Geração de Código Java a partir de Requisitos Funcionais Utilizando *Large Language Models* (LLMs)

Seu Nome / Nome do Orientador
Instituição
Edital PIBIC/CNPq

4 de junho de 2025

Resumo

Este projeto de iniciação científica investiga a viabilidade e eficácia do uso de *Large Language Models* (LLMs) na geração automática de código Java funcional a partir de requisitos funcionais de sistemas. O objetivo principal é avaliar a qualidade, correção e conformidade do código gerado, visando otimizar a transição de requisitos para implementação. Serão conduzidos experimentos com diferentes LLMs, aplicando métricas de avaliação e um mecanismo de validação baseado em testes unitários. A pesquisa busca contribuir para a redução de erros, o aumento da produtividade do desenvolvedor e a aceleração do ciclo de desenvolvimento de software. Os resultados esperados incluem uma análise comparativa das LLMs e a documentação das descobertas.

1 Introdução

O desenvolvimento de software moderno enfrenta desafios crescentes em relação à eficiência, agilidade e qualidade na fase de implementação. A transição de requisitos funcionais para código executável, especialmente em linguagens como Java, é um processo intensivo em mão de obra, propenso a erros e que demanda considerável tempo e expertise em programação (Li et al., 2023). Tradicionalmente, desenvolvedores interpretam documentos de requisitos, traduzem-nos em lógica de programação e, então, escrevem o código, um fluxo que pode introduzir inconsistências e atrasos (Chen et al., 2021).

Com o avanço das *Large Language Models (LLMs)*, como GPT-4, Llama 3 e Gemini, surgem novas oportunidades para automatizar e otimizar partes do ciclo de vida do desenvolvimento de software (Salloum et al., 2023). Esses modelos, treinados em vastos corpora de texto e código, demonstram notável capacidade de compreender linguagem natural e gerar código em diversas linguagens de programação (Zou et al., 2024). No entanto, a aplicação dessas LLMs na geração automática de código Java a partir de requisitos funcionais detalhados de sistemas ainda é um campo em evolução, com lacunas a serem exploradas em termos de precisão, robustez e adaptabilidade a diferentes domínios de software (Sobania et al., 2023). O problema central que este projeto busca abordar é a complexidade e o custo associados à tradução manual de requisitos funcionais em código Java de alta qualidade e livre de erros, que podem ser mitigados pela automação inteligente.

—

2 Motivação

Este projeto tem sua motivação em diversas frentes. A seguir, são expostas quatro delas:

1. **Aumento da Produtividade do Desenvolvedor:** A automação da geração de código a partir de requisitos funcionais pode liberar os desenvolvedores de tarefas repetitivas e de baixo nível, permitindo que se concentrem em aspectos mais complexos do projeto, como arquitetura, design de sistemas e otimização de desempenho. Isso pode levar a um aumento significativo na produtividade e na satisfação no trabalho.

2. **Redução de Erros e Inconsistências:** A interpretação humana de requisitos pode levar a ambiguidades e, consequentemente, a erros de codificação. Um sistema baseado em LLMs, devidamente treinado e validado, pode gerar código mais consistente e aderente aos requisitos originais, reduzindo a incidência de *bugs* e o tempo gasto em depuração.

3. **Aceleração do Ciclo de Desenvolvimento:** A capacidade de gerar rapidamente protótipos ou partes do código a partir de requisitos funcionais pode acelerar drasticamente o ciclo de desenvolvimento, permitindo entregas mais rápidas e *feedbacks* mais ágeis, um benefício crucial em metodologias ágeis de desenvolvimento.

4. **Demanda Crescente por Soluções Inovadoras:** A indústria de software busca constantemente inovações que otimizem seus processos. A aplicação de LLMs na geração de código representa uma fronteira de pesquisa promissora com alto potencial de impacto no mercado.

Uma contribuição mais geral para esta proposta (principalmente considerando que se trata de uma iniciação científica) consiste em contribuir para o avanço do conhecimento nas áreas de Inteligência Artificial e Engenharia de Software, explorando os limites das LLMs e suas aplicações práticas no desenvolvimento de sistemas complexos. Na verdade, este é o objetivo do projeto de pesquisa no qual esta proposta está inserida. Por óbvio, não pretende-se chegar nisso estritamente com a IC. —

3 Objetivo

O Objetivo Geral consiste em investigar a viabilidade e a eficácia do uso de *Large Language Models* (LLMs) na geração automática de código Java funcional a partir de requisitos funcionais de sistemas, avaliando a qualidade, correção e conformidade do código gerado com os requisitos especificados.

Os Objetivos Específicos são:

- Definir um Conjunto de Requisitos Funcionais de Teste: Criar ou selecionar um conjunto de requisitos funcionais bem definidos para um sistema em Java, de complexidade média, que sirva como base para a experimentação.

- Experimentar com Diferentes LLMs: Realizar experimentos com pelo menos duas LLMs de ponta (ex: GPT-4, Llama 3, Gemini) para gerar código Java a partir dos requisitos funcionais definidos.

- Desenvolver Métricas de Avaliação: Propor e aplicar métricas para avaliar a qualidade do código gerado, incluindo correção sintática e semântica, aderência aos requisitos funcionais, legibilidade e manutenibilidade.

- Implementar um Mecanismo de Validação: Desenvolver um mecanismo para validar o código Java gerado, que pode incluir a execução de testes unitários ou de integração baseados nos requisitos.
- Analisar os Resultados: Comparar a performance das diferentes LLMs na geração de código, analisando os pontos fortes e fracos de cada uma e identificando padrões de sucesso e falha.

—

4 Metodologia Detalhada

Este projeto seguirá uma abordagem experimental e comparativa, utilizando um *pipeline* estruturado para investigar a capacidade das LLMs na geração de código Java. Ressalta-se que,por questão de limitação, as referências a ferramentas tecnológicas foram omitidas, mas o acesso a elas é aberto e livre.

A metodologia encontra-se dividida em três grupos: atividades preparatórias, atividades do projeto e atividades inerentes ao edital. O primeiro grupo – Preparação – são as atividades de estudo teórico e de tecnologias empregadas no Projeto de Pesquisa, pré-requisitos para o desenvolvimento do trabalho em si.

As atividades de **preparação** devem ser efetivadas em 3 meses, são elas:

1. Revisão Bibliográfica: Análise de metodologias, resultados, desafios e oportunidades apresentadas na literatura. Síntese dos achados para embasar as escolhas metodológicas do projeto. De antemão, destacam-se os trabalhos já citados aqui.
2. Definição de Requisitos Funcionais: Requisitos para um sistema Java de complexidade média, contendo diferentes tipos de funcionalidade (CRUD, validação de dados, lógica de negócio simples, interação com estruturas de dados básicas). Os requisitos serão especificados em linguagem natural.

As atividades específicas do Projeto devem durar de 8 a 10 meses. São elas:

1. Experimentação e Geração de Código (Meses 4-6)
   (a) Configuração do Ambiente de Experimentação:
      - Acesso a LLMs: Utilização de APIs de LLMs (ex: OpenAI GPT-4, Google Gemini Pro) ou modelos de código aberto de alta performance (ex: Llama 3) hospedados em plataformas de cloud.
      - Framework de Desenvolvimento: Ambiente de desenvolvimento Java (IntelliJ IDEA ou Eclipse) com Maven/Gradle para gerenciamento de dependências.
      - Controle de Versão: Git e GitHub/GitLab para controle de versão do código gerado e dos scripts de experimentação.

   (b) Geração de Código Iterativa:
      - Para cada requisito funcional, as LLMs serão consultadas para gerar o código Java correspondente.

- Prompt Engineering: Será desenvolvida uma estratégia de *prompt engineering* para otimizar as respostas das LLMs. Isso inclui a elaboração de prompts claros, concisos e que orientem a LLM na geração de código Java adequado (ex: especificação da classe, métodos esperados, estruturas de dados).
- Variações de Prompts: Serão testadas diferentes formulações de prompts (e.g., com exemplos de código, com instruções de *design patterns* simples) para observar seu impacto na qualidade do código gerado.
- Registro Detalhado: Todos os prompts utilizados e os códigos gerados serão registrados para análise posterior.

2. Avaliação e Validação (Meses 6-8)

   (a) Avaliação do Código Gerado:
   - Correção Sintática: Verificação da compilação do código Java gerado.
   - Correção Semântica e Aderência Funcional:
     – Testes Unitários: Para cada funcionalidade esperada, serão escritos testes unitários (JUnit) com base nos requisitos originais. O código gerado pela LLM será executado contra esses testes.
     – Inspeção Manual: Desenvolvedores experientes farão uma inspeção manual do código para verificar a lógica, a clareza, a aderência a boas práticas de programação e a cobertura dos requisitos.
   - Métricas de Qualidade de Código:
     – Linhas de Código (LOC): Medida da extensão do código.
     – Complexidade Ciclomática: Medida da complexidade da lógica.
     – Legibilidade: Avaliada por meio de escala Likert por um grupo de avaliadores humanos.

   (b) Comparação entre LLMs:
   - Os resultados de cada LLM serão comparados usando as métricas definidas.
   - Análise estatística simples para identificar diferenças significativas na performance.

   (c) Identificação de Padrões e Limitações:
   - Análise dos tipos de requisitos que as LLMs geram código com sucesso e aqueles que apresentam dificuldades.
   - Identificação de vieses, erros comuns ou limitações inerentes à geração de código pelas LLMs.

3. Análise de Dados e Documentação (Meses 10-12)

   (a) Análise Abrangente dos Resultados:
   - Síntese dos dados coletados nas fases de experimentação e avaliação.
   - Interpretação dos resultados em relação aos objetivos do projeto.

O terceiro grupo de tarefas -– Atividades inerentes ao Edital -– consiste nas atividades demandadas no referido edital. São elas: Relatório Parcial e Relatório Final. Ambas são apontadas no cronograma. A participação do Congresso de Iniciação Científica, embora demandada, não consta na metodologia, mas consta no cronograma. —

5 Cronograma

O cronograma de atividades é apresentado na Tabela 1, sendo que a numeração das atividades está separada em **Preparação**, **Projeto** e **Edital**, conforme consta na Metodologia.

Tabela 1: Cronograma do Projeto de Iniciação Científica

Atividades		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Atividades		1	2	3	4	5	6	7	8	9	10	11	12
Preparação	1	X	X										
	2		X	X									
Projeto	1a			X	X								
	1b				X	X							
	1c					X	X						
	2a						X	X	X				
	2b							X	X				
	2c								X	X	X		
	3a									X	X	X	
Edital	1						X						
	2											X	X

—

Referências

Mark Chen, H. Tworek, Heewoo Jun, Quoc Long, Shixiang Zhou, Annie Kong, Andrew Li, Sam Zimmerman, Shaoteng Wen, Yongschun Lu, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Shangqing Li, Ruichao Huang, Ting Liu, Xinhang Yan, Xin Xia, Tian Zhang, and Xiangjun Wu. Code generation from natural language with large language models: A survey. *arXiv preprint arXiv:2307.04278*, 2023.

Said Salloum, S. Al-Emadi, and S. Al-Maadeed. Large language models for code generation: A systematic review. *Journal of Systems and Software*, 203:111756, 2023.

D. Sobania, M. Briesch, B. Hanna, and S. Fritsch. An empirical study on the effectiveness of large language models for code generation. In *Proceedings of the 17th International Conference on Software Engineering and Applications (ICSEA)*, 2023.

Xing Zou, Qingsheng Wang, Lincheng Liu, Cong Xu, Xiang Zhao, and Gang Chen. A comprehensive survey on large language models for software engineering. *ACM Computing Surveys (CSUR)*, 56(2):1–38, 2024.