UNIVERSIDADE FEDERAL DE SERGIPE
DEPARTAMENTO DE COMPUTAÇÃO
TESTE DE SOFTWARE

Atividade 2

CAIO ROSBERG XAVIER LIMA
ISABELA DE GONDRA MENDONCA PEREIRA
LEVY DOS SANTOS SILVA
VIRNA SANTOS OLIVEIRA

SÃO CRISTÓVÃO, AGOSTO DE 2025

SUMÁRIO

1. **Nome do artigo selecionado e seção da qual faz parte**

Seção: Unit test case generation

Artigo: ChatUniTest: a ChatGPT-based Automated Unit Test Generation Tool

Links úteis:

[2305.04764] ChatUniTest: A Framework for LLM-Based Test Generation

2305.04764v1

Repositório da atividade: Teste_de_software_LLM_2025

Vídeo: https://www.youtube.com/watch?v=52iYKloZDjg&authuser=2

2. **Descrição do problema de teste de software abordado**

O artigo "ChatUniTest: a ChatGPT-based Automated Unit Test Generation Tool" trata da dificuldade recorrente enfrentada por desenvolvedores de software na criação de testes unitários, uma etapa essencial para garantir a qualidade e a confiabilidade do código. A elaboração manual desses testes é frequentemente considerada trabalhosa, repetitiva e sujeita a omissões, o que compromete a cobertura e a eficácia da verificação do sistema. O ChatUniTest propõe uma solução automatizada baseada na inteligência artificial do ChatGPT, capaz de compreender o comportamento do código, gerar testes adequados, validá-los e, se necessário, corrigi-los. Essa abordagem visa reduzir o esforço humano, aumentar a produtividade e melhorar a qualidade dos testes gerados, superando limitações de ferramentas tradicionais e de outras soluções baseadas em modelos de linguagem.

3. **Discussão da solução proposta envolvendo algum modelo (LLM) utilizando a arquitetura trasnformer.**

A solução proposta pelo artigo "ChatUniTest: a ChatGPT-based Automated Unit Test Generation Tool" representa um avanço significativo na aplicação de modelos de linguagem de grande escala (LLMs) para o desenvolvimento de software, especialmente no contexto da automação de testes unitários. O problema central abordado é a dificuldade enfrentada por desenvolvedores ao escrever testes que verifiquem se partes específicas do código — chamadas de unidades — funcionam corretamente. Essa tarefa, embora essencial para garantir a qualidade do

software, é frequentemente negligenciada por ser repetitiva, demorada e suscetível a erros humanos.

Nesse cenário, o ChatUniTest propõe uma abordagem inovadora ao empregar um modelo de linguagem baseado na arquitetura Transformer, a mesma utilizada por sistemas como o ChatGPT. Essa arquitetura é composta por mecanismos de atenção que permitem ao modelo analisar grandes volumes de texto (ou código) e compreender relações complexas entre seus elementos. Diferentemente de abordagens tradicionais, que se baseiam em regras fixas ou geração aleatória de entradas, os Transformers conseguem interpretar o significado do código, identificar suas intenções e gerar testes coerentes e contextualizados.

A ferramenta funciona em três etapas principais: geração, validação e reparo. Na fase de geração, o modelo analisa o método a ser testado e suas dependências, e então produz um teste unitário que simula diferentes cenários de uso. Em seguida, esse teste é validado automaticamente para verificar se compila e executa corretamente. Caso haja falhas, entra em ação a etapa de reparo, que pode ser feita por regras automatizadas ou por uma nova interação com o modelo, que propõe correções com base no erro identificado.

O uso de um LLM com arquitetura Transformer confere à ferramenta uma capacidade sem precedentes de adaptação e generalização. O modelo não apenas entende a estrutura do código, mas também é capaz de inferir comportamentos esperados, sugerir asserções relevantes e até mesmo empregar bibliotecas de teste específicas, como JUnit ou PyTest, conforme o contexto. Isso resulta em testes mais precisos, legíveis e úteis para os desenvolvedores.

Em suma, o ChatUniTest demonstra como a inteligência artificial, por meio de modelos Transformer, pode transformar práticas tradicionais de engenharia de software, automatizando tarefas complexas com qualidade e eficiência. Essa abordagem não apenas reduz o esforço manual, mas também contribui para a construção de sistemas mais robustos e confiáveis.

4. **Indicar o nome e versão do modelo, URL no qual está disponível ou pode ser utilizado, tipo de arquitetura (encoder-only, decoder-only, encoder-decoder).**

Nome do modelo: ChatGPT

Versão: Não especificada explicitamente, mas provavelmente GPT-3 ou GPT-4 (dado o período de publicação).

URL de acesso: https://chat.openai.com/ ou via API em https://platform.openai.com/

Tipo de arquitetura: decoder-only

Essa escolha arquitetural é justificada pela natureza da tarefa de geração de testes unitários, que exige a produção de texto coerente e contextualizado a partir de prompts fornecidos. Diferentemente das arquiteturas encoder-only, voltadas para tarefas de classificação ou compreensão, e das encoder-decoder, mais comuns em tradução automática, o modelo decoder-only é projetado para gerar sequências de texto de forma autônoma, mantendo a fluidez e a lógica da saída com base no histórico de entrada. Essa característica torna o GPT particularmente eficaz para aplicações que envolvem geração de código, como é o caso do ChatUniTest, onde o modelo interpreta o comportamento de funções e métodos e produz testes unitários adequados, simulando cenários de uso e validando a funcionalidade do software.

5. **Indicar o nome, versão e URL da dataset e ou código fonte utilizado no artigo para o cenário de teste de software.**

Nome da dataset principal de métodos: Methods2test (sem versão específica indicada)

Código fonte de projetos de teste: Derivados de projetos open-source no GitHub (Commons CLI, Csv, Gson, JFreeChart)

https://github.com/apache/commons-cli

https://github.com/apache/commons-csv

https://github.com/google/gson

https://github.com/jfree/jfreechart

6. **Descrever como a solução foi avaliada.**

A avaliação da solução ChatUniTest foi conduzida em duas dimensões complementares: **eficácia técnica** e **utilidade prática**.

Avaliação de eficácia (técnica)

O objetivo principal dessa etapa foi mensurar a capacidade do ChatUniTest de gerar testes unitários de qualidade. Para isso, ele foi comparado com duas ferramentas representativas no contexto de geração automática de testes:

- EvoSuite, que adota uma abordagem tradicional baseada em análise de programas;
- TestSpark, que segue uma abordagem moderna utilizando modelos de linguagem de grande porte (LLMs).

O experimento foi realizado em quatro projetos Java com características distintas:

1. Commons-CLI e Commons-Csv – amplamente utilizados em pesquisas relacionadas e, portanto, provavelmente incluídos nos dados de treinamento do GPT-3.5-turbo, modelo base do ChatUniTest;
2. Ecommerce-microservice e Binance-connector – projetos populares, com mais de 100 estrelas no GitHub, criados após o treinamento do modelo, portanto não conhecidos pelo ChatUniTest, o que possibilitou testar sua capacidade de generalização.

Para tornar a avaliação estatisticamente confiável, foi aplicada amostragem aleatória com 95% de confiança e 5% de margem de erro, resultando na seleção de 264 métodos focais, de um total de 835 métodos presentes nos quatro projetos. Para cada método, a ferramenta tentou gerar um teste automaticamente, considerando uma rodada de geração inicial seguida de cinco tentativas de reparo caso houvesse falha.

A métrica central de análise foi a cobertura de linhas de código (line coverage), que indica o percentual de linhas executadas pelos testes gerados. Essa métrica é amplamente aceita em engenharia de software como indicador da eficácia dos testes unitários.

Os resultados mostraram que o ChatUniTest atingiu uma cobertura média de 59,6%, superando as demais ferramentas. Essa avaliação também evidenciou limitações nas ferramentas concorrentes:

- O TestSpark não conseguiu gerar testes para o projeto Commons-Csv, pois o prompt construído ultrapassou o limite de tokens do modelo.
- O EvoSuite apresentou falhas no Ecommerce-microservice, devido a incompatibilidade entre a versão do JDK e o projeto.

Portanto, a avaliação de eficácia demonstrou que o ChatUniTest é robusto e eficaz, mesmo quando aplicado a projetos novos e fora de seu conjunto de treinamento.

Avaliação de utilidade (prática)

A segunda etapa avaliou a percepção dos usuários sobre o valor prático da ferramenta. Foram enviados questionários para 19 pessoas que demonstraram interesse pelo ChatUniTest (por meio de estrelas, forks ou issues no GitHub e contatos por e-mail), recebendo 9 respostas.

O público foi composto por:

- 5 estudantes, com experiência inicial em testes de software;
- 4 desenvolvedores sêniores, com domínio da área e experiência prévia no uso de LLMs para programação.

Os resultados apontaram que:

- 89% dos participantes já utilizavam o ChatUniTest para auxiliar na escrita de testes unitários;
- 33% consideraram o uso da ferramenta altamente benéfico, sobretudo para desenvolvedores iniciantes, que tendem a ter dificuldade em criar casos de teste eficazes;
- Alguns respondentes já contribuíam para evolução do ChatUniTest, como integrar novos LLMs, adicionar suporte a múltiplos idiomas e incorporar a ferramenta em seus fluxos de desenvolvimento.

Além disso, sugestões relevantes foram coletadas, como:

- Incluir testes de desempenho além dos testes unitários;
- Adicionar suporte a frameworks internos e outras bibliotecas de teste;
- Validar a confiabilidade do código gerado em ambientes de produção reais.

7. Apresentar um exemplo de avaliação da solução.

Um exemplo concreto da avaliação do ChatUniTest ocorreu no projeto Commons-Csv, um dos quatro projetos Java utilizados no estudo. Nesse caso:

- O ChatUniTest conseguiu gerar testes unitários com alta cobertura de linhas, demonstrando sua eficácia mesmo em um projeto amplamente utilizado e complexo;
- O TestSpark falhou em gerar testes devido ao tamanho excessivo do prompt, que ultrapassou o limite de tokens do modelo;
- O EvoSuite conseguiu gerar testes, mas não superou a cobertura de linha do ChatUniTest.

8. Discutir o potencial de uso da solução no seu dia a dia de programação

A solução apresentada no artigo ChatUniTest: A ChatGPT-based Automated Unit Test Generation Tool possui um grande potencial para ser utilizada no dia a dia do desenvolvimento de software, especialmente em situações em que a elaboração de testes unitários é uma tarefa frequente, porém geralmente ignorada devido à sua natureza repetitiva e ao elevado tempo que demanda. No contexto prático, a implementação de uma ferramenta para automatizar a criação de testes pode resultar em uma mudança considerável na eficiência das equipes e na qualidade do código desenvolvido.

O ChatUniTest utiliza modelos de linguagem baseados na arquitetura Transformer decoder-only (como o ChatGPT) para entender trechos de código-fonte, reconhecer dependências e comportamentos esperados. Com essas informações,

ele é capaz de criar casos de teste unitário relevantes e bem organizados. Essa habilidade possibilita uma significativa economia de tempo no processo de desenvolvimento, principalmente nas etapas de manutenção ou ampliação de sistemas legados, que frequentemente não possuem uma cobertura de testes adequada.

Sob a perspectiva educacional, a ferramenta é útil para programadores iniciantes ou em formação, pois oferece exemplos de testes bem elaborados e alinhados às melhores práticas de engenharia de software. Dessa forma, desempenha um papel como ferramenta de apoio no processo de aprendizagem e na difusão de métodos de verificação e validação automatizados.

Ademais, ele pode ser incorporado ao pipeline de desenvolvimento em ambientes profissionais que empregam integração contínua (CI) e entrega contínua (CD), auxiliando na automação da validação de novas funcionalidades. Por exemplo, a criação de testes em tempo real durante as revisões de código pode agilizar o processo de desenvolvimento e diminuir a ocorrência de regressões.

Em resumo, o ChatUniTest se mostra uma ferramenta promissora para agilizar, aumentar a confiabilidade e a acessibilidade do processo de desenvolvimento, diminuindo o trabalho manual necessário para criar testes e proporcionando maior cobertura e robustez aos sistemas de software.

9. Listar possíveis limitações da solução

Embora o ChatUniTest tenha feito contribuições significativas e obtido resultados positivos na avaliação experimental apresentada no artigo, ele possui algumas limitações que precisam ser levadas em conta tanto em sua implementação quanto em seu desenvolvimento futuro.

Em primeiro lugar, é importante ressaltar a limitação de tokens que é característica dos modelos de linguagem empregados. Em projetos que utilizam métodos extensos ou com várias dependências, o prompt criado para a geração do teste pode exceder o limite de tokens que o modelo consegue suportar, impossibilitando a resposta. Esse ponto foi demonstrado nos experimentos do artigo, nos quais o modelo TestSpark, fundamentado em LLMs, não teve sucesso ao lidar

com o projeto Commons-Csv por essa razão. Isso indica que o ChatUniTest enfrenta o mesmo desafio, devido à semelhança na arquitetura.

A dependência da clareza e qualidade do código-fonte analisado representa outra limitação significativa. Quando o código possui nomes pouco descritivos, baixa coesão ou falta de documentação mínima, a habilidade do modelo de fazer inferências pode ser afetada, o que compromete a relevância e a precisão dos testes produzidos. Isso é especialmente importante em sistemas legados ou desenvolvidos com pouca padronização.

Além disso, nota-se que a ferramenta se concentra apenas na criação de testes unitários, não abordando outros níveis de teste, como testes de integração, sistema, aceitação ou desempenho. Esse recorte funcional pode ser restritivo em projetos que exigem uma estratégia de testes mais completa.

É importante ressaltar que, apesar de o modelo ser capaz de propor testes com base na análise sintática e semântica do código, ele ainda não possui validação semântica mais aprofundada. Isso significa que é necessário uma revisão humana para assegurar que os testes criados atendam aos requisitos de negócio e não gerem falsas garantias de qualidade.

Por último, é importante destacar que a ferramenta pode oferecer suporte limitado para algumas linguagens ou bibliotecas específicas, sendo mais eficiente em ecossistemas que possuem ampla documentação, como Java e Python. Projetos criados em linguagens menos populares ou que utilizam frameworks internos personalizados podem não aproveitar completamente os recursos da ferramenta.

10. Apresentar uma sugestão sua de como a solução poderia ser melhorada

Um dos pontos de melhoria da solução está relacionado à limitação de tokens, um problema comum em modelos de linguagem com essa arquitetura. A dificuldade em processar entradas grandes (prompts) acaba impedindo a análise de códigos mais extensos e complexos, por exemplo.

Uma sugestão de melhoria seria criar regras para lidar com esse tipo de situação, evitando a perda de informações. A ideia é que a ferramenta consiga

identificar quando a quantidade de tokens estiver se aproximando do limite e, nesse momento, divida o prompt em partes menores. Um limite fixo é definido para controlar quantos tokens o prompt pode consumir dentro do limite suportado, assim, seria possível garantir que o conteúdo total fique dentro do limite permitido e que o modelo consiga analisar tudo e gerar respostas reais e mais completas para os problemas apresentados, sem eliminar automaticamente os dados menos relevantes no prompt do usuário, para que seja possível está dentro de limite do token e não gerar testes truncados e com erros sintáticos

11. **Identificar um problema registrado no stackoverflow que poderia utilizar a solução descrita no artigo por você selecionado.**

Nome: Como escrever testes para código legado e depois refatorá-lo?

A justificativa da escolha do problema é motivada devido aos recursos oferecidos pelo framework satisfazer a necessidade real do autor. Os testes unitários são uma prática crítica no desenvolvimento de software, garantindo a qualidade das aplicações, o autor do problema informa a dificuldade em criar os testes "Mas sempre que tento escrever testes unitários para código existente, fico travado. Simplesmente não sei como testar meu código antigo. Não consigo escrever nenhum testezinho", relata. A ferramenta automatizada o ajudaria na geração de testes unitários, auxiliando nesse campo em que precisa de ajuda, já que escrever e manter manualmente testes unitários de alta qualidade pode ser uma tarefa desafiadora.

URL: https://stackoverflow.com/questions/31276462/how-to-write-tests-for-legacy-code-and-then-refactor-it

Referências

XIE, Zhuokui; CHEN, Yinghao; ZHI, Chen; DENG, Shuiguang; YIN, Jianwei. ChatUniTest: a ChatGPT-based automated unit test generation tool. Zhejiang University, Hangzhou, China, 2023. Disponível em: https://github.com/ZJU-ACES-ISE/ChatUniTest. Acesso em: 31 jul. 2025.

Neste trabalho, utilizou-se o Microsoft Copilot, baseado no GPT-4 Turbo da OpenAI, e a ferramenta ChatPDF (versão 1.9.0), para auxiliar na análise de documentos e execução das tarefas com mais agilidade e precisão.