Aplicação de Grandes Modelos de Linguagem em Engenharia de Software: Delírio ou Realidade?

Gabriel Tonioni Duarte
Departamendo de Ciência da Computação (DCC)
Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte, Brazil
gabriel.tonioni@dcc.ufmg.br

Marco Tulio Valente
Departamendo de Ciência da Computação (DCC)
Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte, Brazil
mtov@dcc.ufmg.br

***Abstract*—Este trabalho busca alcançar uma compreensão bem embasada do cenário atual de pesquisa em aplicação de grandes modelos de linguagem (LLMs - large language models) em engenharia de software. São discutidos aspectos centrais dos LLMs, como o surgimento de habilidades emergentes, os métodos e estratégias empregados para melhor aplicá-los na engenharia de software, a partir da perspectiva da geração automatizada de testes de unidade, e os desafios que precisam ser enfrentados no desenvolvimento das aplicações e das pesquisas.**

***Index Terms*—Large Language Models, Automated Unit Test Generation, Software Testing**

I. Introdução

O lançamento do ChatGPT no final de 2022[1] demarca um ponto de virada na percepção e discussão da Inteligência Artificial (IA). A capacidade da aplicação de gerar respostas coerentes às mais diversas perguntas dos usuários estimulou a imaginação sobre o futuro da IA e infindáveis discussões sobre a capacidade (ou falta) de entendimento acerca do mundo dos grandes modelos de linguagem (LLMs - large language models).

Além da geração de texto, a aplicação também consegue gerar respostas contendo código. Isso desencadeou uma série de iniciativas, dentro e fora da academia, para desenvolver aplicações baseadas em LLMs para resolver tarefas de geração de código. Nos anos de 2023 e 2024 foram experimentadas aplicações de geração de código, e de outros artefatos relacionados ao desenvolvimento de software, em todas as etapas do ciclo de desenvolvimento de software [1], [2]. Já fora do ambiente acadêmico, verificou-se o surgimento de diversas empresas e ferramentas relacionadas à geração de código[2].

A relação do campo da IA com grandes promessas não é boa. Historicamente, dois períodos, conhecidos como invernos da IA[3], mostram isso. Esses dois períodos foram marcados por grandes promessas que não foram cumpridas, ainda quando o paradigma dominante do campo era a IA simbólica. A quebra de expectativas gerou uma forte retirada de investimento das pesquisas em IA.

Posteriormente a esses momentos de decepção, o campo da IA conseguiu um avanço significativo, trabalhando a partir de um outro paradigma. As pesquisas em aprendizado de máquina geraram retorno em tarefas específicas. Esse novo paradigma atingiu seu ápice com a proeminência do aprendizado de máquina profundo a partir da segunda década do século xxi, que conseguiu avanços significativos em tarefas até então difíceis de serem conquistadas pelos algoritmos de aprendizado de máquina anteriores, como classificação de imagens, transcrição de falas, transcrição de escrita à mão, tradução, conversão de texto para fala, assistentes digitais, direção autônoma, etc [3].

Na terceira década do século XXI novos avanços consideráveis foram feitos, com os LLMs alcançando um salto qualitativo no desempenho em algumas tarefas de processamento de linguagem natural (NLP - natural language processing), como a criação de sistemas de diálogo críveis, capacidade de tradução comparável à humana e entendimento de linguagem natural comparável ao humano [3].

O treinamento desses modelos em quantidades massivas de dados, sem ter o foco em uma tarefa específica, podem gerar certas ilusões a respeito do que esses modelos estão de fato fazendo. Isto, juntamente com o problema da explicabilidade dos modelos de aprendizado profundo, ressuscitou as grandes promessas especulativas, capazes de captar investimentos. Porém, diferente dos invernos da IA anteriores, dessa vez não se está buscando mecanismos explícitos para atacar traços do comportamento inteligente, como raciocínio, representação de conhecimento e planejamento. Avanços aconteceram na área de NLP, e alguns especulam que esses outros traços do comportamento inteligente estão sendo, de alguma forma, contemplados [4].

Portanto, é importante tornar explícito e discutir qual a visão adotada sobre os modelos, quais suas hipóteses e quais suas evidências. Pois isso impacta na robustez das estratégias, no sucesso de suas aplicações e nos ajuda a encontrar os trabalhos de pesquisa que têm uma contribuição científica confiável e separá-los do ruído.

O intuito deste trabalho é discutir e analisar as aplicações de LLMs na área de engenharia de software, focando na tarefa de geração de testes de unidade. Discutiremos os métodos empregados e as estratégias subjacentes, buscando entender em quais hipóteses acerca do comportamento dos LLMs eles estão se apoiando. Iremos ressaltar hipóteses competidoras

[1]https://openai.com/index/chatgpt/
[2]https://www.cognition.ai/blog/introducing-devin
[3]https://en.wikipedia.org/wiki/AI_winter

e as evidências a favor e contrárias a cada uma, além das implicações disso sobre os diferentes métodos estudados até agora.

Tendo isso em vista, este trabalho examina aplicações de LLMs, considerando os debates em aberto e testes empíricos relacionados aos LLMs e, a partir disso, avalia os resultados alcançados até agora e as perspectivas futuras. Para isso, na seção II é feita uma breve descrição dos modelos, uma listagem de diferentes visões sobre eles e o que sabemos sobre seu comportamento. Em seguida, na seção III são apresentadas algumas aplicações de LLMs em tarefas de engenharia de software, com o foco na geração de testes de unidade, seus métodos e estratégias. A seção IV apresenta os trabalhos relacionados. A seção V discute as aplicações desenvolvidas e expectativas futuras do campo. A seção VI conclui o texto.

II. Background

LLMs são um conjunto de redes neurais profundas, construídas com base na arquitetura de transformadores, e uma das principais tecnologias de IA Generativa, que utiliza modelos capazes de gerar novos dados, reproduzindo padrões e estruturas presentes no conjunto de treino. Os modelos são treinados em dados gerais provenientes da web de forma não supervisionada, ou seja, seus treinamentos não têm como foco a realização de uma tarefa específica, o que permite aplicá-los em diversos domínios.Por isso, é dito que esses modelos são pré-treinados. Em alguns casos, é feito um treinamento supervisionado, posterior ao pré-treino, que se chama de fine tuning, para que o modelo desempenhe melhor em uma tarefa específica. Um tratamento mais detalhado sobre esses modelos pode ser visto em [5]. O nosso foco está em destacar as diferentes formas de se pensar sobre esse modelos, suas capacidades e limitações.

A. As Diferentes Visões

Existem visões divergentes sobre as capacidades desses modelos e sobre o que eles aprendem. As divergências surgem da necessidade de certo nível de especulação do entendimento desses modelos, já que não sabemos exatamente como eles chegam a uma determinada resposta. Dito isso, algumas visões defendidas são[4]: I) LLMs são papagaios estatísticos; II) LLMs conseguem criar modelos internos do mundo, assim como humanos, que são o reflexo da experiência e conhecimento humano, representados de forma textual na web; III) LLMs não possuem capacidade inteligente alguma que seja análoga às capacidades humanas, como raciocínio deliberado. Mas, sua capacidade de memorizar e recuperar informações é notável.

A escolha e defesa de uma dessas visões impacta nas expectativas do potencial dos modelos e, consequentemente, no alocamento de investimentos, nas direções de pesquisa e na estratégia de avanço do campo de IA, seja aumentando a capacidade desses modelos, ou buscando soluções diferentes, como novas arquiteturas de rede neural.

[4]https://medium.com/aiguys/can-llms-really-reason-and-plan-50b0ac6addd8

B. O que Sabemos?

Existem mais questões em aberto do que certezas no nosso conhecimento do comportamento de redes neurais profundas e ,portanto, do de LLMs também. Não sabemos exatamente o que eles estão aprendendo ao configurar os seus bilhões de parâmetros. O que acontece no interior dos modelos é incerto. Porém, o que pode ser observado no exterior informa e dá suporte para as diferentes visões e hipóteses acerca do seu comportamento. O que acontece no exterior é observado através da medição do desempenho do modelo durante as iterações de treino e da medição de seu desempenho em benchmarks. Dessa forma, iremos discutir a seguir as evidências levantadas a partir de análises dessas medições, para que ao final da seção possamos formular a visão adotada por este trabalho sobre o comportamento dos LLMs.

1) Capacidade de Planejamento e Raciocínio dos LLMs: A capacidade de planejamento é vista como uma das características de um agente inteligente[5]. Os LLMs não apresentaram bom desempenho quando testados de forma sistemática em desafios de planejamento [8, 9]. Basicamente, esses desafios fornecem um cenário, ações e restrições para a realização das ações. O objetivo é determinar uma sequência de ações que levem de um estado inicial até um estado objetivo. Portanto, esse tipo de planejamento foge de questões de conhecimento comum, como planejar um roteiro de férias. Nesses casos, é difícil saber se o planejamento fornecido pelo LLM é fruto de memorização, por ser um tipo de informação comum no conjunto de dados, ou se é fruto de alguma forma de entendimento real. Os desafios de planejamento formais são uma boa evidência de que os LLMs atuais carecem desse traço do comportamento inteligente ou geram uma forma de entendimento que não é confiável e talvez seja incorreta. Ainda em testes com desafios formais de planejamento, os LLMs conseguiram melhor desempenho na tarefa de gerar um modelo para uma sequência de ações na linguagem de definição do domínio de planejamento (PDDL - planning domain definition language) [10]. Dessa forma, os LLMs podem ser melhor utilizados como auxiliadores na formulação de planejamentos, necessitando de humanos ou outros sistemas automatizados de planejamento para gerar um plano executável e correto.

2) Memorização, Generalização e Grokking: O problema central atacado pelo aprendizado de máquina é encontrar transformações, que, quando aplicadas aos dados de entrada, geram outras representações desses dados que facilitem a resolução de uma determinada tarefa, pela aplicação de certas regras (ou padrões) sobre essas representações. Essas transformações são encontradas utilizando-se redes neurais ao se definir um conjunto de pesos, também chamados de parâmetros, que definem essas transformações, em sucessivas iterações no processo de treinamento. No caso do aprendizado profundo, são feitas sucessivas transformações nos dados, dispostas em várias camadas de uma rede neural profunda.

[5]https://en.wikipedia.org/wiki/Artificial_general_intelligence#Characteristics

Então, são encontradas transformações dos dados de entrada utilizando-se um conjunto de treino e, posteriormente, elas são validadas em um conjunto de teste. Nesse processo existe um tradeoff central em aprendizado de máquina, que consiste em utilizar uma quantidade maior de parâmetros, permitindo mais possibilidades de transformações e, consequentemente, representações mais complexas que permitem a utilização de regras mais precisas, ou utilizar uma quantidade menor de parâmetros que encontrem representações e regras mais simples, mas que sejam mais gerais. Relacionado ao primeiro caso, pode acontecer o que se chama de overfitting, que consiste em encontrar representações e regras muito atrelados aos dados do conjunto de treino. Intuitivamente, podemos pensar que o modelo memoriza esses padrões, e quando é alimentado por um dado não visto no conjunto de treino, ele basicamente faz um chute baseado no que ele memorizou. Isso não é o ideal. Melhor seria se o modelo encontrasse regras que sejam generalizáveis à diversas situações que ele possa encontrar, tanto nos dados em que foi treinado quanto em dados novos.

Porém, uma rede neural, com muitos parâmetros, que memoriza as regras para um conjunto de treino finito não é necessariamente um fracasso. Foi observado um fenômenochamado de grokking em redes pequenas de transformadores, treinadas em conjuntos de dados sintéticos para resolver problemas de aritmética modular. Ao se continuar as iterações de treinamento, bem depois do ponto de overfitting ao dados de treino, aconteceu uma melhora no desempenho com os dados de teste [6]. Isso indica que alguma forma de generalização ocorreu. O estudo desse fenômeno pode ajudar na compreensão do comportamento de redes maiores, como os LLMs, elucidando quando esses modelos estão memorizando e quando eles estão generalizando. Outras questões importantes ainda permanecem, como o que causa essa transição de memorização para generalização, e qual é a generalização feita[6].

De qualquer forma, alguns trabalhos, como [7], mostram comportamentos dos LLMs que se assemelham mais a um aprendizado por memorização do que a um aprendizado por generalização. Neste caso, se existe no conjunto de treino a afirmação “A é B”, o LLM não consegue inferir do conjunto de treino a relação “B é A”. Mas, um ponto interessante, ressaltado ainda nesse artigo [7], é que o LLM consegue inferir a relação na ordem reversa caso ele tenha acesso a algum contexto fornecido na entrada.

3) Habilidades Emergentes: Em alguns trabalhos é dito que a principal diferença entre LLMs e modelos menores são as chamadas habilidades emergentes [5]. A noção de emergência diz respeito a comportamentos que surgem da interação das partes de um sistema, que só aparecem quando esse sistema atinge um certo nível de complexidade. No estudo de LLMs, a emergência assume a forma, um tanto similar ao que se definiu como grokking, de um aumento repentino e imprevisível no desempenho, quando este é visto em função da complexidade do modelo, que pode ser medida como a quantidade de parâmetros. Em [8], esse fenômeno de emergência nos LLMs foi contestado, ao mostrar que o comportamento do desempenho em função da complexidade do modelo assume um progresso previsível e suave ao se mudar a métrica de desempenho. Dessa forma, a emergência não teria relação com o funcionamento interno dos LLMs. Contudo, esse é um debate ainda em andamento dentro da comunidade científica.

O debate acerca desse fenômeno é importante para o desenvolvimento de métodos que consigam prever o comportamento dos LLMs. Já com relação ao desenvolvimento de aplicações baseadas nesses modelos, a discussão também tem importância, pois algumas das estratégias de engenharia de prompt se baseiam na suposição desse fenômeno[7].

4) Leis de Escalabilidade Neural: A discussão de habilidades emergentes está intimamente ligada com a escala dos LLMs. A arquitetura de transformadores permitiu que esses modelos atingissem uma complexidade de bilhões de parâmetros. A partir daí, foi possível observar, pela primeira vez, o fenômeno de emergênciam e surgiram todas essas discussões. Isso traz a pergunta: até onde o aumento de escala pode nos levar?

O que se chama de lei de escalabilidade neural diz, basicamente, que o desempenho dos modelos melhora quando se aumenta sua escala. Mais especificamente, o aumento da escala e a melhora no desempenho se relacionam por leis de potência [9]. Aumentar a escala consiste em aumentar a quantidade de parâmetros do modelo, aumentar o tamanho do conjunto de dados ou aumentar a capacidade computacional utilizada no treino. Se o desempenho for limitado por apenas um desses fatores, ele pode ser predito por uma lei de potência [9]. Porém, essa melhora no desempenho não significa que o modelo está ficando mais inteligente. Para entender esse ponto, será preciso avançar no debate sobre grokking e habilidades emergentes.

Para tornar o último ponto mais claro, a diferença entre emergência e leis de escalabilidade se dá no fato desse último apresentar uma relação bem definida entre aumento de escala e melhora no desempenho. Portanto, podemos prever qual será a melhora no desempenho com um aumento de escala de um LLM. Porém, não sabemos qual a relação entre o aumento de escala e o surgimento de novas capacidades [10]. Não sabemos nem se essa relação realmente existe. É possível que no futuro esse fenômeno de emergência, entendido da forma descrita na seção II-B3, seja observado também em modelos menores.

5) Avaliação e Benchmarks Automatizados: Avaliar LLMs e compará-los ainda é uma tarefa difícil de se fazer com confiança. Alguns rankings que listam vários modelos sofrem variações quando são introduzidos diferentes tipos de perturbações no método de avaliação que gera esses rankings [12], mostrando como eles são sensíveis. Outro problema é quando ocorre contaminação do benchmark, pois o LLM é treinado no conjunto de testes do benchmark [13]. Essa é uma área que ainda carece de soluções, portanto é importante ser

[6]https://pair.withgoogle.com/explorables/grokking/

[7]https://www.jasonwei.net/blog/emergence

criterioso para aceitar os resultados divulgados em artigos ou em outros lugares.

C. Qual visão adotar?

Diferentes visões sobre LLMs foram listadas na seção II-A. Feito um resumo dos principais pontos sobre o estudo de LLMs nas seções anteriores, podemos dizer que a visão I é mais conservadora, a visão II é mais desafiadora e a visão III pode ser colocada em uma posição intermediária entre as duas. Este trabalho não adota nenhuma dessas visões completamente. Para a análise da aplicação de LLMs em engenharia de software, feita até o momento, é considerado com qual dessas visões as estratégias e métodos desenvolvidos melhor se relacionam e no que isso implica no potencial e incerteza dessa aplicação. Seguimos com essa análise na seção III.

III. Aplicação de LLMs em Engenharia de Software

LLMs são modelos de propósito geral. Isso significa que esses modelos realizam várias tarefas diferentes, mesmo que elas não estejam representadas explicitamente no conjunto de treino. Essa flexibilidade transformou esses modelos em candidatos para a geração de diversos artefatos de engenharia de software.

O restante da seção faz uma descrição em alto nível da geração automatizada de testes de unidade e, depois, analisa como as características dos LLMs são exploradas para automatizar a geração de testes de unidade.

A. Geração Automatizada de Testes

Existem várias áreas da engenharia de software que podem explorar o potencial dos LLMs, cobrindo todo o ciclo de desenvolvimento de software [1], [2]. Este trabalho foca em pesquisas feitas sobre a geração automatizada de testes de unidade. Essa já é uma área de pesquisa bem consolidada. Várias ferramentas e técnicas já existiam antes do advento dos LLMs. Com isso, as novas aplicações baseadas em LLMs podem ser validadas, comparando-as com a referência definida por outras ferramentas de geração automatizada de testes. Outro ponto importante é que a área de teste de software proporciona várias métricas e métodos bem conhecidos para medir diferentes aspectos da qualidade de um conjunto de testes, como diversas métricas de cobertura e métodos para realizar testes de mutação. Além disso, o processo para obter esses resultados é automatizado por vários frameworks para várias linguagens de programação.

1) Desafios: Uma série de desafios para o desenvolvimento dessa área de pesquisa já são conhecidos [1]. O mais crítico deles é o desafio de corretude dos testes. Quando um teste é gerado e ele acusa um problema, i.e., o teste não passa, como saber se de fato existe algum problema no código, ou se o problema está na forma como o código é testado? Este problema também é conhecido como Problema do Oráculo. No caso de programadores humanos, o programador é o oráculo, pois ele tem capacidade de entender qual comportamento o sistema deve apresentar ao receber certo estímulo, ou, da perspectiva do programador, qual saída esperar para uma entrada fornecida. No caso de um teste gerado por uma ferramenta, seria necessário um oráculo automatizado. Daí surge o problema. Portanto, é fundamental que aplicações baseadas em LLMs forneçam métodos para tratar as respostas do modelo.

B. LLMs Utilizados

Existem três tipos principais de LLM, categorizados de acordo com as seguintes arquiteturas: I) encoder-only, II) encoder-decoder e III) decoder-only [1], [2], [5]. Intuitivamente, a arquitetura I tem como foco tarefas de entendimento, a arquitetura II tem como foco tarefas de geração e a arquitetura III une as duas tarefas [2], [5]. A aplicação de LLMs em tarefas de engenharia de software é dominada por modelos do tipo III [2].

C. Explorando as Capacidades dos LLMs

Existem duas formas de explorar LLMs para gerar testes de unidade: através de finetuning ou através de engenharia de prompt. No caso, a segunda forma é a mais adotada, pois não sofre dos custos de treinar o modelo, como construir e manter um conjunto de dados com unidades de código e testes.

Diferentes estratégias de montagem de prompt já foram empregadas para descrever a tarefa de geração de testes para LLMs [2]. Alguns padrões de prompt recorrentes acabam recebendo nomes. Os mais populares são conhecidos como few-shot prompting, zero-shot prompting e chain-of-thought. O foco aqui não é destacar algum desses padrões e descrevê-lo profundamente. A partir deste ponto, é dada maior atenção para o raciocínio envolvido na montagem do prompt para descrever uma tarefa de geração de teste.

Antes de falar sobre a montagem do prompt, vale a pena reforçar a tarefa alvo. O objetivo do LLM é gerar testes para uma base de código já existente. Não é objetivo do modelo fazer uma prática de desenvolvimento dirigido por testes (TDD - test driven development), na qual primeiro se escreve os testes, que servem como uma especificação para a posterior escrita do código de produção. Como discutido na seção II-B1, LLMs não parecem se dar bem com tarefas que envolvem planejamento e raciocínio, mas podem ter utilidade como auxiliares. Portanto, gerar os testes para código já existente é a tarefa mais apropriada.

Existe uma estratégia predominante para a montagem do prompt entre os trabalhos selecionados. Ela constrói um contexto que diz respeito à unidade de código que está sendo testada. O prompt que descreve a tarefa de geração de teste contém esse contexto [11], [12] [13]. No caso da testagem de um método, o contexto pode conter sua assinatura, seu corpo e suas dependências, por exemplo. Para construir um prompt com a descrição da tarefa e do contexto, é definido um template, que serve como modelo para a criação automatizada dos prompts.

Existem variações a respeito do template de prompts. O prompt pode ser uma descrição textual da tarefa e do contexto

[11] ou pode ser um trecho de código incompleto, para que o LLM complete a parte faltante [12], [13]. Também existem diferentes maneiras de representar e definir o contexto. Os elementos do contexto podem ser embutidos em partes específicas da descrição textual [11]. No caso do template de prompt que utiliza código, o contexto pode ser representado por comentários no código [12], ou pode ser incluído como código também, por exemplo, o código da classe que contém um método a ser testado [13]. Por fim, o contexto pode ser construído para conter a maior quantidade de informações possível [11] ou pode ser feito um processo de seleção das informações [12] ou uma quantidade fixa de informações pode ser definida para compor o contexto [13].

1) Cuidados com a Utilização de Prompts: LLMs possuem uma janela de tokens, que define um limite de tokens que podem ser fornecidos como entrada e que podem ser gerados pelo modelo como resposta. É possível definir limites de tokens separados para entrada e saída em alguns modelos, como o Codex, o GPT-3.5-Turbo e o StarCoder [13]. De qualquer forma, se muitos dos tokens da janela forem consumidos pelo contexto, o LLM pode retornar uma resposta incompleta, por não ter tokens suficientes sobrando. Se os limites forem separados, a especificação da tarefa fornecida no prompt pode ficar incompleta.

Uma estratégia para lidar com a janela de tokens é utilizar um contexto adaptável [11], [12], que estabelece uma quantidade de informações mínima e obrigatória para compor o contexto, e varia a quantidade de informações extras que são adicionadas para cada execução da tarefa. Para adotar essa estratégia é preciso definir quais informações serão adicionadas ao contexto, em um dado momento, e como. Uma possibilidade é adicionar informações extras, seguindo uma ordem de prioridade pré-estabelecida, até atingir o limite de tokens [11]. Outra possibilidade é utilizar vários prompts com combinações diferentes de informações extra no contexto [12].

Fornecer um contexto com muitas informações não necessariamente faz o LLM gerar testes de maior qualidade [13]. Além disso, um determinado tipo de informação pode ter uma contribuição maior para a eficácia da geração de testes [12]. Isso torna desejável a exploração de estratégias mais flexíveis para a geração do contexto, como a adotada por Schäfer et al. [12].

2) Desafios: A falta de entendimento completo sobre o comportamento dos LLMs dificulta o desenvolvimento de um método que consiga, garantidamente, extrair o pontencial desses modelos de forma ótima. É difícil saber com exatidão se uma boa resposta gerada pelo LLM é fruto do método de construção do prompt ou se é fruto do acaso. Avanços em pesquisas sobre habilidades emergentes, grokking e escalabilidade, que aprofundem nosso conhecimento sobre o comportamento desses modelos e como melhor controla-lo serão de grande valia para pesquisas de engenharia de software baseada em LLMs como um todo.

D. Validação dos Testes Gerados

LLMs não conseguem validar e executar planos em tarefas de planejamento [14]. Da mesma forma, em tarefas de geração de testes de unidade, LLMs não tem a capacidade para validar e executar os testes gerados. Alguns aspectos a se considerar para fazer a validação de um teste gerado são: sintaxe correta, compilação sem erros, execução sem erros, corretude [11]–[13] e verificar se o teste é flaky (um teste flaky falha ou passa de forma intermitente e imprevisível) [15]. Vamos abordar com mais detalhes a corretude dos testes:

- Corretude: mesmo um teste que não tem erros de sintaxe, compila sem erros, executa sem erros e tem boa qualidade pode não ser um teste correto. Esse aspecto de validação é mais sutil que os outros e está intimamente ligado com o Problema do Oráculo, que até o momento não tem solução. Para fugir desse problema, um teste pode ser considerado correto se o código que ele testa conseguir passar por ele [13]. A consequência dessa decisão é que os testes gerados verificam apenas eventuais regressões do sistema, desvios do comportamento funcional após uma modificação. Eles não conseguirão ajudar a encontrar bugs [1].

Como foi visto, a validação dos testes envolve uma série de checagens que são feitas sobre o teste gerado. O tempo gasto para fazer isso faz com que essas aplicações não sejam adequadas para gerar respostas em tempo real [16].

E. Validação da Aplicação

Além da validação dos testes, também é importante validar a aplicação como um todo. Para validar a aplicação isoladamente são considerados alguns aspectos: ausência de test smells, métricas de cobertura, utilização de APIs de frameworks de teste, utilização de asserts não triviais, proporção de testes corretos [11], [13]. Para obter resultados dos LLMs que cobrem esses aspectos são utilizados pacotes de software de repositórios públicos, como github e gitlab, e de gerenciadores de pacotes, como o npm para JavaScript [11], [12]. Também são utilizados conjuntos de dados contendo prompts, como o HumanEval, descrevendo problemas de programação para avaliar LLMs e conjuntos de dados contendo projetos de software open-source, como o SF110 [13].

Também é importante comparar o desempenho das aplicações com outras ferramentas estado da arte (SOTA - state of the art), quantitativamente e qualitativamente. Ferramentas que se baseiam em análise estática ou dinâmica para gerar testes automaticamente, como a Nessie, geralmente escrevem testes muito diferentes dos testes escritos por programadores humanos e de difícil legibilidade. Outro problema é que essas ferramentas não utilizam as APIs de frameworks de teste. LLMs apresentam resultados melhores que essas ferramentas nesses dois quesitos, gerando código que segue os padrões de códigos gerados por humanos e utilizando as APIs de frameworks de teste [12]. Em algumas avaliações as aplicações baseadas em LLMs superaram ferramentas como EvoSuite, AthenaTest, A3Test e Nessie no desempenho medido por

métricas de cobertura, como cobertura de linha e cobertura de branch, em diferentes níveis, a nível de projeto e de método, por exemplo [11], [12]. Porém, em outras avaliações, os LLMs não conseguiram superar as métricas da ferramenta Evosuite [13]. Isso mostra que avaliações envolvendo LLMs é sensível e pode apresentar resultados contrastantes dependendo do cenário.

1) Desafios: A validação da aplicação é uma parte crítica para as pesquisas de aplicação de LLMs em engenharia de software. Se considerarmos que o objetivo subjacente de qualquer pesquisa científica é gerar conhecimento confiável, então, no contexto desta área, o que gera essa confiança é uma validação bem feita. É a partir dela que se encontra o sinal em meio ao ruído. Porém, uma série de desafios precisam ser enfrentados para conseguir uma validação que seja considerada bem feita [17]: I) utilizar LLMs proprietários gera problemas com relação à transparência dos dados de treino, que podem conter dados sensíveis, e com relação à evolução do LLM; II) vazamento de dados de benchmarks [18]; III) dificuldades para reproduzir resultados, devido a variabilidade nas resposta dos LLMs, evolução do modelo ao longo do tempo e falta de rastreamento. Rastrear significa conseguir determinar de onde veio uma resposta, i.e., de qual prompt, fornecido para qual versão de um LLM, com quais configurações de hiperparâmetros.

A solução de desafios dessa natureza ainda não tem uma solução completa e amplamente adotada. Estamos em uma fase de discussão. Contudo, existem recomendações que ajudam a aliviar esses problemas [17].

F. Alvos da Geração de Testes

Os trabalhos coletados e analisados escolhem como unidade de teste métodos de classes. A escolha do método facilita a construção do contexto, por ser uma unidade menor e mais específica. Isso também ajuda a lidar com o tamanho da janela de tokens.

G. Questões Além da Geração

Além da construção dos prompts e do tratamento das respostas dos LLMs existem outros pontos que merecem atenção para o desenvolvimento de aplicações baseadas em LLMs:

- Escolha da linguagem de programação: tipagem estática ou dinâmica, popularidade, existência de outras ferramentas SOTA para geração de testes. A escolha da linguagem de programação afeta consideravelmente o desempenho do ChatGPT em tarefas de geração de código, por exemplo [19];
- Escolha da fonte dos dados para validação: pacotes/projetos de software ou benchmarks;
- Procedimento para identificação, extração e tratamento do método a ser testado e documentação relacionada.

H. Arquitetura das Aplicações

As aplicações de geração de testes baseadas em LLMs precisam extrair a unidade de código de alguma fonte, um projeto de software, por exemplo, inseri-la em um prompt juntamente com seu contexto, validar o teste gerado pelo LLM e, caso necessário, reparar testes defeituosos. Essa divisão em etapas se adequa naturalmente a uma arquitetura de pipeline[8].

I. Outros Tipos de Tarefa

Dentro da área de teste de software, existem outros tipos de tarefa que podem ser fornecidas para LLMs, diferentes da geração de novos testes. Algumas delas são [1]: avaliar a efetividade de um conjunto de testes, prever testes flaky e remoção de testes redundantes.

Uma tarefa parecida com a geração automatizada de testes é o melhoramento de um conjunto de testes. Nesta tarefa, a aplicação recebe como entrada um conjunto de testes e utiliza um LLM para adicionar novos testes a esse conjunto. Essa mudança sutil facilita a integração da aplicação dentro de um fluxo de trabalho existente em uma organização [15].

J. Geração de Testes de Unidade é uma Habilidade Emergente?

Recapitulando rapidamente, no contexto de LLMs, habilidade emergente é um fenômeno pode ser observado em curvas de escalabilidade. Esses fenômenos são caracterizados por uma mudança brusca de desempenho, do aleatório para algo substancialmente melhor que aleatório. A observação desse fenômeno é imprevisível e varia de acordo com a combinação de LLM, tipo de tarefa, métrica de desempenho e cenário de experimentação [10]. A geração de testes de unidade parece ter desempenho melhor que respostas aleatórias. Porém, o presente trabalho desconhece se o fenômeno de emergência, como descrito acima, foi observado para essa tarefa. De qualquer forma, não sabemos como forçar esse tipo de emergência, nem exatamente o que ela significa. Portanto, há muitas incertezas acerca desse conceito para considerá-lo no desenvolvimento de métodos para geração de testes com LLMs.

Uma caracterização extensa de erros do ChatGPT em tarefas de geração de código foi feita em um dos trabalhos coletados [19], evidenciando que LLMs, mesmo com desempenho melhor que aleatório, não necessariamente possuem capacidades extraordinárias de programação. Portanto, enquanto permanecer o desconhecimento sobre o fenômeno de habilidades emergentes, a melhor suposição acerca da capacidade dos LLMs é obtida a partir desse tipo de caracterização sistemática de suas respostas.

IV. Trabalhos Relacionados

Fan et al. [1] fornecem um survey da literatura de aplicação de LLMs em engenharia de software, descrevendo tendências e problemas em aberto da aplicação dos LLMs em diversas atividades de desenvolvimento de software, desde engenharia de requisitos até manutenção e implantação de software.

Hou et al. [2] fornecem uma caracterização sistematizada da literatura de aplicação de LLMs em engenharia de software, cobrindo quais LLMs são utilizados, quais conjuntos de dados são utilizados e como são pré-processados, quais técnicas de otimização de desempenho de LLMs são utilizadas e quais

[8]https://airspeed.ca/when-to-use-the-pipeline-architecture-style/

tarefas de engenharia de software são abordadas. Além disso, também são identificadas e discutidas desafios e oportunidades de pesquisa na área.

V. Discussão

A. Os Modelos

LLMs são um desenvolvimento recente da área de IA. O desconhecimento sobre eles é maior que o que sabemos. Ainda há muito espaço para melhora [10]:

- No desenvolvimento de melhores formas de treinar os LLMs;
- No melhoramento das arquiteturas;
- Na coleta de conjuntos de dados maiores;
- Melhores técnicas para entender e criar prompts;
- Na construção de dados de melhor qualidade;
- No aprofundamento do estudo de fenômenos como grokking e emergência;
- No desenvolvimento de melhores técnicas para construir melhores prompts;
- No desenvolvimento de melhores técnicas que permitam entender melhor o efeito de um tipo de prompt no comportamento dos LLMs;
- No estudo de tarefas que não conseguem apresentar desempenho substancialmente melhor que aleatório em acurácia, como tarefas de planejamento [14].

O maior limitante para o sucesso de aplicações em engenharia de software baseadas em LLMs são os próprios modelos. Avanços nessas linhas de pesquisa certamente impactarão as aplicações dos LLMs em engenharia de software.

B. As Aplicações

O desenvolvimento de aplicações baseadas em LLMs requer a elaboração de métodos que melhor aproveitem o que os modelos têm a oferecer de melhor atualmente, fazendo atualizações à medida que o campo de pesquisa em LLMs avance.

O aumento no desempenho das novas aplicações comparadas ao SOTA anterior foi verificado em alguns casos [11], [12]. Porém, eles não são disruptivos. Mas indicam potencial para continuar explorando a área. É mais provável que o avanço do SOTA ocorra pelas aplicações baseadas em LLMs do que pelas ferramentas anteriores.

Retomando a discussão das diferentes visões acerca de LLMs, levantada na seção II-A, parece que, no momento atual, a área de pesquisa esteja mais alinhada com a visão III, LLMs não possuem nenhum aspecto de inteligência análogo à aspectos humanos, mas possuem capacidades surpreendentes de memorização e recuperação de padrões. Essa é uma visão adequada, considerando o avanço alcançado pelos LLMs em várias tarefas historicamente difíceis para o campo de NLP. Mas, também se alinha bem com as limitações encontradas para outras tarefas de interesse, como geração de código [19], de tal forma que as aplicações devem ser desenvolvidas com certa cautela, não deixando a responsabilidade somente com os LLMs.

C. As Pesquisas

Para garantir que pesquisas desenvolvidas nessa área não se percam no ruído, é fundamental que os desafios elencados sejam discutidos pela comunidade científica [17], buscando uma convergência sobre um conjunto de boas práticas, principalmente no que concerne aos métodos de validação das aplicações.

VI. Conclusão

A aplicação de LLMs em engenharia de software tem o potencial de avançar o SOTA em tarefas como geração automatizada de testes. Por causa disso, um grande volume de pesquisas foram acumuladas nos anos de 2023 e 2024. Este trabalho tem o intuito de trazer um pequeno resumo, para gerar um entendimento em alto nível dessa área de pesquisa nascente. Para isso, discutimos brevemente os principais aspectos que geram empolgação em torno dos LLMs. Em seguida, analisamos os desafios e estratégias desenvolvidas para criar aplicações efetivas de LLMs em tarefas de engenharia de software, focando em apenas uma tarefa, geração automatizada de testes de unidade, o que permitiu uma discussão mais detalhada dos trabalhos coletados. Por fim, listamos alguns trabalhos relacionados e encerramos com uma discussão a respeito do cenário atual e expectativas futuras da pesquisa em LLMs e da pesquisa em aplicações de LLMs em engenharia de software.

References

[1] A. Fan, B. Gokkaya, M. Harman, M. Lyubarskiy, S. Sengupta, S. Yoo, and J. M. Zhang, "Large language models for software engineering: Survey and open problems," 2023.

[2] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang, "Large language models for software engineering: A systematic literature review," 2024.

[3] F. Chollet, "Deep learning with python - second edition," 2021.

[4] S. Bubeck, V. Chandrasekaran, R. Eldan, J. Gehrke, E. Horvitz, E. Kamar, P. Lee, Y. T. Lee, Y. Li, S. Lundberg, H. Nori, H. Palangi, M. T. Ribeiro, and Y. Zhang, "Sparks of artificial general intelligence: Early experiments with gpt-4," 2023.

[5] S. Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, and J. Gao, "Large language models: A survey," 2024.

[6] A. Power, Y. Burda, H. Edwards, I. Babuschkin, and V. Misra, "Grokking: Generalization beyond overfitting on small algorithmic datasets," 2022.

[7] L. Berglund, M. Tong, M. Kaufmann, M. Balesni, A. C. Stickland, T. Korbak, and O. Evans, "The reversal curse: Llms trained on "a is b" fail to learn "b is a"," 2024.

[8] R. Schaeffer, B. Miranda, and S. Koyejo, "Are emergent abilities of large language models a mirage?," 2023.

[9] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," 2020.

[10] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, E. H. Chi, T. Hashimoto, O. Vinyals, P. Liang, J. Dean, and W. Fedus, "Emergent abilities of large language models," 2022.

[11] Y. Chen, Z. Hu, C. Zhi, J. Han, S. Deng, and J. Yin, "Chatunitest: A framework for llm-based test generation," 2024.

[12] M. Schäfer, S. Nadi, A. Eghbali, and F. Tip, "An empirical evaluation of using large language models for automated unit test generation," *IEEE Transactions on Software Engineering*, vol. 50, no. 1, pp. 85–105, 2024.

[13] M. L. Siddiq, J. C. S. Santos, R. H. Tanvir, N. Ulfat, F. A. Rifat, and V. C. Lopes, "Using large language models to generate junit tests: An empirical study," 2024.

[14] K. Valmeekam, S. Sreedharan, M. Marquez, A. Olmo, and S. Kambhampati, "On the planning abilities of large language models (a critical investigation with a proposed benchmark)," 2023.
[15] N. Alshahwan, J. Chheda, A. Finegenova, B. Gokkaya, M. Harman, I. Harper, A. Marginean, S. Sengupta, and E. Wang, "Automated unit test improvement using large language models at meta," 2024.
[16] N. Alshahwan, M. Harman, I. Harper, A. Marginean, S. Sengupta, and E. Wang, "Assured llm-based software engineering," 2024.
[17] J. Sallou, T. Durieux, and A. Panichella, "Breaking the silence: the threats of using llms in software engineering," in *Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*, ICSE-NIER'24, ACM, Apr. 2024.
[18] O. Sainz, J. A. Campos, I. García-Ferrero, J. Etxaniz, O. L. de Lacalle, and E. Agirre, "Nlp evaluation in trouble: On the need to measure llm data contamination for each benchmark," 2023.
[19] Y. Liu, T. Le-Cong, R. Widyasari, C. Tantithamthavorn, L. Li, X.-B. D. Le, and D. Lo, "Refining chatgpt-generated code: Characterizing and mitigating code quality issues," 2023.
[20] K. Valmeekam, M. Marquez, A. Olmo, S. Sreedharan, and S. Kambhampati, "Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change," 2023.
[21] L. Guan, K. Valmeekam, S. Sreedharan, and S. Kambhampati, "Leveraging pre-trained large language models to construct and utilize world models for model-based task planning," 2023.
[22] N. Alzahrani, H. A. Alyahya, Y. Alnumay, S. Alrashed, S. Alsubaie, Y. Almushaykeh, F. Mirza, N. Alotaibi, N. Altwairesh, A. Alowisheq, M. S. Bari, and H. Khan, "When benchmarks are targets: Revealing the sensitivity of large language model leaderboards," 2024.