Análise de Qualidade em Aplicações Paralelas: Avaliando o Uso de IA Generativa em Pipelines de Desenvolvimento

Sara Barbosa[1], **Erivan Júnior**[1], **David Cunha**[1]

[1]CESAR School
Recife – PE – Brasil

svob@cesar.school, easj@cesar.school, dcpc@cesar.school

Abstract. *This paper investigates the use of Generative Artificial Intelligence (LLMs) to overcome the limitations of traditional static analysis tools (SAST) in ensuring the quality of parallel applications. Because conventional analyzers lack execution context interpretation, they struggle to detect complex concurrency anomalies, such as race conditions and deadlocks. To validate the generative AI approach in this scenario, a target application called Parallel System was developed, a controlled Java/Spring Boot environment that simulates workloads and I/O bottlenecks. The research integrates generative models into CI/CD pipelines to evaluate their accuracy, computational cost, and explanatory capabilities against conventional approaches, aiming to provide practical guidelines for the continuous analysis of high-performance software.*
Keywords *Generative Artificial Intelligence; Parallel Applications; Software Quality; Code Analysis; CI/CD.*

Resumo. *Este artigo investiga o uso de Inteligência Artificial Generativa (LLMs) para superar as limitações das ferramentas tradicionais de análise estática (SAST) na garantia da qualidade de aplicações paralelas. Como os analisadores convencionais falham em interpretar o contexto de execução, eles apresentam dificuldades para detectar anomalias complexas de concorrência, como condições de corrida e impasses (deadlocks). Para validar a aplicação de IA Generativa nesse cenário, desenvolveu-se uma aplicação-alvo denominada Parallel System, um ambiente controlado em Java/Spring Boot que simula cargas e gargalos de I/O. A pesquisa integra modelos generativos em pipelines CI/CD para comparar sua acurácia, custo computacional e capacidade explicativa frente às abordagens convencionais, visando fornecer diretrizes práticas para a análise contínua de software de alto desempenho.*
Palavras-Chave *Inteligência Artificial Generativa; Aplicações Paralelas; Qualidade de Software; Análise de Código; CI/CD.*

1. Introdução

A garantia da qualidade de *software* ágil exige análises automatizadas e integradas aos *pipelines* de desenvolvimento [Semerano e de Oliveira 2024]. Ferramentas de análise estática (SAST), embora úteis para extrair métricas e avaliar a qualidade fundamental do produto [ISO/IEC 2011], falham na interpretação contextual. Elas frequentemente geram falsos positivos diante da complexidade de aplicações paralelas, onde condições de corrida e impasses (*deadlocks*) dependem do contexto de execução [Marcelino 2025]. Para suprir essa lacuna, este trabalho propõe o uso de Inteligência Artificial Generativa

(LLMs) na análise contínua de *software* paralelo, investigando se os modelos generativos superam os analisadores convencionais ao compreender semanticamente o código e diagnosticar desafios de sincronização de forma explicável em linguagem natural.

2. Fundamentação e Contexto

A avaliação da qualidade tem se baseado em ferramentas de Análise Estática (SAST) que inspecionam o código sem executá-lo. Embora o controle rigoroso seja essencial para minimizar defeitos e elevar a maturidade profissional [Pressman e Maxim 2021], essas abordagens sofrem com alta incidência de falsos positivos e dificuldade crônica em detectar vulnerabilidades contextuais [Marcelino 2025]. Esse cenário é crítico na computação concorrente, que introduz *bugs* complexos que impactam diretamente os pilares de Confiabilidade e Eficiência de Desempenho definidos pela norma ISO/IEC 25010, como contenção de recursos (*locks*) e condições de corrida [Hennessy e Patterson 2002]. Como o SAST falha em abstrair o comportamento dinâmico de múltiplas *threads*, a depuração paralela exige um nível de compreensão semântica que ferramentas puramente quantitativas não possuem.

É nesse gargalo que a IA Generativa atua. O mercado caminha para a "Engenharia de IA Abrangente", onde os LLMs conectam todo o ciclo de vida do desenvolvimento [PRILL 2026]. Capaz de processar históricos complexos e atuar como agente ativo na garantia da qualidade em equipes ágeis [Semerano e de Oliveira 2024], a IA Generativa integrada a *pipelines* CI/CD transforma a mera contagem de métricas em abstrações detalhadas sobre problemas finos de concorrência.

3. Metodologia

A pesquisa é conduzida de forma experimental, estruturada a partir da coleta e análise de dados provenientes de repositórios reais de *software*. Inicialmente, realiza-se a mineração desses repositórios para a construção de um *dataset* consolidado que relaciona o histórico evolutivo (*commits*, *pull requests* e *issues*) com métricas de qualidade extraídas por meio de ferramentas tradicionais (SAST). Esses indicadores incluem complexidade ciclomática, cobertura de testes, vulnerabilidades e *code smells*.

Em paralelo, para superar a análise puramente quantitativa, Modelos de Linguagem de Grande Escala (LLMs) especializados em código, especificamente GPT-4o, CodeLlama e StarCoder, são aplicados sobre o mesmo conjunto de dados. A abordagem metodológica utiliza técnicas avançadas de engenharia de *prompts* combinadas com *fine-tuning* para otimizar a análise preditiva e a detecção de anomalias. A proposta é que esses modelos identifiquem os problemas de paralelismo, interpretem os padrões do código e gerem relatórios de qualidade e explicações compreensíveis em linguagem natural.

A avaliação empírica consolida uma comparação direta entre as ferramentas estáticas tradicionais e a abordagem baseada em IA Generativa. Para garantir o rigor científico, os critérios de validação combinam aspectos qualitativos, como a interpretabilidade das explicações geradas, com métricas quantitativas robustas, incluindo *Precision* e *Recall* na detecção de falhas de concorrência, tempo de resposta e custo computacional (consumo de *tokens*).

Todo esse fluxo de validação ocorre de forma contínua no contexto de *pipelines* CI/CD. A cada alteração no código, métricas tradicionais são coletadas e enviadas aos modelos generativos juntamente com o histórico de *commits*. A IA atua transformando esses dados técnicos isolados em relatórios explicativos, facilitando a identificação de padrões de degradação e a tomada de decisão pelos desenvolvedores.

4. Ambiente Experimental

Para avaliar a efetividade da nossa ferramenta baseada em IA Generativa, é fundamental utilizar um ambiente controlado que permita a validação empírica contra abordagens convencionais, à semelhança do uso de *benchmarks* estruturados na avaliação de ferramentas de Análise Estática de Código (SAST) [Marcelino 2025]. Para este fim, utilizamos como aplicação-alvo um sistema denominado *Parallel System*. Desenvolvido em Java com o *framework* Spring Boot, o sistema expõe uma API REST cujo *endpoint* principal (`GET /benchmark/run`) aciona a execução de um conjunto de tarefas paralelas. Esse *endpoint* permite a configuração dinâmica da carga, variando a quantidade de tarefas (`items`) e o tamanho do *pool* de *threads* (`threads`). O fluxo arquitetural segue o padrão clássico, onde o *Controller* captura os parâmetros e delega o processamento para a camada de *Service*.

A lógica de paralelismo do *testbed* baseia-se na instanciação de um *ExecutorService* de tamanho fixo. As tarefas são submetidas assincronamente utilizando `CompletableFuture.runAsync()` e sincronizadas por meio de uma barreira com `CompletableFuture.allOf(...).join()`, que bloqueia a *thread* principal até a conclusão do lote. Para tornar o *benchmark* representativo, a simulação introduz latências aleatórias de 5 a 30 ms, emulando com precisão operações *I/O-bound* e gargalos reais de serviços externos. O tempo de processamento é medido pela biblioteca *Micrometer*, que captura a métrica `parallel.processing.duration`, englobando desde o enfileiramento até a sincronização final. Com esses dados, a IA avalia o desempenho para detectar a desaceleração paralela (*parallel slowdown*), condição onde o excesso de *threads* aumenta o tempo de resposta devido à pesada sobrecarga de contexto, limitando a escalabilidade prevista pela Lei de Amdahl [Hennessy e Patterson 2002].

Nesta pesquisa, o *Parallel System* atua como o laboratório de provas prático para os modelos de linguagem (LLMs). O objetivo é extrair e avaliar métricas fundamentais de qualidade de *software* [Klauck 2023], verificando se a IA consegue detectar de forma autônoma vulnerabilidades e limitações de concorrência presentes no código-fonte, superando a dificuldade das ferramentas estáticas tradicionais na interpretação de contexto [Marcelino 2025]. Entre os desafios intencionais do sistema que a IA deve identificar, destacam-se o risco de *overhead* por troca de contexto, caso o número de *threads* configurado exceda os núcleos disponíveis na máquina, e o possível esgotamento de tempo no bloqueio do `awaitTermination(5, TimeUnit.SECONDS)`, o que poderia mascarar a conclusão de tarefas sob cargas elevadas.

5. Resultados Esperados e Considerações Finais

Os resultados fornecem evidências iniciais de que a utilização de Inteligência Artificial Generativa pode complementar as ferramentas tradicionais de análise estática em pipelines de CI/CD. Avaliando o mesmo repositório, a ferramenta SAST (SonarQube)

identificou 167 *issues*, enquanto a abordagem baseada em IA reportou 4 *issues*. Embora a ferramenta tradicional tenha apresentado maior cobertura de detecção, a análise baseada em IA demonstrou maior capacidade de priorização dos resultados, alcançando uma densidade de sinal de aproximadamente 50%, contra 30% da abordagem convencional.

Diferenças também foram observadas nos indicadores de qualidade. Na dimensão Confiabilidade, a ferramenta SAST registrou 8 *issues* e atribuiu grau **E** ao sistema, enquanto a abordagem baseada em IA identificou 1 *bug* relevante e classificou o código com grau **C**. Em Manutenibilidade, a SAST reportou 156 *issues*, contra 2 *code smells* identificados pela IA, resultando em avaliação mais objetiva e interpretável.

A IA generativa agrega capacidade de contextualização à análise automatizada de software, reduzindo o esforço de triagem e fornecendo diagnósticos mais compreensíveis. Em aplicações paralelas, nas quais problemas de concorrência e desempenho dependem da compreensão do contexto de execução, essa característica representa uma vantagem em relação às abordagens SAST. Ao combinar a ampla cobertura dos analisadores estáticos com a capacidade de contextualização dos modelos baseados em IA, esta pesquisa contribui para mitigar o débito técnico acidental causado pela complexidade do paralelismo e o *bit rot* (degradação do código ao longo do tempo) citeradigan2026divida. Esse alinhamento aumenta a maturidade das equipes ágeis e favorece o desenvolvimento de sistemas paralelos mais seguros, previsíveis e eficientes.

Como continuidade da pesquisa, pretende-se ampliar a validação experimental no ambiente *Parallel System*, incorporando métricas formais de precisão, *recall* e custo computacional, de modo a viabilizar uma avaliação mais rigorosa da efetividade da IA generativa.

Referências

Hennessy, J. L. e Patterson, D. A. (2002). *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 3 edition.

ISO/IEC (2011). Iso/iec 25010:2011 - systems and software engineering — systems and software quality requirements and evaluation (square) — system and software quality models.

Klauck, J. (2023). 11 indicadores de qualidade de software e sua importância na área de qa.

Marcelino, L. P. (2025). Análise estática de código com sonarqube: Avaliação da efetividade na detecção de vulnerabilidades. Trabalho de graduação em segurança da informação, Faculdade de Tecnologia de Araraquara (FATEC), Centro Paula Souza.

Pressman, R. S. e Maxim, B. R. (2021). *Engenharia de Software: Uma Abordagem Profissional*. AMGH Editora.

PRILL (2026). O futuro da engenharia de software: Implicações chave e estratégias para 2026. *Tecnologia e Consultoria*.

Semerano, V. Z. e de Oliveira, L. F. (2024). O papel estratégico do analista de qualidade (qa) em equipes scrum, kanban e scrumban no desenvolvimento de software Ágil. *Advances in Global Innovation & Technology*, 2(2):32–45.