大语言模型在软件工程上的应用和影响

黄一凡，符支正, 史言, 马新凯,

Abstract

最近发布的 LLM，如 OpenAI 的 ChatGPT4 和谷歌在 Bard 中使用的 BERT，使用将终端用户的计算机交互从“这里是一个可能找到你的问题的答案的地方列表”到“这里是对你的问题的语法建议答案，你的下一个问题是什么?”毫无疑问，LLM 也有辅助软件工程任务的用例，包括用编程语言训练的代码生成模型，比如 GitHub 的辅助。自 2022 年以来，软件工程社区对 LLM 一直享受的加速进步的反应各不相同，从考虑这些模型提供的万金油到“我们所知的编程和计算机科学教育的结束”。在本文中，在简要概述了 LLM 之后，我们将重点讨论 LLM 对软件工程师的挑战，以及将 LLM 合并到系统中以及协助完成软件开发任务的影响。

什么是大语言模型

大语言模型，如 GPT 系列、BERT 系列和 Transformer-XL 等，都是基于自然语言处理（NLP）的深度学习模型。这些模型通过在大量文本数据上进行预训练和微调，能够生成连贯的文本、进行情感分析、回答问题等。下面我们详细介绍大型语言模型的训练过程和常见模型结结构。

预训练

在预训练阶段，模型通常使用无监督学习的方式，在大量的文本数据（例如维基百科、书籍、新闻文章等）上进行训练。这个过程可以帮助模型学习到一般的语言知识，包括词汇、语法、语义等。预训练任务通常包括以下两种：

-自回归语言建模（Autoregressive Language Modeling）：预测给定上下文中的下一个单词，如 GPT 系列。简单来说，在自回归语言建模中，我们训练一个模型来预测给定上下文（之前的单词序列）下的下一个单词。通过逐个生成单词，自回归模型可以生成完整的句子和段落。自回归语言模型的核心思想是将文本序列的概率分解为条件概率的乘积。

给定一个文本序列 $W = w_1, w_2, ..., w_n$，我们可以将其概率表示为：$P(W) = P(w_1) * P(w_2|w_1) * P(w_3|w_1, w_2) * ... * P(w_n|w_1, ..., w_{n-1})$ 在这个公式中，$P(w_i|w_1, ..., w_{i-1})$ 表示在给定之前单词的条件下，第 i 个单词的概率。训练过程中，我们使用最大似然估计（MLE）来优化模型参数，使得给定上下文的情况下，下一个单词的概率尽可能地接近真实分布。(Radford et al. 2018)

掩码语言建模（Masked Language Modeling，MLM）：从输入序列中随机屏蔽一些单词，让模型预测被屏蔽的单词，如 BERT 系列。与自回归语言建模不同，在掩码语言建模中，我们首先从输入的文本序列中随机选择一些单词并将其替换为一个特殊的掩码符号（例如”[MASK]”）。然后，模型的任务是根据上下文（包括被掩盖单词之前和之后的单词）预测这些被掩盖的单词。给定一个文本序列 $W = w_1, w_2, ..., w_n$，以及一个被掩盖的单词位置 i，掩码语言建模的目标是最大化如下条件概率：$P(w_i|w_1, ..., w_{i-1}, w_{i+1}, ..., w_n)$ 表示在给定上下文单词的条件下，第 i 个单词的概率。(Devlin et al. 2018)

微调

在微调阶段，预训练好的模型会在具体的下游任务（如情感分析、命名实体识别等）上进行有监督学习。这个过程相当于在预训练好的模型基础上，为其定制特定的任务知识。微调主要是通过在有标签数据上进行训练，来调整模型的参数。

常见模型结构

大型语言模型的主要结构通常是基于 Transformer 架构的。Transformer 是一种自注意力机制（Self-

Attention）的神经网络结构，相较于传统的循环神经网络（RNN）和长短时记忆网络（LSTM），它在处理长距离依赖和并行计算方面具有优势。

GPT 系列（Generative Pre-trained Transformer）

GPT 是 OpenAI 研发的一系列预训练生成式语言模型。GPT 系列模型采用自回归语言建模任务进行预训练，只使用输入文本的前文信息进行预测。GPT 系列模型结构主要包括多层 Transformer 解码器（Decoder）。GPT-3 是目前最新的版本，拥有超过 1750 亿个参数，是目前最大的语言模型之一。

BERT 系列（Bidirectional Encoder Representations from Transformers）

BERT 是谷歌研发的一系列预训练双向语言模型。与 GPT 不同，BERT 采用掩码语言建模任务进行预训练，同时利用输入文本的前文和后文信息进行预测。BERT 模型结构主要包括多层 Transformer 编码器（Encoder）。BERT 具有多种不同大小的版本，如 BERT-Base、BERT-Large 等。

Transformer-XL

Transformer-XL（Transformer with extra-long context）是一种针对长文本序列训练的模型结构。与标准的 Transformer 不同，Transformer-XL 引入了循环机制和相对位置编码，从而能够捕捉更长距离的依赖关系。这使得 Transformer-XL 在处理长文本序列时具有更好的性能。

代码生成原理

可以看到，主流的大语言模型都是基于 Transformer 架构，基于该架构的模型在训练过程中都要遍历大量的文本数据集中学习并捕捉到各种各样的模式，这其中就包括编程语言、框架、库和算法的相关信息。这些数据包括了各种编程语言、库和框架的使用示例。通过观察这些示例，模型可以学会编写代码。并且，由于代码本身的特性，同一种语言的语法规则是固定的，这使得具有很强模式辨识能力的神经网络（特别是 Transformer 架构）可以检测到代码中的结构和语法规则。这使得模型能够生成符合语法规则的代码。

但是，仅仅了解语法规则不会生成令人满意的代码，而 Transformer 架构的大模型在理解上下文信息的能力十分优秀，这使得如果用户提供了一个关于编写特定功能的代码的描述，模型可以理解这个描述并生成相应的代码。值得注意的是，优秀的代码的生成需要依赖于有意义的上下文和合理的代码功能描述，否则模型不能很好地发挥作用。最后，这些模型通常具有很好的迁移学习能力，这意味着它们可以在一个领域（如自然语言处理）中学到的知识应用到另一个领域（如编程）。这使得模型能够生成不同编程语言和框架的代码。

大语言模型在软件工程方面的应用

改善需求的挖掘与建模过程

需求的挖掘与建模在软件工程中起着关键作用，它涉及到准确理解用户需求并将其转化为清晰的需求文档。传统的需求工程方法依赖于人工收集和分析需求信息，这可能受限于个人经验和主观偏见。然而，随着大语言模型的出现，如 GPT，我们可以利用其强大的自然语言处理能力来改善需求的挖掘和建模过程。

通过引导 GPT，我们可以指导模型生成高质量的需求文档。这种引导可以通过向模型提供关键信息、示例需求或特定的问题来实现。通过在输入中引入领域相关的上下文和约束条件，GPT 能够生成更具准确性和完备性的需求描述。由于 GPT 所接触到的数据远比我们个人的经验丰富，它能够综合并学习大量的语言模式和领域知识，因此在需求的完备性方面，其表现往往更为出色。

此外，GPT 生成的需求文档在一致性、规范性等方面的质量通常优于一般人的写作水平。由于 GPT 在训练过程中接触到大量的语言数据，并学习到了语法、用词和句式的规律，它能够生成具有一致性和规范性的文本。这对于需求文档的可读性、可理解性和可维护性都具有重要意义。通过使用 GPT 生成的文档，我们可以减少人为错误和模糊性，并提高整个软件开发团队对需求的一致理解。

辅助编程、测试代码生成和运维任务

随着大型语言模型训练数据规模的增大，模型在处理更长序列和更复杂指令上的表现也越来越好，这使得它在软件编程领域的适用性不断提高。

首先，大型语言模型可以成为软件工程师的有力辅助工具。通过与 LLM 进行交互，工程师可以提出问题、请求代码示例或寻求算法实现的建议。模型可以理解并回应这些请求，通过学习大量的训练数据和模式，提供相关的代码片段、技术文档或设计方案，帮助工程师更高效地解决问题。

其次，大型语言模型可以用于测试代码生成。软件测试是开发过程中至关重要的一环，而 LLM 可以根据其训练数据中的经验和模式，在给定的测试需求和条件下，生成针对特定情况的测试代码。这种自动生成的测

试代码可以提高测试覆盖率，并帮助发现潜在的代码错误或边界情况，从而减轻了开发人员的工作负担。

此外，大型语言模型还可以在软件运维方面发挥作用。运维人员经常需要解决复杂的系统配置、故障排查和性能优化等问题。LLM 可以根据给定的环境和指令，提供运维方案、配置建议或故障处理的步骤。通过利用 LLM 的智能能力，运维人员可以更快速地解决问题，提高系统的可靠性和稳定性。

促进团队中专业与非专业人士的协作

大语言模型提供了非专业人士参与软件构建的机会。传统上，软件开发主要由专业编程者完成，需要具备深厚的编程知识和技能。然而，大语言模型的出现使得非专业人士也能够通过自然语言交互功能与大语言模型进行合作，而无需深入了解编程语言和技术细节。这种非专业人士的参与扩大了软件开发的参与范围，为团队带来更多创新思路和灵感。

大语言模型促进了专业编程者和非专业人士之间的有效沟通。在团队协作中，专业编程者与非专业人士之间的技术差异可能导致沟通障碍和误解，然而，大语言模型可以作为一个中介，帮助专业编程者和非专业人士之间进行有效的交流。非专业人士可以使用自然语言向模型提出问题、解释需求，并获得相应的技术建议和解决方案。专业编程者则可以通过理解大语言模型生成的回复，更好地理解非专业人士的需求和意图。这种有效的沟通提高了团队的协作效率和成果质量。

此外，大语言模型还可以在团队协作中充当知识库和学习资源。它可以存储和提供大量的技术文档、编码示例和最佳实践等信息。专业编程者和非专业人士可以向大语言模型提问，并获取相关的知识和学习资源。这种即时的、个性化的学习方式能够提升团队成员的技术能力和知识水平，促进团队的长期发展。

看向 LLM 的未来——为什么看好 LLM 和 LLM 软件工程

为什么在 ChatGPT 为代表的 MML 发布后，众多的领域当中，首当其冲的是软件方向呢?

当前软件开发的范式

除去较为灵活和抽象的“架构设计”部分，流程中“产品设计”和“运维”主要涉及对自然语言的分析处理，“开发”涉及自然语言（即需求和方案）转化为编程语言（高质量的结构和关系），“测试”和“交付”主要涉及对编程语言的分析处理。

可见在软件开发流程中主要运用的是自然语言和编程语言，它们都属于语言类知识，这类知识相对静态，而且数量巨大，只用 1000 万到 1 亿单词的语料，就能让 LLM 学好句法语义等语言学知识。(Zhang et al. 2020) 这使得 LLM 和软件开发十分契合。而且早在 ChatGPT 发布之前，机器的数据分析和处理能力就已经远超人类了。这也是为什么 LLM 一经发布就能被快速部署到软件开发的全流程中。

以 GPT 为代表的 MML 也给我们提供了很大的扩展性和想象的空间，比如最近 ChatGPT-4 增加了识别用户发送的图片的功能。随着 GPT 的发展，未来，很多独立存在的研究领域（图像、多模态……）将被纳入 LLM 的技术体系里面，逐渐消失。同时，随着预训练规模不断扩大，众多科学领域将非常有可能在 MML 中实现交融，成为其中的一部分。这种趋势使得软件开发过程的 AI 部署可以整合在一起，改变现在各司其职的现状，使得效率更高。

未来软件开发的范式

LLM 对于软件开发来讲，可能影响到的不仅仅是开发者。对于现有范式来讲，LLM 提高了开发者的开发效率，但随着 LLM 的发展，编程——这一个将计算机用户和计算机分隔开的墙壁也许会被推倒。

当然，让每个人具有独立开发软件的能力是不现实的，但是计算机用户使用计算机的方式可能会发生巨大的改变。比如，现在我们使用视频编辑软件的时候，可能只能使用软件所提供的固有功能，对于一些用户的新需求，可能需要各种反馈渠道反馈、汇总，经过一个周期发布新版本，对于某些较少的需求可能不会更新，而有些更新内容并不是所有用户都想要的，这就导致程序越来越臃肿，功能体验上的提升却不多。但是在 LLM 的加持下，一次性脚本、GUI 变成了可能。类比被很多开发者喜爱的 VIM 编辑器，其很重要的一点就是轻量、可定制。这样一来就极大地提高了软件的灵活性，这将改变未来软件开发的范式。

目前，已经有一些如 Low-code LLM 的方法 (Cai et al. 2023) 来提高 LLM 在处理复杂问题的准确性，相信这类 prompt 的方法知识 LLM 发展过程中的一个过渡，随着 LLM 的不断发展，即使面对复杂问题的时候，LLM 依然可以展现出很高的正确性。

几点应用中的不足

GPT-4 在代码纠错和代码补全的实时反馈场景中可能存在一些限制，但其主要局限性在于无法提供即时的、逐步的反馈。由于 GPT-4 的工作方式是基于预训练的模型，它需要将完整的代码输入后才能生成结果，而无法像人类一样逐行逐步的进行纠错和补全。这可能

对那些需要及时调试和交互式开发的情况造成一些不便。

GPT-4 的一个挑战是上下文序列的有限性，这使得它难以顾全大中型项目的全部上下文。由于模型对于输入序列的长度有限制，当处理超长的项目或大规模代码库时，可能会遇到限制或性能问题。长期依赖性和跨文件的上下文关系也能难以完全捕捉和理解，这可能导致一些细节被忽略或误解。

在特定行业问题的解决中，GPT-4 对于领域知识的理解可能存在一定的限制。尽管 GPT-4 在广泛的预训练中获得了大量的语言知识，但对于特定领域的专业术语、约束关系和行业特定的问题，它可能需要额外的培训和领域特定的数据来进行适应和理解。对于石油、煤炭安全、核电站和医疗等行业，这些领域都拥有复杂的专业知识和规范，要求对细节和特定上下文的准确理解。因此，对于 GPT-4 是否能够全面了解和掌握这些领域知识，持有保留态度是合理的。

discussion

在过去的软件开发中，我们经历了 1.0 的工程化和标准化变革，强调流程的重要性以确保结果的可靠性和质量。然而，随着时间的推移和技术的进步，我们逐渐意识到开发过程中人的角色和潜力的重要性。这引领着软件工程 2.0 的时代，以人为本，充分发挥程序员、测试人员和产品经理的创造力和能力。

然而，随着我们进入数字化时代，计算机系统和工具的发展已经开始理解我们存储的信息。这是一个巨大的突破，我们可以通过使用自然语言处理和生成技术，例如 GPT（Generative Pre-trained Transformer），进行人机对话。这意味着我们可以与计算机进行自然的对话，而不仅仅是通过特定的界面和命令来进行交互。这对软件开发产生了深远的影响。

在软件开发中，人工智能可以扮演重要的角色。AI 可以生成各种开发文档，如需求文档、功能规格说明书、代码、测试用例和测试脚本。这意味着我们可以借助 AI 的能力来加速和改善软件开发过程，实现更高效的持续交付。AI 可以分析大量的数据和信息，并生成符合要求和规范的文档和代码，减轻开发人员的负担，提高开发效率。

这一切标志着软件工程的新时代，即软件工程 3.0 的到来。软件工程 3.0 强调人与机器的紧密协作和相互补充，通过利用 AI 和自然语言处理等技术，实现更智能化、自动化的软件开发过程。这使得软件开发团队能够更加专注于创造性的工作和解决复杂的问题，而不是被重复的、机械的任务所束缚。

然而，我们也要意识到，虽然 AI 在软件开发中的应用带来了巨大的潜力和机会，但它并不是万能的解决方案。人类的专业知识、经验和判断力仍然是不可或缺的。AI 可能存在理解限制、缺乏领域专业知识等问题，因此，在应用 AI 时需要谨慎并结合人类的智慧和判断力。

conclusion

本文剖析了 LLM 的工作原理，并以此指出它在软件工程领域，尤其是代码生成、NLP 与编程语言的转换方面有着显著的优势。之后具体探讨了 LLM 在当前软件开发流程中的应用，进一步阐释 LLM 对当前软件开发范式的巨大影响以及一些不足之处。并且在 LLM 加持下，我们可以看到未来软件开新发范式的更多可能性。

参考文献

Cai, Y.; Mao, S.; Wu, W.; Wang, Z.; Liang, Y.; Ge, T.; Wu, C.; You, W.; Song, T.; Xia, Y.; et al. 2023. Low-code LLM: Visual Programming over LLMs. arXiv preprint arXiv:2304.08103.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

Radford, A.; Narasimhan, K.; Salimans, T.; Sutskever, I.; et al. 2018. Improving language understanding by generative pre-training.

Zhang, Y.; Warstadt, A.; Li, H.-S.; and Bowman, S. R. 2020. When do you need billions of words of pretraining data? arXiv preprint arXiv:2011.04946.