大语言模型在代码生成中的应用：挑战、技术、应用与未来

尹超
2023K8009926003
人工智能 2023 级

2025 年 5 月

Abstract

大语言模型（LLMs）通过自然语言处理技术，使用户能够以人类语言生成可执行代码，显著降低了编程门槛。本文探讨了 LLMs 在代码生成中的技术原理、面临的挑战、应用场景及未来发展趋势。基于可靠的学术论文和技术博客，我们分析了 LLMs 如何通过 Transformer 架构和大规模代码数据集实现代码生成，讨论了资源需求、代码错误、偏见和安全风险等挑战，并展示了其在代码补全、调试、翻译和低代码平台中的应用。未来，LLMs 预计将通过改进训练方法和模型架构进一步提高准确性和安全性。本文通过数据和案例（如 HumanEval 和 HumanEval+ 基准测试）以及多张图表（模型性能对比、错误类型分布、代码生成流程图、性能随时间改进）增强可读性，为研究者和开发者提供全面参考。

1 引言

大语言模型（LLMs）是基于深度学习的自然语言处理模型，能够理解和生成类人文本。近年来，LLMs 在代码生成领域的应用显著增长，使非专业人士也能通过自然语言描述生成可执行代码。例如，GitHub Copilot 通过理解用户输入的注释或代码片段，自动生成完整的函数实现，研究表明其可将数据预处理任务的耗时从小时缩短到分钟。本文系统探讨 LLMs 在代码生成中的技术原理、挑战、应用场景及未来发展，基于可靠来源提供深入分析。

1.1 LLMs 的发展

自 2017 年 Transformer 架构提出以来，LLMs 经历了快速发展。早期模型如 GPT-3 拥有 1750 亿参数，而 2024 年的模型如 Llama 3.1 405B 进一步扩展了规模。这些模型通过在包含数十亿行代码的公开数据集（如 GitHub 代码库）上训练，展现出强大的代码生成能力。专门为代码生成设计的模型，如 OpenAI 的 Codex 和 Meta 的 Code Llama，进一步优化了性能，使编程更加普及。

2 技术原理

2.1 Transformer 架构

LLMs 通常基于 Transformer 架构，这是一种通过自注意力机制捕捉序列数据关系的神经网络。Transformer 由多个层组成，包括自注意力层、前馈层和归一化层，能够高效处理长序列数据。自注意力机制允许模型关注输入序列中的关键部分，从而理解复杂的代码结构。代码生成模型如 Code Llama 和 Codex 在训练时使用包含数十亿代码行的数据集，以学习编程语言的语法和语义。

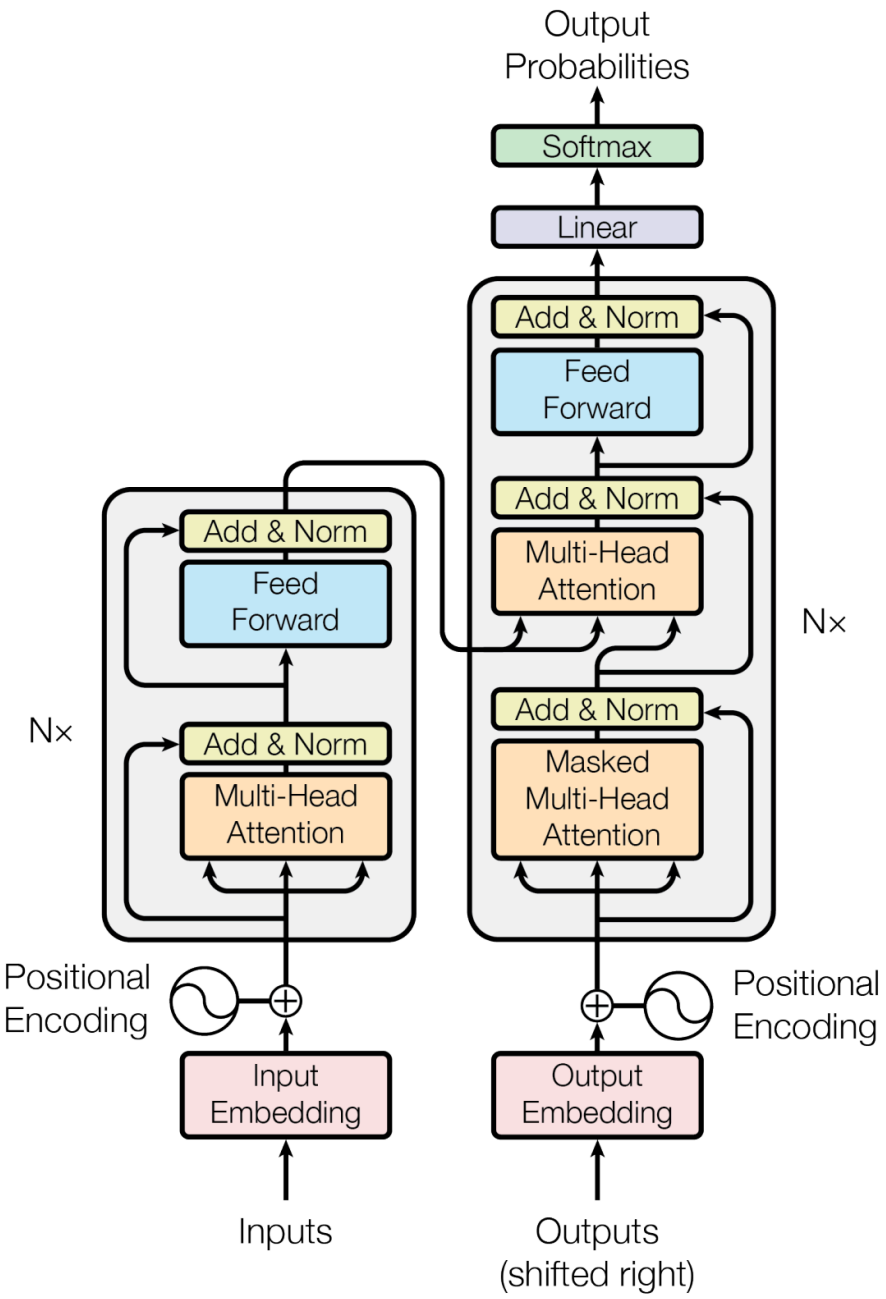

Figure 1: Transformer 架构示意图

2.2 代码生成流程

LLMs 生成代码的过程包括以下步骤：

1. **理解提示**：分析用户输入的自然语言描述，提取意图和需求。例如，用户输入“写一个排序函数”，模型识别排序算法的需求。
2. **检索模式**：从训练数据中检索相关代码模式，如快速排序或冒泡排序的实现。
3. **组装片段**：将检索到的代码片段组合成连贯结构，确保语法正确。
4. **生成代码**：输出符合语法的可执行代码，并根据上下文调整细节。

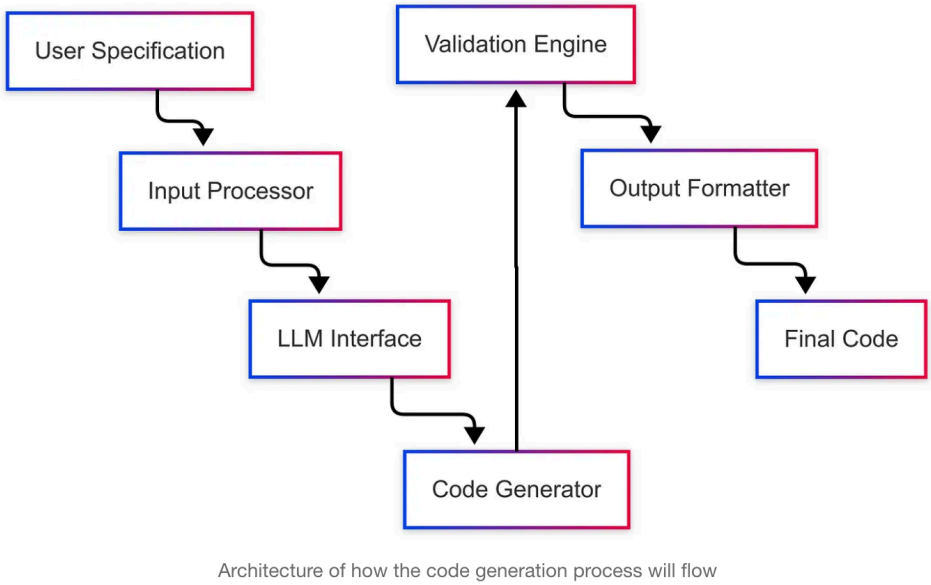

Figure 2: 大语言模型代码生成流程图

3 挑战

3.1 资源需求

训练 LLMs 需要大量计算资源。例如，Llama 3.1-8B 模型需要 700 万 GPU 小时，而 405B 模型需要 3100 万 GPU 小时。这不仅导致高昂的成本，还对环境造成影响，如碳排放问题。量化技术（如 OmniQuant 4 位量化，困惑度 5.97）虽然降低了推理成本，但训练成本仍是中小型组织的障碍。

3.2 语法与语义错误

生成的代码常包含错误。研究显示，在 557 个代码片段中，约 40% 存在语法错误，而在 APPS+ 基准测试中，语义错误率超过 50%。例如，ChatGPT 生成的 Java 代码中有 46.4% 存在非法索引错误。研究表明，功能性错误（Functional Bugs）占比最高，其次是运行时错误（Runtime Bugs），语法错误（Syntax Bugs）占比最低。图??展示了错误类型的分布。

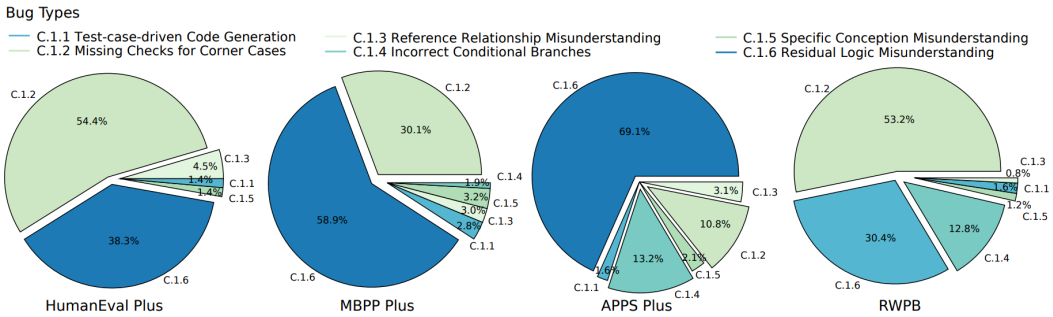

Figure 3: LLM 生成代码中错误类型的分布

3.3 偏见

LLMs 在多语言任务中表现出偏见。例如，中文指令下 pass@1 得分下降 17.2%，CodeLlama-34B 在 Java 任务中下降 37.8%。这可能源于训练数据中英语代码占主导地位。此外，社会偏见也存

在，如 Codex 的 CBS 得分高达 82.64%，可能导致不公平的代码生成结果。

3.4 安全风险

LLMs 生成的代码可能包含安全漏洞。研究表明，Copilot 生成代码的 40% 不安全，81% 的 2049 个代码库存在漏洞。例如，生成的代码可能包含 SQL 注入或缓冲区溢出等问题，这对生产环境部署构成挑战。

4 应用场景

4.1 代码生成与补全

工具如 GitHub Copilot 和 Code Llama 广泛用于代码补全。Code Llama-34B 在 HumanEval 上达到 53.7% 的 pass@1 得分，与 ChatGPT 相当。根据 GitHub 的研究，使用 Copilot 的开发者完成任务的速度提高了 55%（1 小时 11 分钟对比 2 小时 41 分钟），完成率为 78%，而未使用 Copilot 的开发者为 70%。此外，73% 的开发者表示 Copilot 帮助他们保持专注，87% 认为它减少了重复任务的心理负担。

4.2 高级代码生成与搜索

AlphaCode 在 Codeforces 竞赛中排名前 54.3%，解决 34.2% 的 CodeContests 问题。RepoRift 在 CodeSearchNet 上实现 78.2% 的 Success@10 得分，展示出强大的代码搜索能力。这些工具通过理解复杂需求生成高级代码，适用于编程竞赛和大型项目。

4.3 调试与翻译

GPT-4 在 LeetCode 上成功率超过 90%，Codex 修复 Python 错误的能力比 Java 高 50%。Flourine 在 8160 次翻译中实现 47% 的准确率，支持跨语言代码转换，如将 Python 代码转换为 Java。

4.4 集成到开发工作流

LLMs 正被集成到各种开发工具中，如 IDE、CI/CD 管道等。例如，GitHub Copilot 直接集成到 Visual Studio Code 中，提供实时代码建议。此外，LLMs 被用于自动化测试、代码审查和文档生成。例如，LLMs 可以生成测试用例或分析代码以检测潜在错误，从而提高开发效率。

4.5 低代码平台

LLMs 也被集成到低代码平台中，使非专业开发者能够通过自然语言描述创建应用程序。例如，Low-code LLM 框架允许用户通过图形界面与 LLM 交互，设计工作流而无需编写复杂提示。这为“公民开发者”提供了创建复杂应用的可能性，显著降低了技术门槛。

5 未来发展趋势

未来，LLMs 在代码生成中的发展将集中在以下方面：

- **提高准确性**：通过改进训练数据和模型架构减少错误和幻觉。
- **增强上下文理解**：更好地理解代码库和开发者意图，特别是在复杂项目中。
- **实时反馈**：集成到开发环境中，提供即时错误检测和建议。
- **多语言支持**：提升对低资源语言的生成能力，减少偏见。
- **安全性改进**：开发更安全的代码生成方法，降低漏洞风险。

此外，专用 LLMs（如针对金融或医疗领域的模型）和 CI/CD 管道的广泛集成将推动开发流程的自动化。新兴基准测试如 BigCodeBench 表明，顶级模型如 GPT-4o 在更复杂任务上的 pass@1 得分仅为 61.1%，显示出未来改进的空间。

6 数据与案例

6.1 基准测试概述

HumanEval 是评估代码生成模型的标准基准测试，包含 164 个编程问题，测试模型的函数正确性。HumanEval+ 是其扩展版本，增加了更多测试用例以提供更严格的评估。此外，BigCodeBench 等新兴基准测试通过更复杂的任务评估模型的实际编程能力。

6.2 HumanEval 基准测试

以下是部分顶级模型在 HumanEval 上的 pass@1 得分：

模型	Pass@1 (%)
O1 Preview	92.4
GPT-4o	90.2
Llama 3.1 405B	89.0
Grok-2	88.4
Claude3 Opus	84.9

Table 1: 顶级 LLMs 在 HumanEval 上的 pass@1 得分

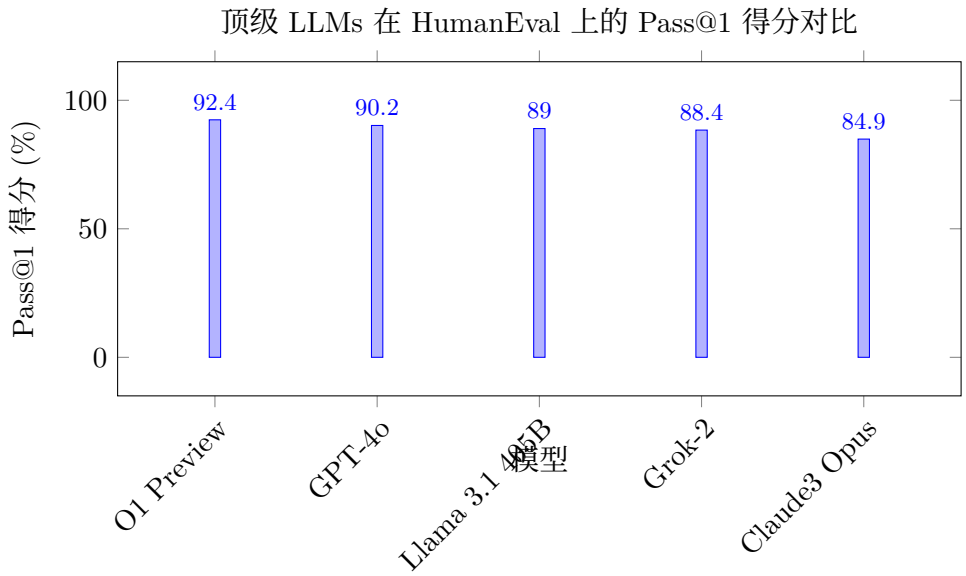

Figure 4: 顶级 LLMs 在 HumanEval 上的 Pass@1 得分对比（来源：BRACAI HumanEval Benchmark）

6.3 HumanEval+ 基准测试

HumanEval+ 通过增加测试用例提高了评估难度。以下是部分模型在 HumanEval+ 上的 pass@1 得分：

模型	Pass@1 (%)
O1 Preview	89.0
Qwen2.5-Coder-32B-Instruct	87.2
GPT-4o	87.2
DeepSeek-V3	86.6
Gemini 1.5 Pro	79.3

Table 2: 顶级 LLMs 在 HumanEval+ 上的 pass@1 得分

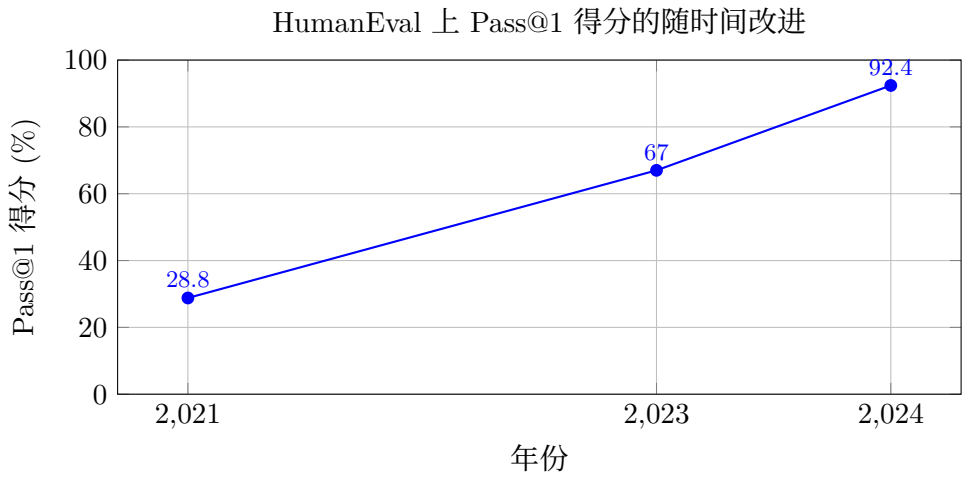

Figure 5: HumanEval 上 Pass@1 得分的随时间改进（来源：相关文献）

6.4 性能随时间改进

自 2021 年 HumanEval 发布以来，LLMs 在代码生成任务上的性能持续提升。图**??**展示了 HumanEval 上 pass@1 得分的随时间改进。

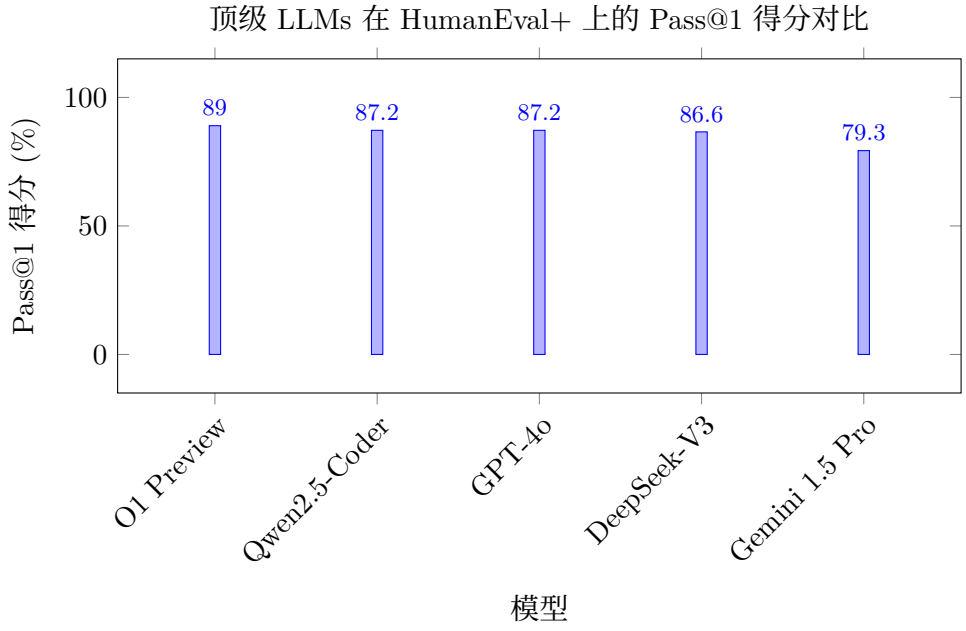

Figure 6: 顶级 LLMs 在 HumanEval+ 上的 Pass@1 得分对比（来源：EvalPlus Leaderboard）

7 结论

大语言模型在代码生成领域的应用显著提高了开发效率，使非专业人士也能参与编程。然而，高资源需求、代码错误、偏见和安全风险仍是主要挑战。未来，通过改进训练方法、增强上下文理解和安全性，LLMs 将在软件开发中发挥更大作用。

8 参考文献

- Attention is All You Need - Vaswani et al.
- What's Wrong with Your Code Generated by Large Language Models? - Dou et al.
- Low-code LLM: Graphical User Interface over Large Language Models - Cai et al.
- Evaluating Large Language Models Trained on Code - Chen et al.
- HumanEval Benchmark - BRACAI
- EvalPlus Leaderboard, HumanEval+ version 0.1.10
- Research: Quantifying GitHub Copilot's Impact on Developer Productivity and Happiness - GitHub Blog
- Code Llama: Open Foundation Models for Code - Meta AI
- BigCodeBench: The Next Generation of HumanEval - Hugging Face Blog