还研究了大语言模型根据通用漏洞披露生成安全测试的能力。

GeneticImprove（Brownlee 等，2023）发现，大语言模型可作为变异算子，结合遗传改进来生成多样化的有效程序。BenchDirect（Tsimpourlas 等，2023）训练语言模型，以生成可读性高的编译器测试基准。WhiteFox（Yang 等，2024）采用多智能体框架，自动合成白盒编译器模糊测试。MetaMut（Ou 等，2024）提出以大语言模型作为变异器进行引导：它利用大语言模型填充精心设计的模板，生成非平凡的变异器设计，然后将其随机应用于测试程序，合成模糊测试程序，并成功发现了 131 个 GCC/Clang 编译器缺陷。ClozeMaster（Gao 等人，2025）同样使用大语言模型对 Rust 编译器进行模糊测试：先生成带有完形填空掩码的代码片段，再填补空白，生成多样化的编译器测试程序。FMCSO（Italiano andCummins 2025）提出了一种变异测试方法，利用大语言模型迭代修改初始代码种子，进而制定差分测试策略，以发现 LLVM 编译器中遗漏的代码大小优化。

4.1.3 代码优化

第三大类的同层级转换是代码优化。这可以理解为将程序从语义正确但次优的版本转换为在性能、内存使用或代码大小方面更高效的改进版本。虽然这是编译器的经典目标，但大语言模型在这一领域的应用尤其广泛。与主要处理源代码的转译和修复不同，优化任务正处于多个抽象层级的探索之中。这不仅包括高层次的源到源重写，还包括中间表示优化和汇编语言级调优等关键的低层次任务。在所有这些层级上，大语言模型都提供了一种新颖的、数据驱动的方法，用于发现复杂的优化策略，而传统的基于启发式方法的编译器可能难以识别这些策略。

首先，与以往机器学习驱动的编译器研究类似，将语言模型用作选择器也是可行的，并且可以通过适当的特征工程进行学习。CoRL（Mammadli 等人，2020年）、神经进化（Heckel，2023年）、GraphFlagOpt（Sajjadinasab 等人，2024年）和 CompilerR1（Pan 等人，2025年）被用作选择器，以优化编译器优化遍次排序。TLM（Zhai 等人，2024年）使用从头训练的大语言模型，从庞大的调度空间中生成预测的调度序列，从而加速 TVM 自动调优搜索。ReasoningCompiler（Tang 等人，2025年）同样使用蒙特卡洛树搜索，通过大语言模型推理探索搜索空间，并将其集成到 TVM 中，以提高采样效率。

除了将大语言模型用作选择器外，还有更多研究在探索利用大语言模型执行生成式优化，将其作为直接翻译器；人们认为，大语言模型中丰富的预训练代码知识能够引导其执行合理的优化。这些优化大多发生在源代码级别。

Effi-Learner（Huang 等人，2024年）提出了一种自优化框架，利用执行开销配置文件提升大语言模型生成代码的效率。OMPar（Kadosh 等人，2024年）研究了 OpenMP 编译指示生成以自动并行化 C语言/C++ 代码的能力。ProblemOriented（Ye 等，2024年）提出了一种锚点验证机制：首先基于慢代码合成测试输入，通过执行慢代码构建经过验证的测试用例，然后利用执行反馈迭代改进优化后的代码。CodeOptCoT（Xu 等人，2024）应用思维链技术来增强结构

理解和自检，以改进代码优化。LangProp（Ishida 等人，2024）专注于在 CARLA 中自动驾驶代码优化。

2025年，与此任务相关的研究迅速涌现。Auto-Comp（Hong 等人，2025）研究使用大语言模型将 C 代码优化为适用于不同张量加速器的代码，并采用各自不同的内在函数。SBLLM（Gao 等人，2025）提出了一种基于搜索的大语言模型框架，结合优化知识检索和受遗传算子启发的思维链提示。PCAOT（Romero Rosas 等人，2025）将基于大语言模型的优化与传统编译器优化进行比较，发现大语言模型难以直接优化大型程序（以代码行数衡量）。CompilerGPT（Pirkelbauer 和 Liao, 2025）提出使用大语言模型分析并处理编译器优化报告，根据优化报告进行定制化代码重写优化。

Perfcodegen（Peng 等人，2025）证明，与仅使用朴素提示的旗舰大语言模型相比，配备适当执行反馈的小型大语言模型能够生成性能更高的代码。SpeedGen（Purschke 等人，2025）使用大语言模型通过重写 Python 代码，使其采用 PyTorch 和 NumPy 等高性能库，从而优化 Python 代码的性能。CodeOPT（Xu 等人，2025）微调了一个 LoRA 适配器，以便在 C/C++源代码级别执行循环展开、内联展开、常量折叠和死代码消除等优化策略。PerfRL（Duan 等人，2025）在强化学习环境中训练一个小型语言模型，通过直接的指标反馈获得更好的代码优化结果。

ECO（Lin等人，2025年）利用历史提交记录已解决的反模式，并使用经过微调的大语言模型自动重构代码、自动验证并提交代码审查，从而构建了一条自动化代码优化流水线，可扩展至谷歌生产环境中的仓库级计算机。CUDA-L1（Li 等人，2025年）展现了强化学习在自主学习 CUDA 优化方面的卓越能力；采用监督微调+GRPO（Shaoet al. 2024）微调的模型能够生成 250 个 KernelBench（Ouyang 等人，2025年）案例中的 249 个，并将其中 240 个优化得优于原生 PyTorch。LLMVectorizer（Taneja 等，2025）研究如何在源代码级别使用单指令多数据内在函数自动向量化 C 代码，并通过 Alive2（Lopes 等，2021）进行符号验证，以实现形式化验证。RACL（Wang 等人，2025）利用归约分析分而治之，降低程序复杂度并执行以输入为中心的代码优化，从而显著减少性能分析工作。

CodeTransform（Cummins 等，2024年）探索性地尝试采用 LLM 作为生成器方法，让大语言模型编写转换代码，而不是直接转换代码。

除了纯粹的性能优化之外，还有一些工作，例如 NatGen（Chakraborty 等，2022），提出了“代码自然化”这一新任务，将不自然的代码转换为符合编程范式的自然代码，并预训练了一个语言模型，使其能够很好地对 Java 和 Python 代码进行自然化。其他工作，如 CodeOptEdu（Rong 等，2025），则用于出于教育目的改进代码。

4.2 跨层级转换

跨层级转换是指连接不同抽象层次的转换。这些任务是编译器及其相关工具定义的基础，也是该领域中大语言模型最复杂且影响最深远的应用之一。正如表 3所述，我们将在下一节列出这些跨层级转换。

4.2.1 代码生成：自然语言-编程语言

这是将自然语言规范转化为结构化编程语言程序的过程。在这一过程中，大语言模型充当终极的编译器前端，直接将开发者对问题的描述转化为可运行的解决方案。

ANPL（Huang 等，2023）提出了一种交互式方法，将程序生成问题分解为程序草图和待填充部分，之后再填充这些部分，并允许用户灵活地随时介入。

AIOSCompiler（Xu 等，2024年）将大语言模型用作自然语言编程和人工智能代理流程编程的解释器，并引入领域专用语言 CoRE，以统一自然语言编程、伪代码编程和流程编程。Unicoder（Sun 等，2024）也提出了一种代码表示，用于统一不同的编程语言，并使其适用于大语言模型进行代码生成。

Autoiot（Shen 等，2025年）聚焦于利用自然语言编程生成 AIoT 应用代码，并借助背景知识检索模块、思维链程序合成模块和自动代码改进模块提供支持。

ArabicLLMCompiler（Sibaee 等，2024年）将大语言模型用作解释器，把基于阿拉伯语的编程语言转换为可执行的 Python 代码。

代码生成的逆向任务——代码理解——也是软件工程中的一项重要任务。然而，由于我们的研究以编译器为中心，因此本综述将以代码理解为目标的研究视为范围之外的内容。

4.2.2 编译：编程语言—汇编语言

这是经典编译任务的核心：使用大语言模型将人类可读的编程语言转换为中间表示或汇编语言等低层表示。这是一项核心挑战，目前人们正在探索如何利用大语言模型增强传统编译器后端的组件，甚至取而代之。一些类似的子任务，例如高层次综合和硬件描述语言生成，也属于这一类别。

Transformer-x86（Armengol-Estape 和 O’Boyle, 2021年）初步研究了如何从编译器生成的 C-x86 语料库中学习 Transformer。Transformer- llvm（Guo 和 Moses，2022年）同样研究了设置优化级别后的 C-LLVM 和 C-x86。二者都未能达到良好的翻译准确率；对于优化后的设置，翻译准确率通常为 0%。

VirtualCompiler（Gao 等，2024）将大语言模型编译建模为与汇编代码搜索类似的任务，并

首字母缩略词	引文	任务	代码级别
ANPL	Huang 等人（2023年）	代码生成	NL-Python
AIOSCompiler	Xu等人(2024年)	代码生成	NL-CoRE DSL
Autoiot	Shen 等人（2025年）	代码生成	自然语言-AIoT 代码
ArabicLLMCompiler	Sibaee 等人（2024年）	代码生成	阿拉伯语-Python
Unicoder	Sun 等人(2024年)	代码生成	伪代码-Python、JS
transformer-x86	Armengol-Estape 和 O'Boyle (2021年)	编译	C-x86
Transformer-llvm	Guo 和 Moses(2022年)	编译	C-LLVM IR
llm-x86	Zhang等人 (2024年)	编译	C-x86
LEGO-Compiler	Zhang等人 (2025年)	编译	C-x86、Arm、RISC-V
VirtualCompiler	Gao 等人(2024年)	编译	C-x86
VeriLOCC	Jin 等（2025年）	编译	MIR-SASS，RDNA
InstCombiner	Mannarswamy 和 Das (2022年)	编译	arm
VEGA	Zhong 等 (2025年)	编译	LLVM C语言++、TableGen
ComBack	Zhong 等（2024年）	编译	LLVM C语言++、TableGen
AutoChip	Thakur 等（2023年）	RTL 生成	Verilog
Origen	Cui等人(2024年)	RTL 生成	Verilog
VeriGen	Thakur 等人（2024年）	RTL 生成	Verilog
VerilogLLM	Wang 等人 （2025年）	RTL 生成	Verilog
MakeMoveCount	DeLorenzo 等人(2024年)	RTL 生成	Verilog
RTLLM	Lu 等人（2024年）	RTL 生成	Verilog
BuiltRome	Nakkab 等人（2024年）	RTL 生成	Verilog
HLSPilot	Xiong 等人（2024年）	HLS	C语言++高层次综合 C语言++
Slade	Armengol-Estape 等人（2024年）	反编译	x86、Arm-C
LLM4Decompile	Tan 等人 (2024年)	反编译	（混淆）x86-C
Degpt	Hu等人(2023年)	反编译	x86-C
Lmpa	Xu等人(2023年)	反编译	x86-C
BoostDecompile	Cao 等人（2022年）	反编译	x86-C
BinSum	Jin 等（2023年）	反编译	x86-NL
DecLLM	Wong 等（2025年）	反编译	x86-C
IR-LLM	Jiang 等（2025年）	中间表示反编译	LLVM IR-C
Forklift	Armengol-Estape 等人（2024年）	反编译为中间表示	x86、Arm、RISC-V-LLVM 中间表示
NeuralShapeCompiler	Luo 等人（2023年）	多模态	文本-点云-代码
RASP	Weiss 等人（2023年）	神经编译	RASP领域专用语言
Tracr	Lindner 等人（2023年）	神经编译	RASP Transformer 权重
ALTA	Shaw 等人（2025年）	神经编译	ALTA-transformer 权重
TransformerProgram	Friedman 等人(2023年)	NeuralDecompilation	transformer weight-RASP
AlgorithmicLM	Saldyt 和 Kambhampati(2025年)	NeuralDecompilation	transformer weight-python

表 3：用于跨层级代码转换的大语言模型总结

提出使用大语言模型，将任意语言的任意源代码编译为汇编代码，方法是匹配汇编结果。

Llm-x86（Zhang 等，2024年）采用以数据为中心的增强流水线来提升编译器生成语料的质量，重点关注数值表示和标签生成，并使用改进后的语料对大语言模型进行微调，使 C-x86 编译性能得到显著提升，超过 91%。

LEGO-Compiler（Zhang 等人，2025）研究了神经编译的可扩展性问题，将函数级代码拆分为更细的粒度，例如基本块级或语句级，并通过管理上下文，为每个片段提供必要的符号信息并进行相应编译。结合可验证中间结果的逐步工作流程和自动修复循环，该方法在 O0 设置下实现了跨三种架构超过 99% 的神经编译准确率，并在不推进模型的情况下将可扩展性提升了一个数量级。然而，基于大语言模型的编译仍处于初步阶段，因为其表现尚未超越现有编译器系统。

除了由大语言模型执行完整编译过程之外，也有一些工作研究编译流水线中的特定步骤。VeriLOCC（Jin 等人，2025年）研究了大语言模型在 CUDA SASS 和 AMD RDNA 汇编代码生成中的寄存器分配能力。InstCombiner（Mannarswamy 和 Das，2022）则研究了大语言模型在 ARM64 汇编中的指令组合能力。

除了 LLM 作为翻译器方法外，还有一些有趣的研究致力于生成特定的编译器代码。VEGA（Zhong 等人，2025年）提出了一种使用预训练语言模型生成 LLVM 后端的方法，并采用精心构建的编译器后端数据集 Com-Back（Zhong 等人，2024年）。该模型将 LLVM 代码视为需要学习的特征，并将不同后端作为训练数据，因此无需人工干预，便能够为超过 60% 的 LLVM 后端函数生成程序草图。然而，完全自动化的编译器代码生成距离现实仍然十分遥远，因为编译器是最复杂、最难维护的软件之一。

除了编译为汇编代码外，还有一些工作专注于寄存器传输级代码生成，这也可以

被视为更广义的编译器任务。其中，AutoChip（Thakur等人 2023年）提出了一个自我反思循环，用于修复琐碎的生成错误。Origen（Cui等人 2024年）提出了一个使用 Claude3 蒸馏数据的数据增强流程，以及类似的自我反思机制。VeriGen（Thakur等人 2024年）通过有监督微调，对一系列 Verilog 生成大语言模型进行微调。VerilogLLM（Wang 等人 2025年）进一步利用带有测试平台反馈的强化学习（RL）训练 Verilog 生成大语言模型。

MakeMoveCount（DeLorenzo 等人，2024年）研究了在 RTL 代码生成中执行 MCTS。RTLLM（Lu 等人，2024年）提出了首个基于大语言模型的 RTL 基准测试，以及一种简单而有效的自规划提示策略，以提升使用 GPT3.5 进行 RTL 代码生成的性能。BuiltRome（Nakkab 等人，2024年）还发现，分层提示结构可以显著提升大语言模型在硬件设计任务中的性能，从而成功生成原本不可能实现的复杂模块。

HLSPilot（Xiong 等人，2024年）不再专注于 Verilog RTL 生成，而是生成由软件 C++代码转换而来的高层次综合（HLS）代码。通过集成性能分析工具和设计空间探索工具，实现自动化的硬件/软件划分与 pragma 调优，其性能超过了手工编写的 FPGA 内核。

4.2.3 反编译：汇编语言-编程语言

这涉及编译的逆向过程，即大语言模型尝试从二进制文件或汇编文件等低层表示中重构出人类可读且具有语义意义的编程语言。这是一项众所周知的困难任务，而大语言模型的模式识别能力为其提供了一个颇具前景的研究方向。

Slade（Armengol-Estape 等人，2024）和 LLM4Decompile（Tan 等人，2024）都是用于反编译任务的学习型语言模型，各自侧重于优化代码恢复或混淆代码恢复。

Degpt（Hu等人，2024）侧重于利用大语言模型解读和改进反编译器输出，以提高其可读性和简洁性，从而辅助逆向工程过程。

Lmpa（Xu等人，2023）开发了一种程序分析辅助方法，用于在反编译过程中恢复符号名称。通过在受控上下文中向 ChatGPT 提问，该模型生成的恢复名称中，有 75% 被用户认为质量良好。

BoostDecompile（Cao 等人，2022年）采用多层反编译流程，通过规则和神经网络协同恢复程序。

BinSum（Jin 等人，2023年）关注大语言模型理解汇编代码的能力，并发现移除调试符号会显著降低反编译准确率。

DecLLM（Wong 等人，2025年）提出了一种以可重新编译为核心的反编译系统，并采用迭代式基于大语言模型的修复循环来改进反编译器的输出，并结合静态重新编译和动态运行时反馈。

除了完整的反编译过程之外，也有一些工作聚焦于其中的特定步骤。IR-LLM（Jiang 等人，2025）聚焦于 LLVM IR 到 C 的反编译过程，而 Forklift（Armengol-Estape 等人，2024）则聚焦于汇编语言到 LLVM IR 的反编译过程。

4.2.4 跨层级转换的特殊情况

除了传统编译栈中的转换之外，近年来还出现了一条研究方向，探索编译中的更多特殊情况，而不局限于典型的基于文本或符号的领域。

例如，NeuralShapeCompiler（Luo 等人，2023）提出了一个受编译器启发的统一框架，用于在多个模块之间转换数据。该框架首先将所有数据转换为统一的离散中间代码，然后使用 Transformer 模型执行跨文本、代码和点云的转换。

另一个独特且新兴的研究方向，也称为“神经编译”，探索将程序直接转换为神经网络的参数。这种方法并非旨在实现传统意义上的执行，而是为了理解 Transformer 等模型的算法能力，从而促进对大语言模型可解释性的研究。该领域的一条奠基性研究路线始于 RASP（Weiss 等人，2021），这是一种抽象编程语言，旨在使用与 Transformer 核心操作相对应的原语来表达算法。在这一理论框架的基础上，Tracr（Lindner 等人，2023）被开发为一种编译器，可将 RASP 程序直接转换为标准 Transformer 模型的权重。ALTA（Shaw 等人，2025）进一步扩展了这一能力，增加了对循环等动态控制流操作的支持。

作为对这一编译过程的补充，一些研究探索了反向路径——即一种反编译形式。例如，TransformerPro- gram（Friedman 等人，2023）提出了一种训练改造后的 Transformer 的方法，使其能够自动转换回离散、可读的人类 Python 程序。此外，AlgorithmicLM（Saldyt 和 Kambhampati，2025年）研究了通过增强型解释器组合这些“神经编译”权重、以执行组合算法的方法。

尽管这一研究领域在理论上十分重要，但目前仍面临显著局限：程序规模有限，可编译的操作类型也不够多样，因此仍是值得进一步研究的关键领域。

从涉及层内（表 2）和层间（表 3）代码转换的研究数量可以看出，处理高级代码的研究（59+34）多于处理低级中间表示/汇编语言的研究

（14+16）。这种数量上的失衡可以从两个方面来看。首先，底层编译器任务受到社区的关注较少，因为大语言模型研究人员的数量远多于编译器研究人员（后者对底层表示相关任务感兴趣）。其次，低级代码表示不太适用于大语言模型方法，原因如下：1. 在大多数大语言模型的预训练阶段，底层代码语料库少于高级语料库；2. 底层语料库的质量相对较低，因为许多低级代码表示由编译器自动生成，缺乏可读性，且难以让大语言模型学习。

4.3 非变换型实用任务

表4概述了相关的基于实用任务的研究。除生成任务外，大语言模型还支持编译器领域中更广泛的实用任务，例如我们前文概述的漏洞测试生成和编译器模糊测试。

至于其他实用任务研究，CompileAgent（Hu等人 2025年）提出使用大语言模型自动配置项目。DCC（Taylor等人 2024年）开发了一款工具，将大语言模型集成到 Debugging C Compiler 中，以生成具有上下文感知能力、面向初学者的解释，说明编译错误和运行时错误，从而帮助编程入门学生。Quantum（Ren等人 2024年）采用序列到序列模型解决量子编译中的量子比特路由问题；与启发式算法相比，该方法减少了新增量子门的数量和编译运行时间。ComPAT（Cai等人 2024年）推出了一款基于大语言模型的教学助手，用于编译原理课程。HPC-GPT（Ding等人 2023年）使用自动生成的领域专用数据对大语言模型进行微调，以提升其在高性能计算任务中的性能，例如数据竞争检测和人工智能模型管理。Fair（Niu等人 2024年）提出了一种用于中间表示的预训练模型，该模型采用新颖的流类型感知图输入、图 Transformer 架构和五项特定的预训练任务，以提升大语言模型理解中间表示的能力。

将深度学习模型融入二进制分析领域正日益受到关注。该领域对逆向工程、漏洞检测和软件供应链安全至关重要。相关研究主要聚焦于两项关键任务：二进制相似性检测和编译器溯源识别（CPI）。

在二进制相似性检测中，目标是判断两个二进制函数在语义上是否等价，尽管它们可能因不同的编译器或优化设置而有所差异。OSCAR 等奠基性工作（Peng 等人 2021年）确立了一种预训练范式，从 LLVM 中间表示中学习代码表示，并利用对比学习应对多种优化。后续方法进一步完善了这一概念。例如，JTrans（Wang 等人 2022年）引入了跳转感知Transformer模型，以更好地捕获控制流；而 DiEmph（Xu等人 2023年）则通过降低二进制代码中编译器引入的噪声的影响来提升鲁棒性。同样，OPTango（Geng 等人 2023年）利用多中心表示学习来创建一种能够抵御编译器优化复杂影响的二进制差分工具。

编译器来源识别（CPI）是其中一项专门应用，旨在识别用于生成给定二进制文件的编译器及其优化级别。研究人员为此探索了各种神经网络架构。例如，MuCPI（Gao 等，2024）通过结合注意力机制改进门控图神经网络（GGNN），从二进制文件的控制流图中学习特征。另一种较为非传统的方法是 ObfuscateCPI（Khan 等，2024），它将二进制文件转换为图像，并使用预训练的计算机视觉模型识别编译器的视觉指纹，即使经过混淆也能奏效，展现出较强的鲁棒性。

在本节末尾，我们终于可以根据近期关于大语言模型编译器的研究来回答**研究问题 2**。

> **研究问题 2**：大语言模型所处理的主要编译器相关任务是什么？
> **答案**：**生成任务**，例如代码生成、代码转译、代码修复、代码优化和反编译，如今已成为由大语言模型驱动的编译器所处理的主要任务。这些编译器借助更强大的大语言模型和更完善的系统设计取得了显著进展，并且如今已在一个或多个维度上超越传统方法。**实用任务**，例如系统自动配置、相似性检测和编译器模糊测试，也因大语言模型而得到显著改进。

然而，对于狭义定义的编译任务，尽管已有关于端到端编译过程（Armengol-Estape 和 O'Boyle, 2021；Zhang 等，2024；Gao 等，2024；Zhang 等人，2025）、部分编译过程（Jin 等人，2025年；Mannarswamy 和 Das，2022）以及中间表示优化过程（Deng 等，2024；Grubisic 等，2024；Cui 等，2025；Cummins 等，2025）的研究，但这些研究或多或少仍处于初步阶段，在性能、成本或可扩展性方面大多无法超越传统编译器。一些研究（Zhong 等人，2024，2025年）尝试从“敏捷开发”的角度解决编译器构建问题，但由于缺乏数据集，仍需要编译器专家的大量介入，并且无法推广到任意编译器开发任务。

尽管如此，随着该领域的研究逐年增多，相关技术也不断进步。我们认为，大语言模型驱动的编译器系统是一个很有前景的研究方向。

5 基准测试与最先进技术的发展

5.1 基准测试　　，指标与规模

评估这一领域的一个关键方面，是系统地分析基准测试本身。表 5概述了各类基准测试和

首字母缩略词	引文	任务	代码级别
LIBRO	Kang 等人（2023年）	CVE 测试生成	Defects4J
SecurityTestGen	Zhang等人(2023年)	CVE 测试生成	Java CVE 测试
GeneticImprove	Brownlee 等人（2023年）	编译器模糊测试	Java
BenchDirect	Tsimpourlas 等人（2023年）	编译器模糊测试	OpenCL
WhiteFox	Yang 等人(2024年)	编译器模糊测试	PT-Inductor/XLA/TF-Lite
MetaMut	Ou 等人(2024年)	编译器模糊测试	C/C++
ClozeMaster	Gao 等人(2025年)	编译器模糊测试	Rust
FMCSO	Italiano 和 Cummins（2025年）	编译器模糊测试	C语言/C语言++
CompileAgent	Hu等人（2025年）	自动配置	CMake/Make
DCC	Taylor等人（2024年）	错误解释	编译器日志
Quantum	Ren等人(2024年)	量子比特路由	量子程序
ComPAT	Cai等人(2024年)	课程助教	编译器教材
OSCAR	Peng 等人(2021年)	二进制检测	LLVM 中间表示
JTrans	Wang 等人(2022年)	二进制检测	汇编语言
DiEmph	Xu等人(2023年)	二进制检测	汇编语言
OPTango	Geng等人（2023年）	二进制检测	汇编语言
MulCPI	Gao 等人（2024年）	二进制检测	汇编语言
ObfuscateCPI	Khan 等人（2024年）	二进制检测	汇编语言
HPC-GPT	Ding等(2023年)	HPC知识预训练	HPC知识
Fair	Niu等(2024年)	IR理解	LLVM 中间表示

表4：编译器工具的大语言模型综述

基准测试/数据集	语言	主要任务	规模（单位）
CodeNet（Puri 等，2021）	多语言（55）	代码理解，转换处理	13.9M
AVATAR（Ahmad等，2023年）	Java、C语言++	代码转译	~9.5k
TransCoder-Test（Roziere 等人（2020））	C语言++、Java、Python	代码转译	1.4k
PIE（Madaan等，2023年）	Python、Java	代码转译、编辑	~77k
MiBench（Guthaus 等人，2001年）	C	嵌入式编译，优化	35
KernelBench（Ouyang 等人，2025年）	CUDA、PyTorch	GPU 内核生成 / 优化	250
TritonBench（Li 等人，2025年）	Triton领域专用语言	图形处理器内核生成 / 优化	184
AnghaBench（Da Silva 等，2021年）	C语言、汇编语言	神经编译	~1M
ExeBench（Armengol-Estap´e 等人，2022年）	C语言、汇编语言	神经编译	0.7M
HumanEval（Chen 等，2021a）	Python	代码生成	164
MBPP（Austin 等，2021年）	Python	代码生成	974
Defects4J（Just 等，2014）	Java	代码修复	854

表 5：大语言模型编译器任务常用基准测试/数据集汇总。“规模（单位）”列经过简化，用于显示核心实体的大致数量，以反映评估规模，例如主要问题、函数或代码样本的数量（百万级或千级）。

在各种大语言模型编译器任务中使用的数据集。如下所示，我们可以从三个反复出现的维度来分析这些基准测试：

基准测试构建通常遵循三种模式：（1）改编大型公开基准测试/数据集（例如 CodeNet、AVATAR），通常会采用新的筛选条件或元数据。我们在表 5中总结了这些常用的基准测试/数据集。

用于通用理解和转译，**CodeNet**（Puri 等，2021) 提供了一个大规模数据集，其中包含近 1400 万个代码样本，涵盖 55 种以上语言，并附有丰富的元数据。 **AVATAR**（Ahmad等，2023年) 在 CodeNet 的基础上，通过整理 9,515 个问题并推导出 3,391 对并行函数，用于细粒度对齐。

TransCoder-Test 提供了一个标准化的并行数据集，自 Roziere 等人(2020年)以来一直用于测试（948 个测试实例），并仍是评估翻译准确率的常用基准。 **PIE（性能改进编辑）**（Madaan等，2023年) 是一个性能优化基准测试，由约 7.7 万对提交组成（主要来自 CodeNet），并借助 gem5 模拟器评估运行时性能提升，因此成为面向优化的大语言模型广泛采用的测试平台。

对于编译和生成任务，**MiBench** （Guthaus 等人，2001年）提供了 35 个贴近实际领域的嵌入式 C 程序，用于对端到端编译进行压力测试。 **KernelBench** （Ouyang 等人，

2025年）专注于 GPU内核生成，包含 250 个任务，并提供正确性和加速比指标。 **Triton-Bench** （Li 等人，2025年）同样专注于基于 Triton 的 GPU 内核评估。对于神经编译，**Ang-haBench** （Da Silva 等，2021年）包含 100 万个可编译的 C 函数，而 **ExeBench** （Armengol-Estap´e 等人，2022年）将规模扩展至 68 万个可执行 C 函数，其中包含一个由 4 万个基于单元测试的可验证划分组成的拆分。对于代码生成，**HumanEval** （Chen 等，2021a）（164 个 Python 问题）是评估 *pass@k* 的事实标准，而 **MBPP** （Austin 等人，2021年）（974 个 Python 任务）则旨在衡量日常编程能力。

第二种模式（2）是精选真实的开源代码仓库，旨在体现端到端的真实情况。例如，**AlphaTrans**(Ibrahimzada 等人，2025) 对十个真实世界的代码仓库进行拆解，以评估代码仓库规模下的运行时行为。类似地，**Oxi-dizer** (Zhang 等人，2025) 精选了若干真实世界的 Go 项目，用于评估 Go 到 Rust 的转译。知名的 **Defects4J** (Just 等人，2014) 基准测试是从真实 Java 项目中精选出的可复现程序错误集合，也遵循这一模式，并成为许多程序修复研究的基础基准测试（Xia 等人，2023；Wei 等，2023）。

第三种模式（3）是合成测试用例，通过定制的下游任务探测能力。例如，**Qimeng-Xpiler** (Dong 等，2025) 构建了一个内核级张量程序套件，用于测试跨领域专用语言转译。这种模式也是**编译器模糊测试**的基础，在这类工作中，例如 **WhiteFox**(Yang 等，2024) 和 **MetaMut** (Ou 等，2024) 专门合成多样且非平凡的测试程序，以发现编译器缺陷。

在**粒度**层面，大多数基准测试仍以函数为中心，因此便于进行并排比较。例如，在表 2列出的 21 项转译相关任务中，只有少数但重要的任务转向整程序／代码仓库场景，例如**Oxidizer** (Zhang 等人，2025) 和 **Alpha-Trans** (Ibrahimzada 等人，2025)；这些任务以端到端方式定义成功：代码必须能够编译、执行并表现出正确行为，反映真实的部署条件，而非代码片段级保真度。

在**指标**方面，各项评估主要分为两类指标：基于文本的相似度和基于语义的正确性。基于文本的指标，如 BLEU（Papineni 等人，2002），通过衡量与参考文本的 n 元语法重叠来进行评估，成本低且可复现，但既不考虑代码语法，也不考虑数据流和控制流，因此难以忠实评估程序；CodeBLEU（Ren等人，2020）则通过结合标准 n 元语法重叠、关键词加权 n 元语法、抽象语法树（语法）匹配以及数据流（语义）匹配，明确弥补了这一不足，从而更好地与代码任务中的专家判断相关联。

以语义为中心的指标会执行或分析生成产物：pass@k 用于估算 k 个生成候选中至少有一个通过所有单元测试的概率（如今已成为 HumanEval 风格代码合成任务的标准指标）；计算准确率（又称功能等价准确率）则对翻译后的代码运行参考测试，并报告能够编译并产生正确输出的比例；而以优化为重点的研究则报告加速比和优化问题占比（新代码超过固定改进阈值的问题所占比例，例如快 10%）。除了这些通用指标，针对特定任务的度量还会反映领域目标：**SALLM**(Siddiq 等人（2024）) 在 pass@k 的基础上增加了 secure@k 和 vulnerable@k，用于量化生成样本的安全性。**CompilerDream** (Deng 等，2024) 通过报告相对于 LLVM -Oz 等编译器基线的代码大小缩减（例如中间表示指令数），评估端到端代码优化，使指标与嵌入式部署目标保持一致。 **CoTran** (Jana 等人（2024）) 则在正确性指标的基础上增加了错误位置统计。

5.2 演进

虽然 §§ 5.1概述了评估领域的全貌，但本节将按时间顺序深入分析四项代表性任务中最先进水平的演进。这使我们能够追踪近年来取得的具体技术进展。

我们选取了四项任务来展示这一演进：**代码转译**、**神经编译**、**GPU内核生成**以及**LLVM IR 优化**。

我们将分析各项任务的关键基准测试和指标，追踪最先进成果随时间推移的进展，并在图 6中以可视化方式总结。需要特别指出的是：**直接比较并不总是完全公平**。不同研究经常采用不一致的实验设置，例如不同的数据集划分、基线编译器版本或测试条件。尽管如此，这项分析清楚地展现了该领域快速且显著的技术进步。

代码转译：TransCoder 系列研究为源到源翻译建立了基础基准测试，重点关注编程语言之间的转译，例如 Python、Java 和 C语言++。早期采用的指标包括 BLEU 和完全匹配（EM），而计算准确率（CA@1）——即翻译后的代码能否通过一组单元测试——逐渐成为最有意义且经久适用的指标。如图 6(a)所示，最初的 **TransCoder** （Roziere 等人（2020））取得了如下 CA@1 分数：29.9%（Python-Java）和 70.6%（Java-C语言++）。后续研究（如专注于反混淆的**DOBF**（anne Lachaux 等人（2021）））在此基础上有所提升（例如，Python-Java 达到 44.5%）。**TransCoder-ST** （Roziere 等人（2022））带来了显著飞跃：该方法引入了基于测试的筛选和改进的数据整理，将 CA@1 提升至 62.3%（Python-Java）和 75.4%（Java-C语言++）。**TransCoder-IR** （Szafraniec 等人（2023））探索了一种新颖的基于编译器 IR 的方法，取得了相当的成绩：68.7%（Java-C语言++），并展现出对更多编译器后端的稳健适应能力。

神经编译端到端神经编译（例如 C 到 x86）的目标极具挑战性。该领域的进展显然体现为从表面指标转向严格的、基于执行的

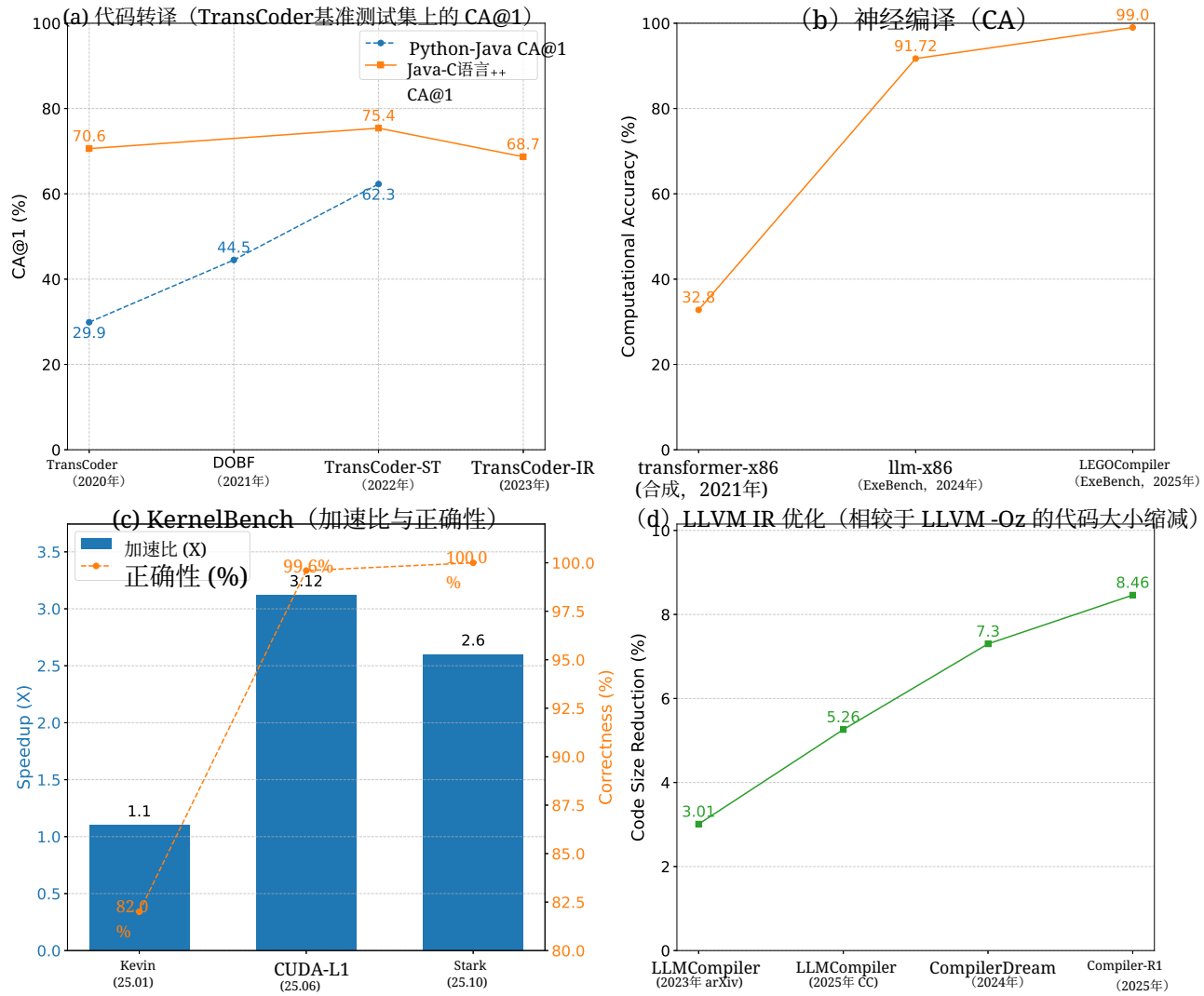

图 6：四项关键大语言模型编译器任务的最先进水平进展可视化：(a)代码转译（TransCoder 上的 CA@1）、(b) 神经编译（ExeBench 上的 CA）、(c) 图形处理器内核生成（KernelBench 上的加速比与正确性），以及(d)LLVM 中间表示优化（代码大小缩减）。

基准测试，如图 6（b）所示。 Armengol-Estape 和 O'Boyle（2021年）的早期工作使用 AnghaBench 对标准 Transformer 进行训练，以测试神经编译能力，并报告了较高的 BLEU（90.2）和语法（98.5）分数。然而，这些指标未能反映语义正确性；在一个包含 64 个项目的小型合成基准测试上——其难度与 ExeBench（尚未发表）相近——他们的 CA 仅为 32.8%。Zhang等人（2024年）通过使用大量数据增强，对大型语言模型 CodeLlama-13B 进行微调，取得了重大进展。在严格的 ExeBench 上进行评估后（该基准由 AnghaBench 开发而来，包含超过 1.7 万个可验证测试用例），他们取得了令人瞩目的 91.72% CA。最近，LEGO-Compiler（Zhang 等人，2025）采用无需训练的分治方法论，通过简化的逐步翻译和基本块级别的代码复杂度降低，将多个模型在 ExeBench 上的 CA 进一步提升至 99% 以上。

图形处理器内核生成对于图形处理器内核生成，评估标准从单纯的语义正确性转向实现**性能加速比**优于Torch eager等强基线。该领域的主要基准测试是 **KernelBench**（Ouyang 等人，2025），多项关键研究都采用了这一基准，具体如图 6（c）所示。然而，直接比较仍然具有挑战性，因为**不同研究可能在不同的 GPU 硬件**（例如 A100 与 L40）上进行评估，而这会显著影响加速比结果。因此，以下发展进程应被视为趋势展示，而非直接且公平的比较。早期工作（**Kevin**（Baronio等人，2025年））突出了正确性挑战：相较于Torch eager，其正确性仅达到**82%** ，平均加速比也仅为**1.10x** 。随后，**Stark**（Dong 等，2025）取得了重大进展，首次在所有任务上实现**100%的正确性**，并在不同基准测试级别上实现了**1.58x** 至**3.03x** 的大幅加速，平均加速比达到**2.60x**。另一项知名工作**CUDA-L1**（Li 等人，2025年）同时采用了 SFT 和强化学习（GRPO），实现了接近 100% 的正确性（249/250），并将相较于

Torch eager的平均加速比进一步提升至最先进水平。其在多种 GPU 上公开的评估结果也进一步支持了这一成果。

LLVM IR 优化对于 LLVM IR 优化，通常以相对于编译器最激进的尺寸优化标志（例如 -Oz）的代码大小缩减率来衡量进展。如图 6(d) 所示，该领域凸显了基准测试缺乏标准化这一挑战。不同研究采用不同的数据集和基线编译器版本，因此难以直接比较。尽管如此，仍能清楚地看到进步趋势。最初的 Meta-LLMCompiler 工作（Cummins 等，2023）通过大规模预训练，实现了相较于 LLVM -Oz 3.01% 的缩减。之后的 Meta-LLMCompiler（Cummins 等，2025）通过增加多步后训练，使模型适应编译器行为仿真和反编译等子任务，将缩减率提升至 5.26%。类似地，CompilerDream（Deng 等，2024）构建了自己的编译器世界模型，并采用奖励平滑技术，通过引导式搜索实现了相较于 LLVM -Oz 7.3% 的缩减，位居 CompilerGym（Cummins 等，2022）排行榜首位。另一种采用智能体方法的研究秉持 LLM 即选择器理念；Compiler-R1（Pan 等人，2025年）使用 GRPO（强化学习）进行编译器标志调优，并报告称相较于 LLVM -Oz 实现了 8.46% 的缩减。

总而言之，本节批判性地审视了**大语言模型与编译器的交汇领域**中的评估方法和基准测试驱动的进展。我们的分析涵盖了对全貌的广泛综述（§§ 5.1），以及对最先进水平演进的深入探讨（§§ 5.2），揭示了两项关键见解。**首先**，各类任务均取得了明确且快速的技术进展，从神经编译正确率超过 99%，到 GPU 内核生成实现显著加速。**其次**，评估工作也日趋成熟，整个领域正果断地从浅层的文本指标（如 BLEU）转向严格的基于执行的指标（如计算准确率和性能加速比）。然而，这项分析也凸显出仍待解决的重大挑战。正如 GPU 和中间表示优化任务所显示的那样，**缺乏标准化的比较协议**（例如固定的硬件、基线和数据集），阻碍了公平、直接的评估。此外，正如我们在§§ 5.1 和表 5中的分析所示，绝大多数基准测试仍停留在**函数级**。具有统一标准的大规模**项目级基准测试**仍然罕见，因此难以恰当地评估可扩展性——这是实际编译器系统的一项关键因素。

6 讨论

在依据设计理念（§ 3）和代码抽象层级（§ 4）对现有研究成果进行分类后，我们现在转向对该领域展开更广泛的讨论。本章综合我们的研究发现，以回答本综述最后两个研究问题。我们首先总结基于大语言模型的方法所带来的主要进展（**研究问题3**）。随后，我们深入探讨定义该研究前沿的共同挑战及其相应的未来机遇（研究问题4）。最后，我们探讨若干对该领域健康演进至关重要的其他主题。

6.1 基于大语言模型的方法的主要进展（研究问题3）

将大语言模型融入编译过程，催生了多项根本性进展，突破了传统手工编写的编译器和早期机器学习技术的能力边界。这些进展主要源于模型能够直接从海量源代码语料库中学习深层语义和结构模式。

- **编译器代码开发与优化的民主化**：大语言模型显著降低了创建复杂代码转换和优化工具的门槛。开发者不再需要多年的专业经验来设计和实现复杂的编译器/转译器启发式方法，而是可以通过对预训练模型进行微调或应用大型基础模型来取得令人瞩目的成果。这种能力源于模型在海量代码语料库上进行预训练，其规模远超任何代码专家一生所能阅读的代码量，使模型能够将广泛的编程模式和技术内化到其权重中，远远超出任何单个人类专家的知识范围。因此，这不仅加速了新型优化器的开发，也让定制化的高性能编译能够被更广泛的群体所使用。

- **发现新颖的优化策略**：几十年来，代码优化一直由人工设计的启发式方法指导。大语言模型能够从海量真实世界代码数据集中学习，因此可以识别复杂模式，并发现人类专家可能难以察觉的新颖优化策略。通过探索广阔的优化空间，并从预训练期间接触到的成功案例中进行泛化，大语言模型有望突破现有基于启发式方法的系统所能达到的性能上限。

- **拓展“编译”的范围与实用性**：大语言模型的应用拓展了编译器传统的角色定义。大规模代码转译和自动程序修复等任务，如今都被视为可行的编译问题。这种实用性的拓展，使编译器不再仅仅是用于翻译和优化的静态工具，而成为支持持续代码维护、迁移和现代化的多功能平台，在整个软件生命周期中创造巨大价值。

研究问题3：基于大语言模型的方法带来了哪些主要进展?
回答：主要进展体现在三个方面：(1) 通过降低所需专业知识门槛并加快实现速度，**让编译器代码开发更加普及**；(2) 通过从海量代码库中学习，发现超越人工设计的启发式方法的**新颖优化策略**；(3) **拓展编译器的用途**，使其成为可用于代码转译和修复等任务的多功能工具。

6.2 常见挑战与未来机遇（研究问题4）

尽管进展迅速，该领域仍面临重大挑战，必须解决这些问题，才能从学术研究迈向生产级的可靠工具。反过来，这些挑战也凸显了未来研究的 promising 方向。

6.2.1 常见挑战与特定任务方法

尽管进展迅速，该领域仍面临重大挑战，亟须弥合与生产级可靠工具之间的差距。要解决这些问题，就必须超越孤立的函数级研究。不过，正如我们将要讨论的，所提出的解决方案**并非普适范式**，而是**高度特定于任务的方法**。当前大语言模型的能力与传统编译器的稳健性之间仍存在显著差距。

确保正确性与可验证性这仍然是最关键的挑战。传统编译器必须保证语义等价性，但大语言模型的概率性可能会引入细微的程序错误。应对这一问题的方法正在不断涌现。

一种策略是**约束生成过程**本身（M¨undler 等人，2025年）。语法引导的约束解码等方法（Willard 和 Louf，2023年；Dong 等，2024）可以强制大语言模型生成遵循给定语法约束的输出，从而确保输出在语法上有效，并保证可编译性。然而，这只能解决语法问题，无法解决语义正确性问题，而后者仍是更难应对的挑战。

更占主导地位的策略是**“翻译器 +验证器”混合模式**，即在大语言模型生成输出后，由确定性组件对其进行检查。这种模式主要有两种形式。最常见的形式是**通过测试套件进行动态功能验证**。这通常置于迭代反馈循环中（Ye 等，2024；Wong 等人，2025），其作用类似于受约束解码，通过迭代纠正错误，但成本更高。

第二种更严格的形式是**以特定任务的方式进行形式化验证**：**LLMLift** (Bhatia 等，2024) 使用 Floyd-Hoare 逻辑，**Qimeng-Xpiler** (Dong 等，2025) 使用 SMT 求解器验证关键变换，而**LLMVectorizer** (Taneja 等，2025) 则使用 Alive2 验证器对生成的单指令多数据内在函数进行形式化验证。尽管结合这些策略前景可期，但要大规模地对所有转换进行严格的形式化验证，仍是一项成本高昂且尚待解决的问题。

对大型真实世界代码库的可扩展性这是一个关键障碍，因为“可扩展性”的定义完全取决于任务。对于图形处理器内核优化等任务，例如 CUDA-L1（Li 等人，2025年）和 KernelBench（Ouyang 等人，2025年），其范围自然局限于单个函数或内核，因此仓库级可扩展性并非问题。

然而，对于遗留代码转译而言，挑战变成了如何管理整个代码库中的过程间上下文。为此，人们采用了分治策略，例如**Alpha-Trans** (Ibrahimzada 等人，2025中的项目划分，或**Oxidizer** (Zhang 等人，2025中的类型检查特征映射。对于项目级修复，**CoCoGen** (Bi 等人，2024)依赖基于 RAG 的上下文检索。

一个更根本的挑战在于**设计假设上的差异**。大语言模型在海量公开代码库上进行预训练，而这些代码库大多遵循“整洁代码”原则或单一职责原则（SRP）。这使其训练数据偏向短小、专注的函数。**然而，真实的编译器无法作出这一假设**；它们必须能够稳健地处理任何语法有效的代码，包括违反这些以人为中心原则的大型单体函数（“上帝函数”）。这给任何旨在取代或补充核心编译器组件的大语言模型带来了关键差距。

LEGO-Compiler （Zhang 等人，2025年）是一项具有代表性的工作，通过将函数分解为更细粒度的基本块或语句，解决了这一以编译器为中心的 问题。这种语义保持分解，作为对可扩展性依赖任务性质的进一步补充，对于非优化场景或仅包含局部、单元内优化的翻译而言才有效；如何将这一分解策略扩展到全局优化场景，仍不明确且充满挑战。

归根结底，即使采用了这些分解策略，大语言模型有限的上下文窗口仍然是难以突破的瓶颈。像 LLVM 或 LinuxKernel 这样的真实世界代码库，其庞大的规模和复杂性远远超出了当前基于大语言模型的方法的可扩展性。

可解释性与可调试性 大语言模型的“黑箱”特性使人们难以信任其决策。这方面的研究尚不成熟，但一些具体策略为解决这一问题提供了方向。一种方法是通过思维链（CoT）提示，要求模型“展示其推理过程”，例如 **CodeOptCoT** (Xu 等人，2024) 和 **SBLLM** (Gao 等人，2025)。

另一种策略侧重于调试与解释：**CompilerGPT**(Pirkelbauer 和 Liao 2025) 分析编译器报告，而 **DCC** (Taylor 等人，2024) 则为新手生成易于理解的错误解释。或许最稳妥的解决方案是 **LLM-as-Generator** 理念（§§ 3.3），其代表是 **Code-Transform** (Cummins 等人，2024)，在这种方法中，生成的产物（转换脚本）本身可供人阅读，也便于调试。

性能与成本效益大型模型的推理成本可能相当高。要让基于大语言模型的优化器切实可用，它带来的性能提升必须超过其自身执行所需的计算成本和延迟。如何在模型规模、推理速度与优化质量之间取得恰当平衡，仍是一项持续面临的挑战。

管理这种成本效益权衡的策略正逐渐涌现。一种策略着眼于通过使用更小的专用模型来降低大语言模型本身的成本；**Perf-codegen** (Peng 等人 2025年) 和 **PerfRL**(Duan等人 2025年均表明，经过执行反馈训练的小型模型也能取得出色的性能，避免了使用大规模模型的高昂成本。

另一种策略则试图通过将大语言模型用于加速现有的高成本流程，以抵消其成本。例如，在自动调优中，**Reasoning-Compiler** (Tang等人 2025年) 和 **TLM** (Zhai等人 2024年利用大语言模型智能地引导搜索。在这种情况下，与其节省的数小时编译和基准测试时间相比，大语言模型的推理成本微不足道。

6.2.2 未来机遇

- **混合编译器系统**：近期最有希望的发展方向并不是取代传统编译器，而是增强它们。将大语言模型富有创造力的模式匹配能力与形式化、确定性的编译器算法所具备的严谨性和速度相结合的混合系统，有望实现两者的优势互补。这可以通过**选择器** 或**生成器** 理念来实现，但同样也适用于**翻译器**模型。例如，系统可以将大部分代码编译工作交给快速可靠的传统编译器，同时调用大语言模型来处理编译器无法处理的特定复杂部分。这样一来，系统便能够支持其最初并未设计用于处理的任务，例如编译包含混合语言代码库的项目，或为尚不存在成熟后端的新兴硬件架构扩展支持。

- **自我改进型与“学习型”编译器**：一个重要的机遇在于，创建能够随着时间推移不断学习和进化的编译器，将大语言模型创造性、非确定性的发现转化为永久且确定的编译器能力。这可以通过一个多阶段过程来实现：

  – 首先，可以将**作为翻译器的大语言模型** 用于探索，发现针对特定代码片段的新颖临时优化，而传统启发式方法会错过这些优化。

  – 接下来，将这些成功且经过验证的转换收集到一个专门的高质量优化示例数据集中。– 最后，使用该数据集来指导**作为生成器的大语言模型** 实现更具雄心的目标：不只是执行一次性的翻译，而是编写新确定性编译器遍或组件的源代码，以系统地实现所发现的优化策略。

  随后可以验证这一新生成的组件，并将其集成到传统编译器框架中。由此产生的编译器将永久地“学会”一项新技能，从而有效地开创一种强大的编译器演进范式。

- **新一代交互式开发者工具**：大语言模型可以改变开发者与编译器交互的方式。我们可以设想未来的集成开发环境：由大语言模型驱动的编译器智能体不仅能优化代码，还能用自然语言解释性能瓶颈、提出复杂的重构建议，并与开发者进行交互，共同改进代码。

基于对关键障碍及相应研究方向的详细讨论，我们现在可以综合这些发现，简要回答第四个研究问题（**研究问题4**）。

> **研究问题4**：这一新兴领域面临哪些共同挑战，又有哪些未来机遇？**回答**：**挑战**：正确性与可验证性、可扩展性、可解释性和性能开销。**机遇**：混合系统、自我改进型编译器和交互式开发者工具。

6.2.3 进一步讨论要点

- **标准化基准测试的必要性：**由于缺乏针对基于大语言模型的编译器所面临独特挑战而设计的基准测试，该领域的发展受到阻碍。虽然已有一些针对特定下游任务的基准测试问世，例如 ExeBench（Armengol-Estap´e 等人，2022年）、Tri- tonBench（Li 等人，2025年）和 VerilogEval（Liu 等人，2023年），但它们往往无法充分涵盖复杂性各个维度，最关键的是可扩展性。对于在真实的大规模应用中评估基于大语言模型的翻译器和优化器而言，这一缺口构成了重大障碍。相反，尽管 SPEC （ Hen- ning 2006 年）等传统测试套件具备所需的规模和复杂性，但它们并非为基于大语言模型的工作流程而设计，其端到端难度也可能超出现有模型的承受能力。这表明，我们迫切需要

一类专为混合评估而设计的新型“大语言模型友好型”基准测试：由外部流程处理样板代码，使基准测试能够专注于评估大语言模型的核心能力，即翻译或优化程序中最关键部分的能力。

•**编程语言/编译器社区与机器学习/大语言模型社区的协同：**取得实质性进展需要深入、互利的合作。机器学习社区可以构建更强大且更了解代码的模型，但编程语言和编译器社区对于界定恰当的问题、提供领域知识（例如程序语义和中间表示结构）、整理高质量数据集，以及开发严格的验证技术以确保最终输出的正确性，至关重要。

•**编译器工程师角色的演变：**大语言模型的兴起势必改变编译器工程师的角色。工作重点可能会从手动编写复杂的手工设计启发式算法，转向一系列新的职责。这些职责可能包括整理海量代码数据集以用于模型训练、设计有效的提示策略、为大语言模型的输出开发可靠的验证系统，以及分析这些模型发现的新颖优化方法，从而深入了解程序性能。

7 结论

在本综述中，我们系统概述了大语言模型在编译领域这一新兴应用；该领域传统上由手工设计的启发式方法主导。我们提出了一个多维分类体系，用以梳理这一多元领域，并依据设计理念、大语言模型方法、代码抽象层次和具体任务类型对现有研究进行分类。我们的分析表明，大语言模型正通过推动编译器开发普及化、发现新颖的优化策略，以及拓展编译器的用途（涵盖代码转译和修复等复杂任务），取得显著进展。尽管如此，该领域仍面临一些关键挑战，包括确保生成代码的正确性和可验证性、实现对大型软件的可扩展性，以及提升模型的可解释性。本综述通过系统梳理当前最先进的研究，并归纳其主要进展与挑战，为研究人员和从业者探索这一令人振奋且具有变革性的领域提供了基础路线图。

声明

利益冲突全体作者代表通讯作者声明，不存在利益冲突。

资助 本工作部分得到了中国国家研发计划（2024YFB4505603）的资助，江苏省重点研发计划（项目编号：BG2024028）和中国国家自然科学基金（U23B2020、62302479、62232015）的资助。

参考文献

Armengol-Estape, J., O’Boyle, M.：学习 C 到 x86 转译：神经编译实验。载于：编程语言与神经符号系统研讨会进展（2021年）

Armengol-Estape,J.,Rocha,R.C.O., Woodruff, J., Minervini, P., O’Boyle, M.：Forklift：一种可扩展的神经提升器。载于：首届语言建模会议（2024年）。https://openreview.net/forum?id=LWfDcI6txJ

Armengol-Estap´e, J., Woodruff, J., Brauckmann, A.,Magalh˜aes,J.W.d.S., O’Boyle, M.F.P.：ExeBench：可执行 C 函数的机器学习规模数据集。载于：第六届 ACM SIGPLAN 机器编程国际研讨会论文集。MAPS 2022，页码 50–59。计算机协会，纽约州纽约，美国（2022年）。https://doi.org/10.1145/3520312.3534867。https://doi.org/10.1145/3520312.3534867

Armengol-Estape, J., Woodruff, J., Cummins, C., O’Boyle, M.F.：Slade：一种用于优化后的汇编代码的便携式小型语言模型反编译器。载于：2024 IEEE/ACM国际代码生成与优化研讨会（CGO），第 67–80 页（2024年）。IEEE

Albuquerque, L., Gheyi, R., Ribeiro, M.：评估大语言模型识别可配置系统中编译错误的能力。arXiv 预印本 arXiv:2407.19087（2024年）

Ahmad, I., Luo, L.：无监督二进制代码翻译及其在代码克隆检测和漏洞发现中的应用。载于：Bouamor, H., Pino, J., Bali, K.（编）：计算语言学协会论文集：EMNLP 2023，第 14581–14592 页。计算语言学协会，新加坡（2023 年）。https://doi.org/10.18653/v1/2023.findings-emnlp.971 。https://aclanthology.org/2023.findings-emnlp.971/

Lachaux, M.-a.、Roziere, B.、Szafraniec, M.、Lample, G.：DOBF：面向编程语言的反混淆预训练目标。在：Beygelzimer, A.、Dauphin, Y.、Liang, P.、Vaughan, J.W.（编）《神经信息处理系统进展》（2021年）。https://openreview.net/forum?id=3ez9BSHTNT

Anthropic：Claude 3 模型系列：Opus、Sonnet、Haiku。技术报告，Anthropic（2024年3月）。可访问：https://www.anthropic.com/news/claude-3-family

Anthropic：Claude Code：Anthropic 的 Claude 人工智能命令行界面。GitHub。访问日期：2025年8月11日（2025年）

Anysphere, Inc.：Cursor：人工智能优先代码编辑器。https://cursor.sh/。访问日期：2025年8月11日（2023年）

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan,D., Jiang, E.,Cai, C., Terry, M., Le, Q., Sutton, C.：使用大语言模型进行程序合成（2021年）。https://arxiv.org/abs/2108.07732

Ahmad, W.U., Tushar, M.G.R., Chakraborty, S., Chang, K.-W.：AVATAR：一个用于 Java-Python 程序翻译的平行语料库。载于：Rogers, A., Boyd-Graber, J., Okazaki, N.（编）《计算语言学协会论文集：ACL 2023》，第2268–2281页。计算语言学协会，多伦多，加拿大（2023年）。https://doi.org/10.18653/v1/2023.findings-acl.143 。https://aclanthology.org/2023.findings-acl.143/

Brownlee, A.E., Callan, J., Even-Mendoza, K., Geiger, A., Hanna, C., Petke, J., Sarro, F., Sobania, D.：使用大语言模型增强遗传改进变异。载于：国际搜索式软件工程研讨会，第153–159页（2023年）。施普林格 Berabi, B., He, J., Raychev, V., Vechev, M.：Tfix：使用文本到文本转换器学习修复编码错误。载于：机器学习国际会议，第780–791页（2021年）。PMLR Baronio, C., Marsella, P., Pan, B., Guo, S., Alberti, S.：Kevin：用于生成 CUDA 内核的多轮强化学习（2025年）。https://arxiv.org/abs/2507.11948Bhatia, S., Qiu, J., Hasabnis, N., Seshia, S.A., Cheung, A.：使用大语言模型进行经验证的代码转译。神经信息处理系统进展 **37**，41394–41424（2024年）Bi, Z., Wan, Y., Wang, Z., Zhang, H., Guan, B., Lu, F., Zhang, Z., Sui, Y., Jin, H., Shi, X.：通过编译器反馈对项目级代码上下文进行迭代细化，以实现精确的代码生成。载于：Ku, L.-W., Martins, A., Srikumar, V.（编）：计算语言学协会论文集：ACL 2024，第2336–2353页。计算语言学协会，泰国曼谷（2024年）。https://doi.org/10.18653/v1/2024.findings-acl.138 。
h
t
tps://aclanthology.org/2024.findings-acl.138/
Chakraborty, S., Ahmed, T., Ding, Y., Devanbu, P.T., Ray, B.：Natgen：通过“自然化”源代码进行生成式预训练。载于：第30届 ACM 欧洲软件工程联合会议暨软件工程基础研讨会论文集，第18–30页（2022年）

Cai, S.,Chen, H.,Huang, Y., Ming,Z.：ComPAT：编译器原理课程助教。载于：KSEM（5），第74–83页（2024年）

Cassano, F., Gouwar, J., Lucchetti, F., Schlesinger, C., Freeman, A., Anderson, C.J., Feldman, M.Q., Greenberg, M., Jangda, A., Guha, A.：从高资源语言向低资源编程语言迁移知识，以用于代码大语言模型。《ACM编程语言论文集》**8**（OOPSLA2）（2024年）https://doi.org/10.1145/3689735

Cao, Y., Liang, R., Chen, K., Hu, P.：利用神经网络反编译优化后的二进制文件。载于：第38届年度计算机安全应用大会论文集，第508–518页（2022年）

Chen, T., Moreau, T., Jiang, Z., Zheng, L., Yan, E., Cowan, M., Shen, H., Wang, L., Hu, Y., Ceze, L., Guestrin,C.,Krishnamurthy,A.：TVM：一种用于深度学习的自动化端到端优化编译器。载于：第13届USENIX操作系统设计与实现大会论文集。OSDI’18，第579–594页。USENIX协会，美国（2018年）

Cummins,C., Seeker, V.,Armengol-Estap´e, J., Markosyan, A.H., Synnaeve, G., Leather, H：使用大语言模型实现精准代码重写——不要转换代码，而要编写转换（2024年）。https://arxiv.org/abs/2410.08806

Cummins, C., Seeker, V., Grubisic, D., Elhoushi, M., Liang, Y., Roziere, B., Gehring, J., Gloeckle, F., Hazelwood, K., Synnaeve, G., Leather, H：用于编译器优化的大语言模型（2023年）。https://arxiv.org/abs/2309.07062

Cummins, C., Seeker, V., Grubisic, D., Roziere, B., Gehring,J., Synnaeve, G.,Leather, H：LLM 编译器：用于编译器优化的基础语言模型。载于：第34届ACM SIGPLAN国际编译器构造大会论文集。CC ’25，第141–153页。计算机协会，纽约州纽约，美国（2025年）。https://doi.org/10.1145/3708493.3712691。https://doi.org/10.1145/3708493.3712691

Chen, M., Tworek, J., Jun, H.,Yuan, Q., Oliveira Pinto, H.P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F.P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W.H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A.N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish,

S.,Sutskever, I., Zaremba,W.: 评估在代码上训练的大语言模型 (2021年)。
https://arxiv.org/abs/2107.03374Chen, M., T worek, J., Jun, H., Yuan, Q., Pinto, H.P.d.O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al.: 评估在代码上训练的大语言模型。arXiv 预印本 arXiv:2107.03374 (2021年)

Cummins, C., Wasti, B., Guo, J., Cui, B., Ansel, J., Gomez, S., Jain, S., Liu, J., Teytaud, O., Steiner, B., 等：Compilergym：面向人工智能研究的稳健、高性能编译器优化环境。在：2022年 IEEE/ACM 代码生成与优化国际研讨会（CGO），第 92–105 页（2022年）。IEEE

Cui, T., Yew, P.-C., McCamant, S., Zhai, A.：DeCOS：受大语言模型启发的用于编译器优化选择的数据高效强化学习。在：2025年国际超级计算会议论文集。ICS'25。计算机协会，美国纽约州纽约（2025年）。
https://doi.org/10.1145/3721145.3725765。
https://doi.org/10.1145/3721145.3725765

Cui, F., Yin, C., Zhou, K., Xiao, Y., Sun, G., Xu, Q., Guo, Q., Liang, Y., Zhang, X., Song, D., 等：Origen：通过代码到代码增强和自我反思提升寄存器传输级代码生成。载于：第43届 IEEE/ACM 计算机辅助设计国际会议论文集，第1–9页（2024年）

Chen, L., Zhang, S., Xu, F., Xing, Z., Wan, L., Zhang, X., Feng, Z.：一种用于神经代码翻译的免测试语义错误定位框架。arXiv 预印本 arXiv:2410.22818（2024年）

Ding, X., Chen, L., Emani, M., Liao, C., Lin, P.- H., Vanderbruggen, T., Xie, Z., Cerpa, A., Du, W.：HPC-GPT：面向高性能计算的大语言模型集成。载于：高性能计算、网络、存储与分析国际会议 SC '23研讨会论文集。SC-W '23，第951–960页。计算机协会，纽约州纽约，美国（2023年）。
https://doi.org/10.1145/3624062.3624172。
https://doi.org/10.1145/3624062.3624172

DeLorenzo, M., Chowdhury, A.B., Gohil, V., Thakur, S., Karri, R., Garg, S., Rajendran, J.: 让每一步都发挥作用：使用蒙特卡洛树搜索进行基于大语言模型的高质量寄存器传输级代码生成。arXiv 预印本 arXiv:2402.03289（2024年） Duan, S., Kanakaris, N., Xiao, X., Ping, H., Zhou, C., Ahmed, N.K., Ma, G., Capota, M., Willke, T.L., Nazarian, S., Bogdan,P.:PerfRL：一种用于高效代码优化的小型语言模型框架（2025年）。
https://arxiv.org/abs/2312.05657

Deligiannis, P., Lal, A., Mehrotra, N., Poddar, R., Rastogi, A.: RustAssistant：使用大语言模型修复 Rust 代码中的编译错误。载于：2025 IEEE/ACM 第47届国际软件工程会议（ICSE），第267–279页（2024年）。IEEE计算机学会

De Moura, L., Bjørner, N.：Z3：一种高效的 SMT 求解器。载于：第 14 届软件理论与实践国际会议论文集——第 14 届系统构造与分析工具及算法国际会议。TACAS'08/ETAPS'08，第 337–340 页。施普林格，柏林，海德堡（2008年）

Dong, Y., Ruan,C.F., Cai, Y., Lai, R., Xu,Z., Zhao, Y., Chen, T.：XGrammar：面向大语言模型的灵活高效结构化生成引擎（2024年）。https://arxiv.org/abs/2411.15100

Da Silva, A.F., Kind, B.C., Souza Magalh˜aes, J.W., Rocha, J.N., Guimaraes, B.C.F., Pereira, F.M.Q.：AnghaBench：包含一百万个可编译 C语言基准测试的代码大小缩减套件。载于：2021年 IEEE/ACM 代码生成与优化国际研讨会（CGO）论文集，第 378–390 页（2021年）。IEEE

Dong, S., Wen, Y., Bi, J., Huang, D., Guo, J., Xu, J., Xu, R., Song, X., Hao, Y., Zhou, X., 等：Qimengxpiler：采用神经符号方法为深度学习系统转译编译张量程序。载于：第19届 USENIX 操作系统设计与实现研讨会（OSDI 25）（2025年）

Deng, C.,Wu, J., Feng, N., Wang, J., Long, M.：Compil- erdream：学习用于通用代码优化的编译器世界模型。arXiv 预印本 arXiv:2404.16077（2024年）

Dong, J., Yang,Y.,Liu,T., Wang,Y., Qi, F., Tarokh, V., Rangadurai, K., Yang, S.：STARK：用于优化内核的智能体战略团队（2025年）。https://arxiv.org/abs/2510.16996

Fang, X., Mukhanov：面向基于大语言模型的优化编译器。大语言模型能否学会应用单一窥孔优化？推理是大语言模型所需的一切！arXiv预印本 arXiv:2412.12163（2024年） Friedman, D., Wettig, A., Chen, D.：学习 Transformer 程序。《神经信息处理系统进展》**36**，49044–49067（2023年） Gemini 团队，谷歌：Gemini：一系列功能强大的多模态模型。arXiv预印本 arXiv:2312.11805（2023年） Gao, S., Gao, C., Gu, W., Lyu, M.R.：基于搜索的大语言模型代码优化。载于：2025年IEEE/ACM第47届国际软件工程会议（ICSE），第578–590页（2025年）。IEEE Gao, Y., Liang, L., Li, Y., Li, R., Wang, Y.：函数级编译溯源识别

通过多方面的神经特征蒸馏与融合。《Electronics》**13**(9)（2024年）https://doi.org/10.3390/electronics13091692Guo, Z.C., Moses, W.S.：使 Transformer 能够理解底层程序。载于：2022年IEEE高性能极限计算会议（HPEC），第1–9页（2022年）。IEEE google-gemini：gemini-cli：Google Gemini CLI 和 Python API。GitHub。访问日期：2025年8月11日（2025年）Guthaus, M.R., Ringenberg, J.S., Ernst, D., Austin, T.M., Mudge, T.,Brown, R.B.：MiBench：免费的、具有商业代表性的嵌入式基准测试套件。载于：第四届年度IEEE工作负载特征国际研讨会论文集。WWC-4（目录号01EX538），第3–14页（2001年）。IEEE

Grubisic, D., Seeker, V., Synnaeve, G., Leather, H., Mellor-Crummey, J., Cummins, C.：面向编译器的大语言模型优先级采样。载于：第四届机器学习与系统研讨会论文集，第 91–97 页（2024年）

Gao, Z., Wang, H., Wang, Y., Zhang, C.：虚拟编译器是汇编代码搜索所需的一切。载于：Ku, L.-W., Martins, A., Srikumar, V.（编），计算语言学协会第 62 届年会论文集（第一卷：长篇论文）第 3040–3051 页。计算语言学协会，泰国曼谷（2024年）。https://doi.org/10.18653/v1/2024.acl-long.167 。https://aclanthology.org/2024.acl-long.167/

Gao, H., Yang, Y., Sun, M., Wu, J., Zhou, Y., Xu, B.: Clozemaster：通过利用大语言模型填充掩码真实程序，对 Rust 编译器进行模糊测试。载于：2025年IEEE/ACM 第47届国际软件工程大会（ICSE），第712–712 页（2025年）。IEEE计算机学会

Geng, H., Zhong, M., Zhang, P., Lv, F., Feng, X.: Optango：针对二进制差分的无数编译器优化的多中心表示学习。载于：ISSRE，第 774–785 页（2023年）。https://doi.org/10.1109/ISSRE59848.2023.00013

Hong, C., Bhatia, S., Cheung, A., Shao, Y.S.: Auto-comp：面向张量加速器的大语言模型驱动代码优化。arXiv预印本 arXiv:2505.18574（2025年）Hu, L., Chen, G., Shang, X., Cheng, S., Wu, B., LiGangyang, L., Zhu, X., Zhang, W., Yu, N.：CompileAgent：借助集成工具的大语言模型智能体系统，实现真实世界仓库级自动编译。载于：Che, W., Nabende,J., Shutova, E., Pilehvar, M.T.（编）《第63届计算语言学协会年会论文集》（第1卷：长篇论文），第2078–2091页。计算语言学协会，奥地利维也纳（2025年）。https://aclanthology.org/2025.acl-long.103/Huang, D., Dai, J., Weng, H., Wu, P., Qing, Y., Cui, H., Guo, Z., Zhang, J.：Effilearner：通过自优化提升生成代码的效率。《神经信息处理系统进展》**37**，84482–84522（2024年）Heckel, K.：用于代码优化的神经进化编译器控制。载于：《遗传与进化计算大会伴随会议论文集》，第2362–2365页（2023年）Henning, J.L.：SPEC CPU2006基准测试描述。《ACM SIGARCH计算机体系结构新闻》**34**(4)，1–17（2006年）Hu, P., Liang, R., Chen, K.：Degpt：使用大语言模型优化反编译器输出。载于：《2024年网络与分布式系统安全研讨会论文集》，第267622140卷（2024年）

Huang, D., Nan, Z., Hu, X., Jin, P., Peng, S., Wen, Y., Zhang, R., Du, Z., Guo, Q., Pu, Y., Chen, Y.: ANPL：通过交互式分解实现自然编程。载于：第37届神经信息处理系统国际会议论文集。NIPS '23。Curran Associates Inc.，美国纽约州雷德胡克（2023年）

He, K., Zhang, X., Ren, S., Sun, J.：用于图像识别的深度残差学习。载于：IEEE计算机视觉与模式识别会议论文集，第770–778页（2016年）

Italiano, D., Cummins, C.：利用大语言模型发现编译器中遗漏的代码大小优化。载于：第34届ACM SIGPLAN国际编译器构造大会论文集，第81–91页（2025年）

Ishida, S., Corrado, G., Fedoseev, G., Yeo, H., Russell, L., Shotton, J., Henriques, J.F., Hu, A.：Langprop：一种应用于驾驶领域、基于大语言模型的代码优化框架。载于：ICLR 2024年大语言模型（LLM）智能体研讨会（2024年）。https://openreview.net/forum?id=JQJJ9PkdYC

Ibrahimzada, A.R., Ke, K., Pawagi, M., Abid, M.S., Pan, R., Sinha, S., Jabbarvand, R.：Alpha-trans：一种用于仓库级代码翻译与验证的神经符号组合方法。《ACM软件工程汇刊》**2**(FSE)，2454–2476（2025年）

Just, R., Jalali, D., Ernst, M.D.：Defects4J：一个现有故障数据库，用于支持针对 Java程序的受控测试研究。载于：2014年软件测试与分析国际研讨会论文集，第437–440页（2014年）

Jana, P., Jha, P., Ju, H., Kishore, G., Mahajan, A.,

Ganesh, V.：CoTran：一种基于大语言模型的代码翻译器，采用强化学习，并利用编译器和符号执行提供的反馈。在：ECAI（2024年）

Jin, X., Larson, J., Yang, W.,Lin, Z.：二进制代码摘要：对 ChatGPT/GPT-4 及其他大语言模型的基准测试（2023年）。https://arxiv.org/abs/2312.09601

Jin, L., Ruan, Z., Mai, H., Shang, J.：VeriLOCC：通过大语言模型实现端到端跨架构寄存器分配。arXiv预印本 arXiv:2506.17506（2025年）

Jiao, M., Yu, T., Li, X., Qiu, G., Gu, X., Shen, B.：神经代码翻译的评估：分类体系与基准测试。载于：第38届 IEEE/ACM 国际自动化软件工程会议论文集，第1529–1541页（2023年）

Jiang, H., Zhu, J., Wan, Y., Fang, B., Zhang, H., Jin, R., Guan, Q.：大语言模型能否理解编译器中的中间表示？arXiv 预印本 arXiv:2502.06854（2025）Khan, W., Alrabaee, S., Al-kfairy, M., Tang, J., Ray- mond Choo, K.-K.：使用视觉Transformer识别混淆二进制文件的编译器来源。《国际法医学：数字调查》**49**，301764（2024）https://doi.org/10.1016/j.fsidi.2024.301764 。DFRWS USA 2024——美国数字取证研究会议第24届年会精选论文 Kitchenham, B., Charters, S., 等：软件工程系统性文献综述实施指南（2007）Kadosh, T., Hasabnis, N., Soundararajan, P., Vo, V.A., Capota, M., Ahmed, N., Pinter, Y., Oren, G.：OMPar：利用人工智能驱动的源到源编译实现自动并行化（2024）。https://arxiv.org/abs/2409.14771Krizhevsky, A., Sutskever, I., Hinton, G.E.：使用深度卷积神经网络进行ImageNet分类。《神经信息处理系统进展》**25**（2012）Kabir, A., Wang, S., Tian, Y., Chen, T.-H., Asaduz- zaman, M., Zhang, W.：ZS4C：使用大语言模型对不完整代码片段进行可编译代码的零样本合成。《ACM软件工程与方法论汇刊》**34**(4)，1–30（2025）Kang, S., Yoon, J., Yoo, S.：大语言模型是少样本测试器：探索基于大语言模型的一般缺陷复现。载于：2023年IEEE/ACM第45届国际软件工程会议（ICSE），第2312–2323页（2023）。IEEE Lv, J., He, X., Liu, Y., Dai, X., Shen, A., Li, Y., Hao, J., Ding, J., Hu, Y., Yin, S.：Hpctranscompile：用于高性能的人工智能编译器生成数据集

CUDA转译及大语言模型初步探索。arXiv预印本 arXiv:2506.10401（2025）

Lindner, D., Kram´ar, J., Farquhar, S., Rahtz, M., McGrath, T., Mikulik, V.：Tracr：作为可解释性实验室的编译型 Transformer。神经信息处理系统进展**36**，37876–37899（2023年）

Li, J., Li, S., Gao, Z., Shi, Q., Li, Y., Wang, Z., Huang, J., WangHaojie, W., Wang, J., Han, X., Liu, Z., Sun, M.：TritonBench：用于评测大语言模型生成 Triton 算子的能力的基准测试。载于：Che, W.、Nabende, J.、Shutova, E.、Pilehvar, M.T.（编）：计算语言学协会论文集：ACL 2025，23053–23066页。计算语言学协会，奥地利维也纳（2025年）。https://doi.org/10.18653/v1/2025.findings-acl.1183 。https://aclanthology.org/2025.findings-acl.1183/

Lopes, N.P., Lee, J., Hur, C.-K., Liu, Z., Regehr, J.：Alive2：面向 LLVM 的有界翻译验证。载于：第 42 届 ACM SIGPLAN 编程语言设计与实现国际会议论文集。PLDI 2021，第 65–79 页。计算机协会，纽约州纽约，美国（2021）。https://doi.org/10.1145/3453483.3454030。https://doi.org/10.1145/3453483.3454030Luo, T., Lee, H., Johnson, J.：神经形状编译器：用于文本、点云和程序之间转换的统一框架。机器学习研究汇刊（2023）Lu, Y., Liu, S., Zhang, Q., Xie, Z.：RTLLM：一种用于寄存器传输级生成的大语言模型开源基准测试。载于：2024 年第 29 届亚洲及南太平洋设计自动化会议（ASP-DAC），第 722–727 页（2024）。IEEE Lin, H., Maas, M., Roquemore, M., Hasanzadeh, A., Lewis, F., Simonson, Y., Yang, T.-W., Yazdan-bakhsh, A., Altinb¨uken, D., Papa, F., 等：ECO：由大语言模型驱动的仓储规模计算机高效代码优化器。arXiv 预印本 arXiv:2503.15669（2025）Liu, M., Pinckney, N., Khailany, B., Ren, H.：VerilogEval：评估用于 Verilog 代码生成的大语言模型。载于：2023 年 IEEE/ACM 计算机辅助设计国际会议（ICCAD），第 1–8 页（2023）。IEEE Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., K¨uttler, H., Lewis, M., Yih, W.-t., Rockt¨aschel, T., Riedel, S., Kiela：面向知识密集型自然语言处理任务的检索增强生成。载于：第 34 届神经信息处理系统国际会议论文集。NIPS '20。Curran Associates Inc.，美国纽约州雷德胡克（2020）

Li, X., Sun, X., Wang, A., Li, J., Chris, S.: CUDA-L1: 通过对比强化学习改进 CUDA 优化。arXiv 预印本 arXiv:2507.14111（2025）

Mannarswamy, S., Das, D.: 在 LLVMCompiler 中学习组合指令（2022年）。https://arxiv.org/abs/2202.12379

M¨undler, N., He, J., Wang, H., Sen, K., Song, D., Vechev, M.: 利用语言模型进行类型约束代码生成。《ACM 编程语言学报》9(PLDI)，601–626（2025年）

Mammadli, R., Jannesari, A., Wolf, F.: 通过深度强化学习进行静态神经编译器优化。载于：2020年 IEEE/ACM 第6届 LLVM HPC 编译器基础设施研讨会（LLVM-HPC）暨百亿亿次计算分层并行研讨会（HiPar），第 1–11 页（2020年）。IEEE

Madaan, A., Shypula, A., Alon, U., Hashemi, M., Ranganathan, P., Yang, Y., Neubig, G., Yazdan-bakhsh, A.: 学习能够提升性能的代码编辑。arXiv 预印本，arXiv:2302.07867（2023年）

Macedo, M.、Tian, Y.、Cogo, F.、Adams, B.: 探索输出格式对代码翻译大语言模型评估的影响。载于：2024年 IEEE/ACM 人工智能基础模型与软件工程第一届国际会议论文集，第57–68页（2024年）

Nichols, D.、Davis,J.H.、Xie, Z.、Rajaram, A.、Bhatele, A.: 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，第281–294页（2024年）

Niu, C.、Li, C.、Ng, V.、Lo, D.、Luo, B.: Fair：编译器中间表示的流类型感知预训练。载于：IEEE/ACM 第46届软件工程国际会议论文集。ICSE '24。计算机协会，纽约州纽约，美国（2024年）。https://doi.org/10.1145/3597503.3608136。https://doi.org/10.1145/3597503.3608136

NVIDIA公司：CUTLASS Python 接口概述。https://docs.nvidia.com/cutlass/media/docs/pythonDSL/overview.html。访问日期：2025年8月11日（2025年）

Nakkab, A., Zhang,S.Q.,Karri,R., Garg,S.: 罗马并非一日建成：基于大语言模型的芯片设计的层次化提示。在：2024年 ACM/IEEE 计算机辅助设计机器学习国际研讨会论文集，第 1–11 页（2024年）

Ouyang, A., Guo, S., Arora, S., Zhang, A.L., Hu, W., Re, C., Mirhoseini, A.: Kernelbench：大语言模型能否编写高效的图形处理器内核？载于：第四十二届机器学习国际会议（2025年）。https://openreview.net/forum?id=yeoN1iQT1xOu, X., Li, C., Jiang, Y., Xu, C.: 变异器重装上阵：使用大语言模型生成的变异算子对编译器进行模糊测试。载于：第29届 ACM 编程语言与操作系统架构支持国际会议论文集，第4卷，第298–312页（2024年）OpenAI: GPT-4 技术报告（2024年）。https://arxiv.org/abs/2303.08774Palkowski, M., Gruzewski, M.: 使用大语言模型通过多面体编译自动生成 OpenCL 代码。载于：2024年第19届计算机科学与智能系统会议（FedCSIS），第671–676页（2024年）。IEEE

Peng, Y., Gotmare, A.D., Lyu, M.R., Xiong, C., Savarese, S., Sahoo, D.: Perfcodegen：利用执行反馈提升大语言模型生成代码的性能。在：2025年 IEEE/ACM 人工智能基础模型与软件工程第二届国际会议（Forge），第 1–13 页（2025年）。IEEE

Pan, R., Ibrahimzada, A.R., Krishna, R., Sankar, D., Wassi, L.P., Merler, M., Sobolev, B., Pavuluri, R., Sinha, S., Jabbarvand, R.: 迷失在翻译中：研究大语言模型在翻译代码时引入的程序错误。在：IEEE/ACM 软件工程第 46 届国际会议论文集，第 1–13 页（2024年）

Puri,R., Kung,D.S.,Janssen, G.,Zhang, W., Domeni- coni, G., Zolotov, V., Dolby, J., Chen, J., Choud- hury, M., Decker, L., Thost, V., Buratti, L., Pujar, S., Ramji, S., Finkler, U., Malaika, S., Reiss, F.: CodeNet：用于学习多种编程任务的大规模代码人工智能数据集（2021年）。https://arxiv.org/abs/2105.12655

Purschke, N., Kirchner, S., Knoll, A.: Speedgen：通过基于大语言模型的性能优化提升代码效率。载于：2025年IEEE软件分析、演进与再工程国际会议（SANER），第1–12页（2025年）。IEEE

Pirkelbauer, P., Liao, C.: Compilergpt：利用大语言模型分析并处理编译器优化报告。arXiv预印本 arXiv:2506.06227（2025年）

Pan, H., Lin, H., Luo, H., Liu, Y., Yao, K., Zhang, L., Xing, M., Wu, Y.: Compiler-R1：通过强化学习实现智能体编译器自动调优。arXiv 预印本 arXiv：2506.15701（2025年）

Papineni, K., Roukos, S., Ward, T., Zhu, W.-J.: BLEU:

一种自动评估机器翻译的方法。载于：第 40 届计算语言学协会年会论文集，第 311–318 页（2002 年）

Peng, D., Zheng, S., Li, Y., Ke, G., He, D., Liu, T.-Y.：神经网络如何理解程序？载于：国际机器学习会议，第 8476–8486 页（2021年）。PMLR

Rong, Y., Du, T., Li, R.,Bao, W.：将基于大语言模型的代码优化与类人排除式推理相结合，用于计算教育。沙特国王大学计算机与信息科学杂志**37**(5)，87（2025年）

Ren, S., Guo, D., Lu, S., Zhou, L., Liu, S., Tang, D., Sundaresan, N., Zhou, M., Blanco, A., Ma, S.：CodeBLEU：一种代码合成自动评估方法。arXiv 预印本，arXiv:2009.10297（2020年）

Roziere, B., Lachaux, M.-A., Chanussot, L., Lample, G.：编程语言的无监督翻译。载于：第34届神经信息处理系统国际会议论文集。NIPS '20。Curran Associates Inc.，美国纽约州雷德胡克（2020年）Romero Rosas, M.A., Torres Sanchez, M.A., Eigenmann, R.：是否应该让人工智能优化你的代码？经典优化编译器与当前大语言模型的比较研究。载于：2025年亚洲超级计算大会论文集，第22–29页（2025年）Roziere, B., Zhang, J., Charton, F., Harman, M., Synnaeve, G., Lample, G.：利用自动化单元测试进行无监督代码翻译。载于：国际学习表征会议（2022年）。https://openreview.net/forum?id=cmt-6KtR4c4Ren, X., Zhang, T., Xu, X., Zheng, Y.-C., Zhang, S.：利用机器学习优化量子编译。载于：第61届ACM/IEEE设计自动化大会论文集，第1–4页（2024年）Sajjadinasab, R., Arora, S., Drepper, U., Sanaullah, A., Herbordt, M.：一种用于优化GCC编译器标志设置的基于图的算法。载于：2024年IEEE高性能极限计算会议（HPEC），第1–8页（2024年）。IEEE Shaw, P., Cohan, J., Eisenstein, J., Lee, K., Berant, J., Toutanova, K.：ALTA：基于编译器的Transformer分析。机器学习研究汇刊（2025年）Sun, T., Chai, L., Yang, J., Yin, Y., Guo, H., Liu, J., Wang, B., Yang, L., Li, Z.：UniCoder：通过通用代码扩展代码大语言模型。载于：Ku, L.-W., Martins, A., Srikumar, V.（编者）《第62届计算语言学协会年会论文集》（第1卷：长篇论文），第1812–1824页。计算语言学协会，泰国曼谷（2024年）。https://doi.org/10.18653/v1/2024.acl-long.100 。https://aclanthology.org/2024.acl-long.100/Siddiq,M.L.,Silva Santos,J.C.,Devareddy, S., Muller, A.：SALLM：生成代码安全评估。载于：第39届IEEE/ACM自动化软件工程国际会议研讨会论文集。ASEW '24，第54–65页。计算机协会，美国纽约州纽约（2024年）。https://doi.org/10.1145/3691621.3694934。https://doi.org/10.1145/3691621.3694934Saldyt, L., Kambhampati, S.：采用神经编译库的算法语言模型（2025年）。https://arxiv.org/abs/2407.04899

Sibaee, S., Najar, O., Ghouti, L., Koubaa, A.：将大语言模型用作阿拉伯语编程语言的编译器。arXiv 预印本 arXiv:2403.16087（2024年）

Szafraniec, M., Roziere, B., Leather, H.J., Labatut, P., Charton, F., Synnaeve, G.：使用编译器表示进行代码翻译。载于：第十一届国际学习表示会议（2023年）。https://openreview.net/forum?id=XomEU3eNeSQShao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Bi, X., Zhang, H., Zhang, M., Li, Y.K., Wu, Y., Guo, D.：DeepSeekMath：推动开放语言模型数学推理的极限（2024年）。https://arxiv.org/abs/2402.03300Shen, L., Yang, Q., Zheng, Y., Li, M.：Autoiot：由大语言模型驱动的人工智能物联网应用自动自然语言编程。载于：Mobicom 2025（2025年）Tabnine：Tabnine：人工智能代码补全工具。https://www.tabnine.com/。访问日期：2025年8月11日（2022年）Thakur, S., Ahmad, B., Pearce, H., Tan, B., Dolan-Gavitt, B., Karri, R., Garg, S.：Verigen：用于Verilog代码生成的大语言模型。ACM电子设计自动化系统汇刊。**29**（3）（2024年）https://doi.org/10.1145/3643681TehraniJamsaz, A., Bhattacharjee, A., Chen, L., Ahmed, N.K., Yazdanbakhsh, A., Jannesari, A.：Coderosetta：推动无监督并行编程代码翻译的边界。载于：第三十八届神经信息处理系统年会（2024年）。https://openreview.net/forum?id=V6hrg4O9ggThakur, S., Blocklove, J., Pearce, H., Tan, B., Garg, S., Karri, R.：Autochip：自动化硬件描述语言生成

使用大语言模型反馈。arXiv预印本 arXiv:2311.04887（2023年）Tavarageri, S., Goyal, G., Avancha, S., Kaul, B., Upadrasta, R.：利用人工智能的编译器技术进行深度学习代码优化。arXiv预印本 arXiv:2104.05573（2021年）Tillet, P., Kung, H.T., Cox, D.：Triton：用于分块神经网络计算的中间语言和编译器。载于：第三届ACM编程语言特别兴趣小组机器学习与编程语言国际研讨会论文集。MAPL 2019，第10–19页。计算机协会，美国纽约州纽约，美国（2019年）。https://doi.org/10.1145/3315508. 3329973。https://doi.org/10.1145/3315508.3329973Tan, H., Luo, Q., Li, J., Zhang, Y.：LLM4Decompile：使用大语言模型反编译二进制代码。载于：Al-Onaizan, Y., Bansal, M., Chen, Y.-N.（编），《2024年自然语言实证方法会议论文集》，第3473–3487页。计算语言学协会，美国佛罗里达州迈阿密，美国（2024年）。https://doi.org/10.18653/v1/2024.emnlp-main.203 。https://aclanthology.org/2024.emnlp-main.203/Taneja, J., Laird, A., Yan, C., Musuvathi, M., Lahiri, S.K.：Llm-vectorizer：基于大语言模型的经验证循环向量化器。载于：第二十三届ACM/IEEE代码生成与优化国际研讨会论文集。CGO '25，第137–149页。计算机协会，美国纽约州纽约，美国（2025年）。https://doi.org/10.1145/3696443.3708929。https://doi.org/10.1145/3696443.3708929Tang, S., Priebe, C., Mahapatra, R., Qin, L., Esmaeilzadeh, H.：通过大语言模型推理进行编译器优化，以实现高效模型服务。arXiv预印本 arXiv:2506.01374（2025年）Tsimpourlas, F., Petoumenos, P., Xu, M., Cummins, C., Hazelwood, K., Rajan, A., Leather, H.：BenchDirect：面向编译器基准测试的定向语言模型（2023年）。https://arxiv.org/abs/2303.01557Taylor, A., Vassar, A., Renzella, J., Pearce, H.：dcc –help：利用大语言模型生成上下文感知的错误解释，改变编译器的角色。载于：第五十五届ACM计算机科学教育技术研讨会论文集，第V. 1卷。SIGCSE 2024，第1314–1320页。计算机协会，美国纽约州纽约，美国（2024年）。https://doi.org/10.1145/3626252.3630822。https://doi.org/10.1145/3626252.3630822Wen, Y., Guo, Q., Fu, Q., Li, X., Xu, J., Tang, Y., Zhao, Y., Hu, X., Du, Z., Li, L., 等：Babeltower：学习自动并行化程序翻译。载于：国际机器学习会议，第23685–23700页（2022年）。PMLR Weiss, G., Goldberg, Y., Yahav, E.：像 Transformer 一样思考。载于：国际机器学习会议，第 11080–11090 页（2021年）。PMLR Wang, X., Hui, X., Liao, C., Shen, X.：利用编译器引导的大语言模型对以输入为中心的代码优化进行归约分析。《ACM 程序设计语言会议论文集》**9**（PLDI）（2025年）https://doi.org/10.1145/3729282Willard, B.T., Louf, R.：大语言模型的高效引导生成（2023年）。https://arxiv.org/abs/2307.09702Wang, Z., O'Boyle, M.：机器学习在编译器优化中的应用。《IEEE 论文集》**106**（11），1879–1901（2018）https://doi.org/10.1109/JPROC.2018.2817118Wang, H., Qu, W., Katz, G., Zhu, W., Gao, Z., Qiu, H., Zhuge, J., Zhang, C.：JTrans：用于二进制代码相似性检测的跳转感知 Transformer。载于：第 31 届 ACM SIGSOFT 软件测试与分析国际研讨会论文集。ISSTA 2022，第 1–13 页。计算机协会，纽约州纽约，美国（2022年）。https://doi.org/10.1145/3533767.3534367。https://doi.org/10.1145/3533767.3534367Wang, T., Wang, R., Chen, Y., Yu, L., Pan, Z., Zhang, M., Ma, H., Zheng, J.：通过命令反馈，利用大语言模型增强黑盒编译器选项模糊测试。载于：2024 IEEE 第 35 届软件可靠性工程国际研讨会（ISSRE），第 319–330 页（2024年）。IEEE Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E.H., Le, Q.V., Zhou, D.：思维链提示可激发大语言模型的推理能力。载于：第 36 届神经信息处理系统国际会议论文集。NIPS '22。Curran Associates Inc.，美国纽约州雷德胡克（2022年）。Wong, W.K., Wu, D., Wang, H., Li, Z., Liu, Z., Wang, S., Tang, Q., Nie, S., Wu, S.：DecLLM：大语言模型增强的可重新编译反编译，使反编译代码可供程序化使用。《ACM 软件工程论文集》**2**（ISSTA），1841–1864（2025年）Wei, Y., Xia, C.S., Zhang, L.：协同辅助代码补全工具：将大语言模型与代码补全引擎融合，用于自动程序修复。载于：第 31 届 ACM 欧洲软件工程会议与软件工程基础研讨会联合会议论文集。ESEC/FSE 2023，第 172–184 页。计算机协会，美国纽约州纽约（2023年）。https://doi.org/10.1145/3611643.3616271。https://doi.org/10.1145/3611643.3616271

Wang, Y.、Ye, W.、Guo, P.、He, Y.、Wang, Z.、Tian、

B., He, S., Sun, G., Shen, Z., Chen, S., 等：Sym- rtlo：利用大语言模型和神经元启发的符号推理增强寄存器传输级代码优化。arXiv 预印本 arXiv:2504.10369（2025年）

Wang, N., Yao, B., Zhou, J., Hu, Y., Wang, X., Guan, N.,Jiang, Z.: 来自验证的见解：通过结合测试平台反馈的强化学习训练 Verilog 生成大语言模型。arXiv 预印本 arXiv:2504.15804（2025年）

Xu, X., Feng, S., Ye, Y., Shen, G., Su, Z., Cheng, S., Tao, G., Shi, Q., Zhang, Z., Zhang, X.: 通过语义驱动的指令降权改进二进制代码相似性 Transformer 模型。载于：第 32 届 ACM SIGSOFT 国际软件测试与分析研讨会论文集。ISSTA 2023，第 1106–1118 页。计算机协会，纽约州纽约，美国（2023年）。https://doi.org/10.1145/3597926.3598121。https://doi.org/10.1145/3597926.3598121

Xu, C., Guo, H., Cen, C., Chen, M., Tao, X., He, J.：通过知识增强的大语言模型 LoRA 微调实现高效程序优化。《超级计算杂志》81(8)，1006（2025年）

Xiong, C., Liu, C., Li, H., Li, X.: HLSPilot：基于大语言模型的高层次综合。载于：第 43 届 IEEE/ACM 国际计算机辅助设计会议论文集，第 1–9 页（2024年）Xu, S., Li, Z., Mei, K., Zhang, Y.: AIOS 编译器：将大语言模型用作人工智能代理的自然语言编程和流程编程解释器。arXiv 预印本 arXiv:2405.06907（2024年）Xia, C.S., Wei, Y., Zhang, L.: 大型预训练语言模型时代的自动程序修复。载于：2023年 IEEE/ACM 第 45 届软件工程国际会议（ICSE），第 1482–1494 页（2023年）。https://doi.org/10.1109/ICSE48619.2023.00129Xu, Q., Yang, D., Zhang, L.: 代码优化思维链：结构化理解与自我检查。载于：2024 年第四届人工智能、大数据与算法国际会议论文集，第 425–430 页（2024年）Xu, X., Zhang, Z., Feng, S., Ye, Y., Su, Z., Jiang, N., Cheng, S., Tan, L., Zhang, X.: Lmpa：通过大语言模型与程序分析的协同改进反编译。arXiv 预印本 arXiv:2306.02546（2023年）Yang, C., Deng, Y., Lu, R., Yao, J., Liu, J., Jabbarvand, R., Zhang, L.: WhiteFox：由大语言模型赋能的白盒编译器模糊测试。《ACM 编程语言会议录》8(OOPSLA2)，709–735（2024年）

Ye, T., Ma, T., Zhang, X., Yu, H., Yin, J., Wang, W.: 面向问题的视角与代码优化的锚定验证。arXiv 预印本 arXiv:2406.11935 （ 2024 年） Yin,X.,Ni, C.,Nguyen,T.N., Wang,S., Yang,X.: Rectifier：通过大语言模型与纠正器进行代码翻译。arXiv 预印本 arXiv:2407.07472（2024年）Zhang, H., David, C., Wang, M., Paulsen, B., Kroening, D.: 使用大语言模型对整个项目进行可扩展且经过验证的代码翻译。《ACM 编程语言会议录》9(PLDI)，1616–1641（2025年）

Zhong, M., LYU, F., Wang, L., Geng, H., Qiu, L., Cui, H., Feng, X.：Comback：用于提升编译器后端开发效率的通用数据集。载于：第三十八届神经信息处理系统大会数据集与基准测试专题（2024年）。https://openreview.net/forum?id=vfju5hjrJw

Zhong, M., Lv, F., Wang, L., Qiu, L., Wang, Y., Liu, Y., Cui, H., Feng, X., Xue, J.：VEGA：使用预训练 Transformer 模型自动生成编译器后端。载于：第 23 届 ACM/IEEE 国际代码生成与优化研讨会论文集。CGO '25，第 90–106 页。计算机协会，纽约州纽约，美国（2025年）。https://doi.org/10.1145/3696443.3708931。https://doi.org/10.1145/3696443.3708931Zhang, Y., Song, W., Ji, Z., Danfeng, Yao, Meng, N.：大语言模型生成安全测试的效果如何？（2023年）。https://arxiv.org/abs/2310.00710Zhai, Y., Yang, S., Pan, K., Zhang, R., Liu, S., Liu, C., Ye, Z., Ji, J., Zhao, J., Zhang, Y，等：使张量语言模型能够协助生成 {高性能}张量程序以用于深度学习。载于：第 18 届 USENIX 操作系统设计与实现研讨会（OSDI 24），第 289–305 页（2024年）Zhang, S., Zhao, J., Xia, C., Wang, Z., Chen, Y., Cui, H.：将编译器语义引入大语言模型作为编程语言翻译器：从 C语言 到 x86 汇编的案例研究。载于：Al-Onaizan, Y., Bansal, M., Chen, Y.-N.（编）：计算语言学协会 EMNLP 2024 论文集，第 996–1011 页。计算语言学协会，佛罗里达州迈阿密，美国（2024年）。https://doi.org/10.18653/v1/2024.findings-emnlp.55 。https://aclanthology.org/2024.findings-emnlp.55/Zhang, S., Zhao, J., Xia, C., Wang, Z., Chen, Y., Feng, X., Cui, H.：LEGO-Compiler：通过翻译可组合性增强神经编译（2025年）。https://arxiv.org/abs/2505.20356

Zhang, Q., Zhang, T., Zhai, J., Fang, C., Yu, B.,

Sun，W., Chen，Z.：关于大型语言模型在软件工程中应用的批判性综述：以 ChatGPT 和自动程序修复为例。arXiv 预印本 arXiv:2310.08879（2023年）