Análise Comparativa de Qualidade de Código na Refatoração de Sistemas Legados em ABAP com o Uso de Inteligência Artificial

Ana Carolina C. Corrêa

[1]Pontifícia Universidade Católica de Minas Gerais (PUC Minas)
Departamento de Engenharia de Software e Sistemas de Informação
Belo Horizonte - MG - Brasil

ana.correa.1313117@sga.pucminas.br

Abstract. *This study investigated the use of Artificial Intelligence models in the automated refactoring of legacy ABAP code. The objective was to evaluate whether these automated processes are able to update artifacts without compromising their structural quality or introducing new issues. A total of 25 public repositories were analyzed and processed by four generative models (ChatGPT, Gemini, Grok and Codestral). The legacy and refactored versions were compared using metrics extracted from SonarQube (complexity, maintainability and reliability) and ABAPLint (diversity of violations). The results indicate that ChatGPT and Gemini exhibited the best structural performance, while Codestral introduced the highest number of inconsistencies.*

Resumo. *Este estudo investigou o uso de modelos de Inteligência Artificial na refatoração automatizada de códigos legados em ABAP. O objetivo foi avaliar se esses processos automatizados conseguem atualizar artefatos sem comprometer sua qualidade estrutural nem introduzir novos problemas. Foram analisados 25 repositórios públicos, processados por quatro modelos generativos (ChatGPT, Gemini, Grok e Codestral). As versões legadas e refatoradas foram comparadas utilizando métricas extraídas pelo SonarQube (complexidade, manutenibilidade e confiabilidade) e pelo ABAPLint (diversidade de violações). Os resultados indicam que o ChatGPT e o Gemini apresentaram o melhor desempenho estrutural, enquanto Codestral introduziu o maior número de inconsistências.*

Bacharelado em Engenharia de *Software* - PUC Minas
Trabalho de Conclusão de Curso (TCC)

Orientador de conteúdo (TCC I): Joana Souza - joanasouza@pucminas.br
Orientador de conteúdo (TCC I): João Pedro Batisteli - jp.batisteli@hotmail.com
Orientador de conteúdo (TCC I): Leonardo Vilela - leonardocardoso@pucminas.br
Orientador acadêmico (TCC I): Cleiton Tavares - cleitontavares@pucminas.br
Orientador do TCC II: Michelle Hanne Soares de Andrade - michelleandrade@pucminas.br

Belo Horizonte, 15 de DEZEMBRO de 2025.

1. Introdução

A manutenção e evolução de sistemas legados permanecem como desafios recorrentes na Engenharia de Software, devido à sua presença em setores críticos e ao alto custo

de substituição [Bennett and Rajlich 2000]. Segundo Lehman (1980), sistemas de software tendem a acumular complexidade ao longo do tempo, tornando-se progressivamente mais difíceis de modificar e mais suscetíveis a falhas. Diante disso, uma alternativa à substituição completa é a refatoração, definida por Fowler (2019) como o processo de aprimorar a estrutura interna do código sem alterar seu comportamento externo. Entretanto, esse processo pode ser arriscado, principalmente quando aplicado a sistemas com alta complexidade estrutural, acoplamento excessivo e escassa documentação [Mens and Demeyer 2008].

A relevância deste estudo decorre dos riscos associados a refatorações mal conduzidas em sistemas desenvolvidos em ABAP, que podem introduzir vulnerabilidades e dificultar a manutenção a longo prazo. A compreensão desses sistemas é frequentemente limitada por dependências estruturais pouco documentadas, o que torna desafiadora a aplicação de refatorações seguras e eficazes. Nesse contexto, ganha importância a modernização de sistemas SAP (do alemão, *Systeme, Anwendungen und Produkte in der Datenverarbeitung*), amplamente desenvolvidos em ABAP, especialmente diante da migração do ambiente SAP ECC para o SAP S/4HANA, que impõe novas diretrizes técnicas. Portanto, o problema central deste trabalho reside na **dificuldade de realizar refatorações em sistemas legados desenvolvidos em ABAP sem gerar degradações estruturais nem novos problemas de qualidade**.

Para enfrentar esse desafio, são utilizados mecanismos que auxiliam na análise estrutural e na tomada de decisões, conferindo maior sistematização ao processo e reduzindo sua suscetibilidade a erros. Nesse contexto, destacam-se abordagens baseadas em Inteligência Artificial (IA), as quais vêm demonstrando potencial para automatizar refatorações e embasar decisões com maior rigor técnico [Liu et al. 2024]. A IA pode contribuir significativamente na modernização de sistemas legados e na identificação de riscos estruturais e de qualidade [Piastou 2024]. Além disso, estudos recentes exploram a aplicação de modelos de linguagem de grande escala (LLMs, do inglês, *Large Language Models*), como ChatGPT, para apoiar atividades de transformação de código a partir de instruções em linguagem natural.

O objetivo geral deste trabalho será **avaliar o uso de quatro modelos de Inteligência Artificial na refatoração de sistemas legados em ABAP, com foco em preservar a qualidade estrutural do código e evitar a introdução de novos problemas**. Como objetivos específicos, pretende-se: (i) avaliar a confiabilidade das refatorações automatizadas geradas pelos modelos de IA; (ii) avaliar a eficácia das refatorações automatizadas na preservação e melhoria de atributos de qualidade estrutural do código; e (iii) avaliar comparativamente os modelos de IA quanto à sua capacidade de refatorar código ABAP de forma consistente e alinhada às diretrizes do ambiente SAP S/4HANA.

Como resultado, este estudo apresenta evidências empíricas sobre o impacto de refatorações automatizadas por quatro modelos de IA (ChatGPT 4.1[1], Gemini[2], Grok[3] e Codestral[4]) aplicadas a códigos legados em ABAP. A análise de 25 repositórios na versão

[1] https://openai.com/gpt-4

[2] https://cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-5-pro?hl=pt-br

[3] https://x.ai/news/grok-3

[4] http://mistral.ai/news/codestral-2501

SAP ECC permitiu examinar em que medida as refatorações sugeridas contribuíram para a melhoria de atributos estruturais, evitando a introdução de novos problemas no código. A avaliação foi conduzida por meio de métricas quantitativas extraídas pelas ferramentas SonarQube[5] e o ABAPLint[6], oferecendo evidências sobre o potencial da abordagem proposta.

Este trabalho está organizado da seguinte forma: a Seção 2 apresenta o referencial teórico, abordando conceitos fundamentais sobre o estudo. A Seção 3 discute trabalhos relacionados que abordam análises semelhantes. A Seção 4 apresenta o detalhamento da metodologia adotada. A Seção 5 apresenta os resultados. A Seção 6 aborda as ameaças à validade. Por fim, na Seção 7, apresenta-se a conclusão.

2. Fundamentação Teórica

Para melhor compreensão dos conceitos que compõem as soluções abordadas neste estudo, esta seção apresenta a fundamentação teórica sobre o contexto tecnológico SAP ABAP, abordagens baseadas em IA para refatoração e análise estática.

2.1. Contexto Tecnológico SAP ABAP

A evolução da linguagem ABAP está associada à arquitetura técnica da plataforma SAP e ao componente *SAP_BASIS*, responsável pelos serviços fundamentais do sistema. No contexto do SAP *On-Premise*, essa evolução acompanha as gerações da própria plataforma. No SAP ECC, o ABAP evolui até o *SAP_BASIS 7.50*, considerado o limite da geração clássica. A partir desse ponto, surge o SAP S/4HANA *On-Premise*, no qual o ABAP passa a operar sobre o banco de dados em memória SAP HANA, exigindo adaptações sintáticas e a substituição de construções obsoletas, ainda que parte da compatibilidade com o ecossistema legado seja mantida.

O ABAP legado organiza o código em programas principais e arquivos *include*, permitindo a separação de unidades lógicas em múltiplos artefatos. Essa estrutura é complementada pelo *Data Dictionary* (DDIC), responsável pela definição centralizada de tabelas, domínios, estruturas e tipos usados pelo código-fonte. Esses elementos caracterizam o modelo tradicional de desenvolvimento e delimitam o escopo tecnológico dos repositórios analisados neste estudo.

2.2. Abordagens Baseadas em IA para Refatoração

LLMs são sistemas treinados para compreender e gerar linguagem natural a partir de grandes volumes de dados textuais. Esses modelos operam estimando a probabilidade de ocorrência de palavras ou *tokens* em um determinado contexto, possibilitando tarefas como tradução, sumarização e geração de texto [Eisenstein 2019]. Apesar de seu foco inicial, os LLMs vêm sendo explorados para interpretar e modificar código-fonte, devido à semelhança estrutural entre linguagem natural e linguagens de programação.

No contexto da refatoração, LLMs atuam como mecanismos de apoio capazes de sugerir ajustes estruturais em artefatos de software. Entretanto, esses modelos não oferecem garantias formais de preservação semântica, o que limita seu uso seguro em

[5] https://docs.sonarsource.com/SonarQube-server/10.8/analyzing-source-code/languages/abap

[6] https://abaplint.org/

sistemas legados ou ambientes críticos sujeitos a riscos operacionais [Liu et al. 2024]. Essa imprevisibilidade torna necessária a adoção de mecanismos de verificação estrutural, como a análise estática, para avaliar objetivamente os impactos das transformações.

2.3. Análise Estática

A análise estática de código é uma técnica que permite examinar a estrutura de um sistema sem executá-lo, com o objetivo de identificar anomalias, duplicações, violações de boas práticas e indícios de baixa manutenibilidade [McConnell 2004]. Essa abordagem é particularmente relevante em contextos de modernização de sistemas legados, nos quais a ausência de testes automatizados limita a verificação comportamental e torna necessário recorrer a mecanismos de inspeção estrutural.

Como parte desse processo, a análise estática atua como meio de fornecer visibilidade organizada sobre o software, contribuindo para a reestruturação e o controle de risco [Bennett and Rajlich 2000]. Em cenários marcados por código fragmentado, dependências obsoletas e documentação insuficiente, métricas sintáticas e heurísticas estruturais tornam-se essenciais para indicar qualidade de forma indireta e apoiar decisões em transformações que não podem ser validadas por execução.

Neste trabalho, adota-se essa perspectiva para verificar a efetividade das refatorações propostas por IAs generativas. A avaliação baseia-se em métricas geradas por ferramentas como o SonarQube e o ABAPLint. Esses dados funcionam como uma base de comparação estrutural entre o código original e o refatorado, permitindo inferir, de forma indireta e controlada, os efeitos das transformações aplicadas.

3. Trabalhos Relacionados

Estudos recentes investigaram a aplicação de Inteligência Artificial na refatoração automatizada de código, com ênfase na compreensão estrutural e na capacidade dos modelos de linguagem em propor transformações coerentes. Nesta seção, discutem-se quatro pesquisas que abordam temas diretamente relacionados a este trabalho.

O estudo de Liu et al. (2024) explorou o uso de LLMs como ferramenta de apoio à refatoração automatizada de códigos na linguagem Java. No experimento, baseado em 180 oportunidades de refatoração extraídas de 20 projetos reais, os autores demonstraram quantitativamente que o uso de *prompts* mais específicos e detalhados aumenta significativamente a taxa de sucesso das refatorações propostas, com o GPT-4 alcançando até 86,7% de acerto, enquanto estratégias genéricas resultaram em desempenhos substancialmente inferiores. Esses resultados sustentam a utilização de LLMs como apoio à refatoração, em consonância com o presente trabalho, que adota um *prompt* padronizado com o objetivo de assegurar comparabilidade entre modelos e analisar os impactos estruturais das transformações geradas.

O estudo de Bandarupalli (2025) investigou o uso de *Graph Neural Networks* (GNNs) na automatização da identificação de padrões de refatoração em grandes volumes de código-fonte. As GNNs permitem modelar relações internas do programa, como fluxos de controle e dependências estruturais, a partir de representações em grafos. Em uma avaliação com aproximadamente 365.000 amostras de código, ferramentas de análise estática como o SonarQube alcançaram cerca de 78% de acurácia na recomendação de refatorações, indicando que métricas estruturais são eficazes para apoiar a avaliação da

manutenibilidade. Esses resultados corroboram a adoção de ferramentas estáticas consolidadas para análise estrutural, em alinhamento com a abordagem deste trabalho.

O estudo conduzido por Piastou (2024) analisou a aplicação de IA em sistemas legados com histórico de manutenção prolongada, avaliando cinco bases de código reais. Os resultados indicaram que modelos de IA podem ser ajustados para lidar com elevados níveis de complexidade estrutural, alcançando cerca de 84% de acurácia na identificação desses padrões. Embora o autor reconheça limitações na compreensão semântica do código, o estudo reforça que abordagens baseadas em métricas estruturais são eficazes para apoiar a análise e a tomada de decisão em contextos de código legado, em consonância com o enfoque adotado neste trabalho.

O estudo de Santos e Gerosa (2018) investigou como 11 práticas distintas de codificação influenciam a legibilidade percebida por desenvolvedores, avaliando fatores como a clareza na nomeação de identificadores e o uso de estruturas de controle em trechos reais de código Java. Os autores observaram que a maioria dessas práticas apresentou impacto estatisticamente significativo, com destaque para aquelas relacionadas à nomeação de variáveis, especialmente o uso de palavras do dicionário em identificadores, que contribuíram positivamente para a legibilidade. Em contraste, aspectos como níveis de aninhamento exibiram percepções divergentes entre os participantes, sem evidências conclusivas de benefício. No contexto deste trabalho, essa pesquisa fornece base empírica para considerar a clareza e a inteligibilidade do código como dimensões relevantes de análise estrutural.

Essas pesquisas evidenciaram que a aplicação de IA à refatoração de software exige o balanceamento entre múltiplos critérios de qualidade. O presente projeto contribui de forma complementar ao comparar, de maneira sistemática, quatro modelos de Inteligência Artificial aplicados à refatoração de sistemas ABAP, avaliando seus resultados por meio de métricas estruturais.

4. Materiais e Métodos

Nesta seção, foram descritos os materiais e métodos utilizados na condução deste estudo. A pesquisa caracterizou-se como experimental, conforme os critérios metodológicos propostos por Gil (2002) e Lakatos e Marconi (2003). Tratou-se de uma investigação que realizou intervenções planejadas sobre trechos de código legados escritos em ABAP, por meio da aplicação de modelos de Inteligência Artificial para refatoração automatizada. A natureza experimental manifestou-se na execução controlada dessas transformações e na posterior medição de seus efeitos por meio de métricas estruturais de qualidade de software. O foco foi a observação empírica dos impactos gerados por essas refatorações, permitindo verificar, de forma quantitativa, como as alterações propostas afetaram a estrutura e as métricas de qualidade do código.

Nas subseções a seguir, foram abordados os métodos, estratégias, materiais e métricas utilizados para alcançar os objetivos propostos. As subseções incluem o Arranjo Experimental e as Métricas.

4.1. Arranjo Experimental

O trabalho foi conduzido em três etapas principais, conforme ilustrado na Figura 1. A primeira etapa consistiu na seleção e coleta de repositórios ABAP públicos hospedados

no site GitHub. Para tornar o processo reprodutível e escalável, foram desenvolvidos quatro *scripts* em Python, em conjunto com a API do GitHub, que automatizaram as fases de busca, ordenação, filtragem e enriquecimento de metadados, com os resultados salvos em arquivos CSV. Todos os repositórios utilizados foram clonados em 07 de setembro de 2025, garantindo consistência temporal no conjunto analisado.

Foram considerados inicialmente todos os repositórios criados desde 2008, totalizando 9.837, os quais foram ordenados em ordem decrescente de tamanho. Em seguida, aplicou-se um processo de filtragem baseado em palavras-chave nos nomes, descrições e estruturas, de forma a excluir aqueles relacionados à versão S/4HANA ou cujo propósito não estivesse diretamente associado ao desenvolvimento em ABAP. Após isso, o conjunto foi reduzido para 6.555 repositórios.

A partir desse conjunto filtrado, realizou-se uma verificação técnica para assegurar a compatibilidade com o ambiente SAP ECC, considerando critérios como a presença de construções sintáticas características dele e a ausência de artefatos específicos do SAP S/4HANA. Os primeiros 25 repositórios que atenderam integralmente a esses critérios foram selecionados para compor o conjunto experimental. Em seguida, os dados selecionados foram complementados por meio de chamadas adicionais à API do GitHub, a fim de coletar atributos complementares para a caracterização dos repositórios.

Durante a segunda etapa da pesquisa, foram aplicados quatro modelos de Inteligência Artificial (ChatGPT 4.1, Gemini, Grok e Codestral) para gerar refatorações automatizadas a partir de um *prompt* padronizado. Esse *prompt*, apresentado integralmente no Apêndice 8, especificou um conjunto rígido de regras que orientou as transformações solicitadas para a versão SAP S/4HANA *On-Premise*, incluindo atualização da sintaxe, adaptação para compatibilidade, renomeação de variáveis, substituição de construções obsoletas e preservação do fluxo lógico original. As instruções também exigiam que as IAs retornassem exclusivamente o código convertido, sem comentários ou explicações adicionais.

Essa estratégia permitiu que todos os modelos recebessem exatamente as mesmas instruções, garantiu comparabilidade entre as saídas e possibilitou avaliar, de forma controlada, o desempenho de cada modelo. As refatorações geradas foram armazenadas em arquivos separados, assegurando organização e rastreabilidade.

Embora não tenha sido possível medir com precisão o tempo total de execução por modelo, o processo apresentou apenas interrupções ocasionais decorrentes de limitações físicas do ambiente de execução. Ainda assim, o mecanismo de controle implementado no *pipeline* garantiu consistência entre as iterações, evitando o reprocessamento de arquivos já transformados. Os registros contínuos permitiram acompanhar o progresso e documentar todas as etapas do processo.

Na terceira etapa, foram empregados os softwares de análise estática SonarQube (*Developer Edition* 10.8), com suporte nativo a ABAP, e ABAPLint, ferramenta especializada em verificação sintática e de estilo para ABAP. O SonarQube foi utilizado para coletar métricas estruturais de complexidade, manutenibilidade e confiabilidade, enquanto o ABAPLint complementou a avaliação identificando inconsistências específicas da linguagem e violações de regras estabelecidas para o código ABAP.

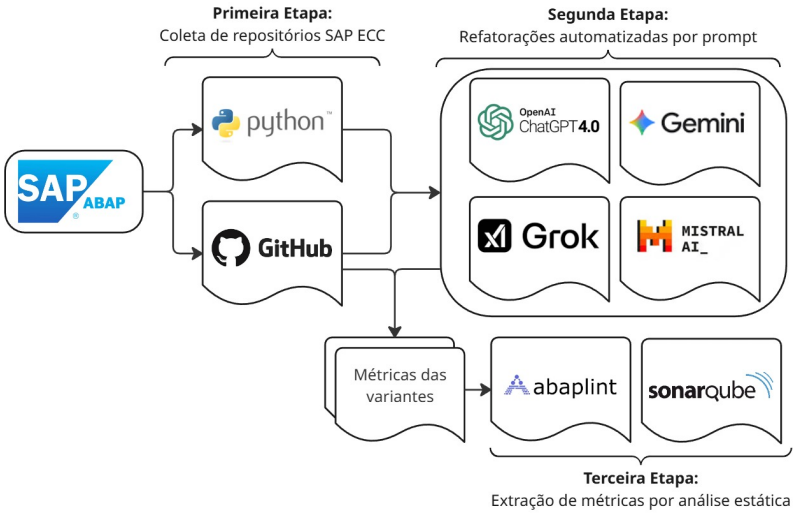

Figura 1. Arranjo experimental.

A extração do código ABAP para análise foi realizada por meio do clone das pastas correspondentes à *branch* principal de cada repositório, preservando a integridade dos arquivos originais. Cada repositório foi organizado em um diretório próprio, separado entre versões legadas e versões refatoradas. Durante a preparação dos dados, foram aplicadas regras de exclusão para remover arquivos irrelevantes, como documentação ou *scripts* auxiliares, garantindo que apenas arquivos ABAP fossem analisados.

A análise estática foi realizada utilizando o SonarScanner, responsável por enviar o código de cada repositório ao servidor SonarQube. Cada versão foi submetida separadamente, permitindo comparações diretas. Durante a execução, o SonarQube identificava automaticamente arquivos com extensões padrão ABAP e aplicava as regras de qualidade definidas no perfil da linguagem. Como os repositórios não continham testes automatizados, métricas de cobertura não puderam ser coletadas. Os resultados foram extraídos por meio da API do SonarQube, permitindo o armazenamento estruturado dos dados para análises subsequentes.

Após a coleta das métricas, realizou-se uma comparação entre os resultados dos códigos SAP ECC e suas respectivas versões refatoradas. Esse processo foi automatizado por *scripts* em Python 3.13 que processaram os arquivos exportados do SonarQube e do ABAPLint, correlacionando os valores de cada métrica entre as duas versões.

Para viabilizar a comparação entre métricas heterogêneas e minimizar distorções decorrentes do tamanho dos arquivos ou da escala absoluta dos valores, foi aplicado um processo de normalização proporcional. Cada métrica foi convertida em um índice normalizado na escala contínua entre 0 e 1, conforme a Equação 1. Valores mais próximos de zero indicam melhor desempenho estrutural, enquanto valores mais próximos de um representam maior degradação relativa.

$$I_{\text{norm}} = \frac{M_i}{M_{\text{max}}} \tag{1}$$

Nessa equação, M_i representa o valor agregado da métrica para uma determinada

variante e $M_{\max}$ corresponde ao maior valor observado entre todas as variantes avaliadas. Os índices agregados foram obtidos por meio de ponderação uniforme das submétricas pertencentes à mesma categoria, de modo a evitar vieses decorrentes da priorização arbitrária de indicadores individuais.

Os *scripts* também geraram visualizações comparativas por meio da biblioteca *Matplotlib*, o que permitiu inspecionar, de forma organizada, as diferenças entre as métricas das versões legadas e refatoradas. Esses gráficos serviram como apoio à análise, facilitando a interpretação dos resultados obtidos ao longo dos repositórios avaliados.

Todos os experimentos, com exceção do SonarQube, foram executados em um ambiente local composto por um notebook equipado com processador Intel Core i5-11400H, 8 GB de memória RAM, executando Windows 11. Já o SonarQube foi executado separadamente em um ambiente local composto por um *desktop* com processador AMD Ryzen 5 7600X, 32 GB de memória RAM, executando Windows 11 Pro.

4.2. Métricas

Na caracterização inicial dos repositórios selecionados, foram consideradas métricas básicas extraídas do GitHub, como tamanho do repositório, número total de estrelas e quantidade de *forks*, permitindo contextualizar o perfil e a escala dos projetos analisados. Para a avaliação dos efeitos das refatorações automatizadas, foram utilizadas quatro categorias principais de métricas. A maior parte delas foi extraída por meio do SonarQube, complementadas pelo ABAPLint quando aplicável.

- **Complexidade:** Inclui complexidade ciclomática, que quantifica o número de caminhos lógicos independentes no código e indica o grau de ramificação estrutural; e complexidade cognitiva, que estima o esforço mental necessário para compreender trechos específicos, considerando fatores como aninhamento, desvios de fluxo e estruturas difíceis de seguir.
- **Manutenibilidade:** Caracteriza o grau de limpeza estrutural e o esforço estimado para futuras intervenções. Abrange métricas como dívida técnica e seu índice, classificação de manutenibilidade e *code smells*, que representam indícios de má qualidade estrutural que não comprometem a execução imediata, mas tornam o código mais propenso a erros e mais difícil de evoluir.
- **Confiabilidade:** Compreende métricas como número de *bugs* e esforço de remediação de confiabilidade, permitindo identificar potenciais pontos de falha apontados pela análise estática.
- **Diversidade de Regras Violadas:** Representa a quantidade de regras distintas do ABAPLint violadas em cada arquivo, permitindo avaliar a dispersão dos tipos de problemas estruturais detectados.

A métrica de Diversidade de Regras Violadas considerou categorias macro do ABAPLint, como estilo, desempenho, segurança e consistência sintática. Essa métrica foi calculada apenas com regras aplicáveis ao contexto de análise por arquivo isolado. No total, 162 regras estavam ativas no processo de verificação, desconsiderando aquelas dependentes de DDIC, *main programs* SAP, *includes* completos ou objetos vinculados ao pacote original.

Este estudo também avaliou três subconjuntos específicos. O primeiro é construções obsoletas, que reúne regras que identificam o uso de comandos desconti-

nuados ou práticas superadas no ABAP moderno. O segundo engloba aspectos de estilo, que abrange verificações relacionadas à formatação e organização superficial do código, incluindo regras de indentação, espaçamento, estruturação de blocos e convenções de nomenclatura. O terceiro contempla aspectos de segurança, que apontam riscos potenciais, como ausência de validações obrigatórias, uso inseguro de instruções sensíveis e práticas capazes de comprometer a integridade de dados.

5. Resultados

Nesta seção, foram apresentados os resultados obtidos na Seção 4. Além disso, realizou-se a caracterização desses dados.

5.1. Caracterização do Conjunto de Dados

Com o objetivo de compreender melhor os 25 repositórios selecionados, realizou-se uma etapa de caracterização exploratória a partir dos metadados extraídos do GitHub. Foram analisados atributos como tamanho, popularidade e temporalidade, organizados em gráficos descritivos e no coeficiente de correlação de Pearson.

Neste estudo, o termo variantes refere-se às cinco versões produzidas para cada arquivo: SAP ECC (código legado original), ChatGPT, Codestral, Gemini e Grok. Do total de 13.739 arquivos ABAP presentes nesses repositórios, foi considerado apenas o subconjunto de 754 arquivos por variante, totalizando 3.770 códigos analisados. Esse subconjunto corresponde aos arquivos efetivamente processados simultaneamente pelos quatro modelos de IA.

A distribuição do tamanho dos repositórios, medida em *kilobytes*, mostrou predominância de projetos de pequeno porte, com a maior parte dos repositórios abaixo de 30.000 KB. Esse padrão indica que grande parte do código ABAP disponível em repositórios públicos corresponde a unidades modulares ou trechos isolados, enquanto arquivos maiores são menos frequentes e tendem a representar implementações mais extensas. Poucos repositórios ultrapassaram 45.000 KB, caracterizando a presença de alguns *outliers*. Entre os maiores volumes identificados, destacaram-se: ABAPCode, contendo exemplos e trechos de código ABAP genéricos; SAPGit, contendo implementações relacionadas ao controle de versão de objetos ABAP; e abap_nantest, que inclui *scripts* de teste e automação voltados à integração de sistemas SAP.

No conjunto geral, observou-se forte heterogeneidade. Em média, aproximadamente 37% dos arquivos dos repositórios corresponderam a código ABAP, com variação que partiu de cerca de 15% até casos em que ele representava praticamente a totalidade do projeto. Essa dispersão indicou a presença tanto de repositórios inteiramente dedicados à linguagem quanto de projetos nos quais ela aparecia apenas como parte complementar.

Quanto à popularidade, observou-se que a maioria dos projetos possuía zero ou poucas estrelas e *forks*, o que reflete baixo engajamento comunitário. Os resultados sugerem que, dentro do conjunto analisado, a visibilidade dos projetos relacionados a ABAP no GitHub tende a ser limitada, concentrando-se em poucos repositórios de maior destaque.

Entre as exceções, destacaram-se dois casos que concentram praticamente toda a atenção da comunidade: o abapGit, com mais de 1.600 estrelas e 500 *forks*, consolidado

como a principal ferramenta comunitária para versionamento de código ABAP em repositórios Git, e o ABAP-SDK-for-Azure, com 184 estrelas, que representou uma iniciativa oficial da Microsoft para integrar sistemas SAP a serviços do Azure, habilitando cenários híbridos de nuvem.

Entretanto, essas análises devem ser relativizadas, pois grande parte do desenvolvimento ABAP ocorre internamente na plataforma SAP, fora de repositórios públicos, o que naturalmente reduz o potencial de colaboração aberta.

Visando a realização de uma análise de correlação entre as variáveis quantitativas coletadas, foi utilizado o coeficiente de Pearson. Esse coeficiente, apresentado na Tabela 1, avaliou a intensidade e a direção das relações lineares entre tamanho, estrelas e *forks*.

Os resultados indicaram que o tamanho dos repositórios não apresentou correlação significativa com o número de estrelas (r = -0,03) nem com a quantidade de *forks* (r = -0,03), indicando que projetos maiores não são necessariamente mais populares ou reutilizados. Em contrapartida, observa-se uma correlação quase perfeita entre as variáveis estrelas e *forks* (r = 0,9999), sugerindo que a visibilidade e o reaproveitamento estão fortemente associados.

A Tabela 2 apresenta o resumo estatístico das variáveis analisadas no conjunto de repositórios. Os valores de média, mediana, desvio padrão, quartis e extremos das variáveis analisadas são apresentados. Os dados evidenciam elevada dispersão, marcada pela coexistência de valores nulos e de casos com magnitudes muito altas, corroborando a assimetria observada no conjunto.

Tabela 1. Coeficiente de correlação de Pearson entre variáveis quantitativas.

	Tamanho (KB)	Estrelas	Forks
Tamanho (KB)	1,000	0,030	0,032
Estrelas	0,030	1,000	0,9999
Forks	0,032	0,9999	1,000

Tabela 2. Resumo estatístico dos repositórios analisados.

	Tamanho (KB)	Estrelas	Forks
Média	23.705,64	75,76	26,16
Mediana	22.622,00	0,00	0,00
Desvio Padrão	14.724,13	332,31	113,53
1º Quartil	17.068,00	0,00	0,00
3º Quartil	24.461,00	1,00	1,00
Mín.	8.445,00	0,00	0,00
Máx.	76.506,00	1.661,00	568,00

Durante a refatoração, o sistema enviou requisições paralelas aos modelos hospedados nas plataformas GitHub Models e Google AI Studio, criando uma *thread* independente para cada IA. Cada instância aplicou o *prompt* definido no Apêndice 8, garantindo a padronização das instruções fornecidas aos modelos.

Os resultados foram armazenados em subpastas específicas para cada modelo dentro de um único diretório, preservando a hierarquia original dos repositórios. O controle da execução foi mantido em um arquivo CSV, estruturado com uma coluna para cada modelo de IA. Nesse arquivo, cada linha correspondia a um arquivo ABAP do repositório original, e cada coluna registrava as refatorações concluídas.

Além disso, falhas como limites de *tokens*, respostas vazias, tempo excedido ou erros de rede, também eram registradas em um CSV, possibilitando reexecuções seletivas apenas nos arquivos pendentes. Para lidar com restrições impostas pelas APIs, o sistema empregou um mecanismo automático de espera e reenvio, que pausava a *thread* correspondente ao modelo afetado e retomava o processamento assim que os limites eram liberados. Esse procedimento garantiu estabilidade durante execuções prolongadas e evitou perda de progresso.

5.2. Análise dos Resultados

Após a obtenção das versões refatoradas geradas pelos LLMs, realizou-se a análise dos resultados estruturais utilizando as métricas extraídas pelo SonarQube, apresentadas nas Figuras 2, 3 e 4, e pelo ABAPLint, representadas nas Figuras 5 e 6.

A Figura 2 apresenta a complexidade, que combina duas métricas estruturais: complexidade ciclomática e cognitiva. No índice normalizado, valores mais baixos indicam melhor desempenho, pois refletem estruturas menos ramificadas e de leitura mais simples. Por outro lado, valores mais altos indicam pior desempenho, sugerindo código mais complexo, com maior esforço cognitivo para entendimento e maior propensão a introduzir erros durante manutenção futura.

O Gemini exibiu o menor índice, servindo como referência estrutural do conjunto. Entre as demais variantes, o SAP ECC e o ChatGPT apresentaram aumento moderado de complexidade. O Grok mostrou elevação mais acentuada, enquanto o Codestral registrou o pior desempenho, apresentando o maior índice agregado. Esses resultados sugerem que alguns modelos preservaram mais fielmente a organização lógica do código legado, ao passo que outros introduziram estruturas mais profundas ou mais difíceis de interpretar.

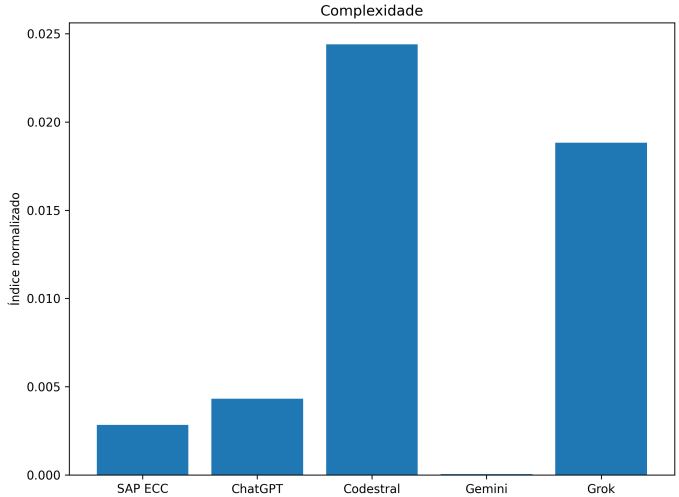

Figura 2. Índice agregado de complexidade.

No índice agregado de manutenibilidade, valores mais baixos representam melhor desempenho, pois refletem código mais limpo, com menor acúmulo de problemas estruturais e menor esforço necessário para futuras intervenções. Valores mais altos indicam pior desempenho, sugerindo maior deterioração estrutural e aumento do custo de manutenção.

A Figura 3 mostra que o Gemini apresentou novamente o menor índice; o SAP ECC manteve desempenho semelhante, enquanto o ChatGPT exibiu aumento moderado em relação a ele. Já o Codestral e o Grok registraram os maiores valores, com o Grok apresentando o pior desempenho, indicando que ambos introduziram mais estruturas problemáticas, elevando o esforço potencial para evolução e manutenção do código.

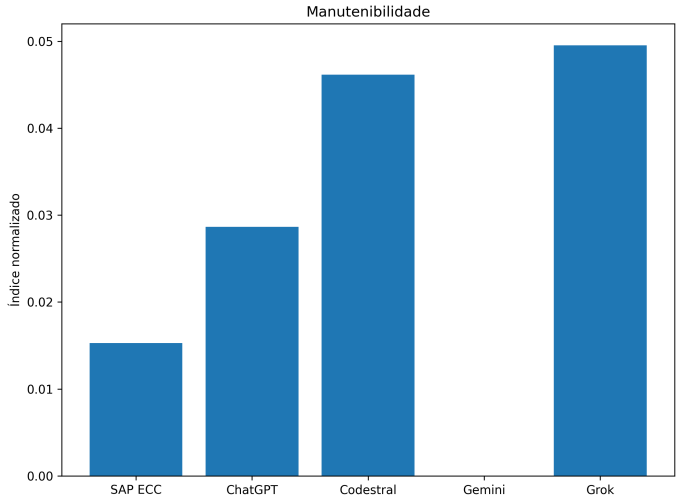

Figura 3. Índice agregado de manutenibilidade.

A Figura 4 apresenta o índice agregado de confiabilidade, no qual valores iguais a zero representam melhor desempenho, pois indicam ausência de falhas estruturais identificadas pela análise estática. Valores maiores que zero indicam pior desempenho, sugerindo a presença de problemas que podem comprometer o comportamento do sistema ou exigir correções adicionais.

O SAP ECC, ChatGPT, Gemini e Grok apresentaram índice igual a zero, demonstrando que, para essas variantes, a refatoração não comprometeu a confiabilidade do código. Apenas o Codestral exibiu valor diferente de zero, concentrando todas as ocorrências de *bugs* identificadas entre as versões analisadas. Esse resultado sugere que, enquanto os demais modelos preservaram a integridade estrutural dos arquivos, o Codestral introduziu alterações que elevaram o risco de falhas potenciais.

As Figuras 5 e 6 referentes à métrica de Diversidade de Regras Violadas, apresentam a comparação de diversidade entre as variantes e a média por arquivo. Além da diversidade total, três subconjuntos de regras foram avaliados separadamente: **construções obsoletas, estilo e segurança**. Os *outliers* observados nessas figuras foram definidos pelo critério estatístico padrão de diagramas de caixa, considerando valores acima de 1,5 vezes o intervalo interquartil (IQR), representando arquivos com quantidade significativamente maior de regras distintas violadas.

O SAP ECC apresentou baixa diversidade, com a maior parte dos arquivos in-

fringindo um conjunto reduzido de regras, o que indica maior uniformidade estrutural e menor propagação de problemas entre categorias. Entre as IAs, o ChatGPT e o Grok exibiram resultados mais próximos ao padrão observado no SAP ECC, sugerindo maior estabilidade e menor variação das violações introduzidas. O Gemini apresentou aumento moderado na diversidade, enquanto o Codestral demonstrou a maior dispersão e o maior número de *outliers*, caracterizando maior heterogeneidade e variabilidade estrutural nas refatorações produzidas.

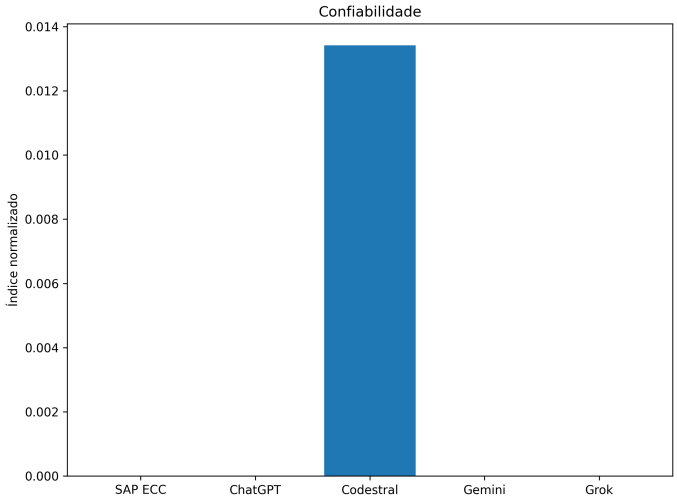

Figura 4. Índice agregado de confiabilidade.

No subconjunto de construções obsoletas, o SAP ECC e o Codestral apresentaram as maiores ocorrências, enquanto ChatGPT, Gemini e Grok reduziram significativamente o uso de comandos descontinuados. Já no de estilo, que reúne verificações relacionadas à formatação e organização superficial do código, o SAP ECC manteve baixa incidência, padrão reproduzido por ChatGPT, Grok e Gemini. O Codestral, por sua vez, apresentou aumento expressivo nesse tipo de violação. No subconjunto de segurança, que avalia riscos decorrentes de práticas inseguras ou ausência de validações, SAP ECC e ChatGPT registraram valores praticamente nulos. Grok permaneceu baixo, Gemini apresentou pequenas oscilações e o Codestral concentrou as maiores ocorrências.

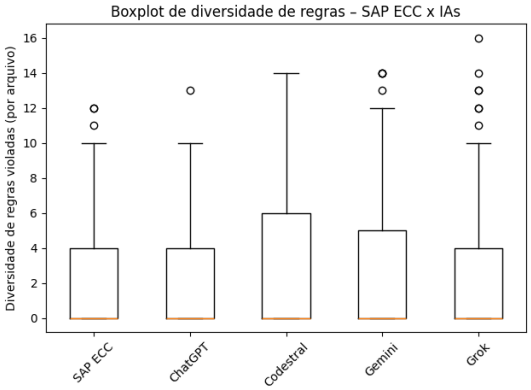

Figura 5. Diversidade de regras violadas.

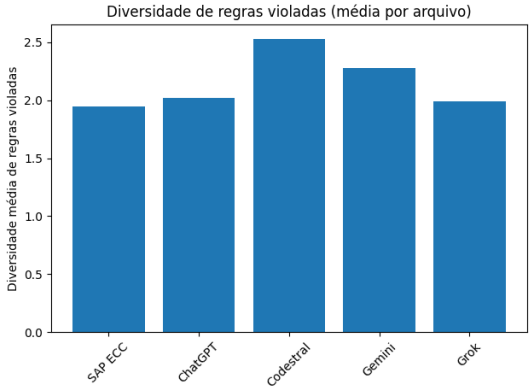

Figura 6. Diversidade média por variante.

6. Ameaças à Validade

Nesta seção, detalham-se as principais ameaças à validade identificadas neste estudo sobre o uso de modelos de Inteligência Artificial na refatoração de sistemas legados em ABAP, assim como as estratégias adotadas para minimizá-las.

Ameaças à Validade Interna: Uma limitação relevante decorre do comportamento não determinístico dos modelos de linguagem, que podem gerar saídas distintas em função de parâmetros internos, latência de rede ou restrições de *tokens*. Adicionalmente, não foi realizada qualquer verificação funcional, teste de execução ou validação semântica dos artefatos produzidos. Dessa forma, não há garantia de preservação completa do comportamento original, uma vez que a análise se restringiu a métricas estruturais. Para mitigar essas ameaças, foram adotados *prompts* padronizados, parâmetros fixos e controle rígido sobre a execução paralela, garantindo consistência entre as variantes.

Ameaças à Validade Externa: A maioria dos sistemas ABAP não está disponível publicamente, diminuindo a representatividade dos artefatos. Dessa forma, os resultados refletem predominantemente o comportamento dos modelos em código aberto, que tende a ser mais fragmentado e heterogêneo do que o código de produção de empresas. Como medida de mitigação, buscou-se diversidade entre os repositórios, em tamanho, propósito e popularidade, para ampliar a representatividade do conjunto.

Ameaças à Validade de Construção: As ferramentas SonarQube e ABAPLint, embora amplamente utilizadas e com suporte nativo a ABAP, não foram concebidas para avaliar especificamente transformações geradas por modelos de IA. Assim, podem não capturar nuances de clareza semântica ou coerência lógica. Além disso, métricas estruturais não abrangem integralmente atributos qualitativos como legibilidade e expressividade. Para reduzir esse viés, as métricas foram analisadas em comparação com antes e depois da refatoração, e interpretadas conforme as normas ISO/IEC 25010 e os princípios de *Clean ABAP*.

7. Conclusão

Este trabalho investigou o uso de modelos de Inteligência Artificial na refatoração automatizada de código ABAP originalmente desenvolvido para o ambiente SAP ECC, avaliando em que medida as transformações produzidas por esses modelos se aproximam das práticas estruturais compatíveis com o SAP S/4HANA. A análise baseou-se em métricas de complexidade, manutenibilidade e confiabilidade fornecidas pelo SonarQube, bem como na diversidade e natureza das violações identificadas pelo ABAPLint.

Os resultados mostraram que ChatGPT e Gemini foram os modelos com melhor desempenho global. Ambos mantiveram índices de complexidade e manutenibilidade próximos aos do ECC, sem introduzir problemas críticos de confiabilidade. Esses modelos também exibiram diversidade moderada de regras violadas no ABAPLint, demonstrando capacidade de preservar a organização e o estilo estrutural do código legado, mesmo após a modernização sintática solicitada.

O Grok, embora tenha se destacado no ABAPLint, apresentou desempenho inferior nas métricas do SonarQube, especialmente em complexidade e manutenibilidade. Isso indica que, apesar de gerar código estruturalmente estável sob o ponto de vista sintático, suas refatorações podem aumentar o esforço de leitura e manutenção.

O Codestral obteve o pior desempenho em todas as métricas analisadas. Apresentou a maior diversidade de regras violadas, maior quantidade de construções obsoletas, mais violações de estilo e o único caso de problemas críticos de confiabilidade. Esses resultados sugerem que suas transformações foram menos consistentes e introduziram maior heterogeneidade estrutural.

De forma geral, os resultados indicam que a refatoração automática mediada por IAs é tecnicamente viável, especialmente quando conduzida por modelos que preservam os atributos estruturais do código, com destaque para o Gemini, que apresentou o melhor desempenho geral segundo os critérios adotados. Entretanto, o desempenho mostrou-se significativamente variável entre os modelos analisados, evidenciando a necessidade de avaliações criteriosas antes da adoção dessas ferramentas em cenários corporativos.

Como trabalhos futuros, poderão ser incorporadas a validação funcional, a análise semântica, a integração com objetos completos da plataforma SAP e a realização de testes com modelos mais recentes ou especializados em código, de modo a ampliar a compreensão sobre o papel das IAs na modernização e manutenção evolutiva de sistemas empresariais.

Pacote de Replicação

O pacote de replicação deste trabalho encontra-se disponível em:

`https://github.com/ICEI-PUC-Minas-PPLES-TI/plf-es-2025-1-tcci-0393100-pes-ana-correa`

8. Apêndices

8.1. Prompt Utilizado nas Refatorações Automatizadas

```
Convert the ABAP code below (ECC) to ABAP compatible with SAP S/4HANA On-Premise.

Mandatory rules:
- Preserve functional logic and control flow.
- Update syntax to modern ABAP (inline DATA/FINAL, @ host variables, modern Open SQL, EXPRESSIONS).
- Update database calls (Open SQL) to S/4HANA syntax; do NOT create CDS, views, or any additional artifacts.
- Replace obsolete statements/features with supported equivalents in S/4HANA On-Premise; if no safe substitute exists, keep the original statement.
- Rename variables to remove Hungarian prefixes (e.g., fs_, lv_, ls_, lt_, lr_, lo_, gv_, mv_) and generic names.
    Apply Clean ABAP naming: descriptive names, no prefixes, lowerCamelCase for local variables and parameters; classes/Interfaces in PascalCase; constants in UPPER_SNAKE_CASE.
    Rename FIELD-SYMBOLS and related DATA refs to coherent, descriptive names.
    Update all usages to the new names.
- Remove only redundant/useless code; otherwise, replace. If a safe replacement is not possible, keep it.
- Do not change public external interfaces (APIs) without functional need.
- Output must compile on S/4HANA On-Premise.
- Do NOT write explanations, comments, or any text outside the code.
- Return **only** the converted source code.

Input code:
```

Listing 1. Prompt utilizado para refatoração automatizada nas IAs.

Referências

Bandarupalli, G. (2025). Ai-driven code refactoring: Using graph neural networks to enhance software maintainability. *arXiv preprint arXiv:2504.10412*.

Bennett, K. H. and Rajlich, V. T. (2000). Software maintenance and evolution: A roadmap. *Proceedings of the Conference on The Future of Software Engineering*.

dos Santos, R. M. and Gerosa, M. A. (2018). Impacts of coding practices on readability.

Eisenstein, J. (2019). *Introduction to Natural Language Processing*. The MIT Press.

Fowler, M. (2019). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, 2nd edition.

Gil, A. C. (2002). *Como Elaborar Projetos de Pesquisa*. Atlas, São Paulo, 4 edition.

Lakatos, E. M. and Marconi, M. d. A. (2003). *Fundamentos de Metodologia Científica*. Atlas, São Paulo, 5 edition.

Lehman, M. M. (1980). Programs, life cycles, and laws of software evolution. In *Proceedings of the IEEE*, volume 68.

Liu, B., Jiang, Y., Zhang, Y., Niu, N., Li, G., and Liu, H. (2024). An empirical study on the potential of llms in automated software refactoring. *arXiv preprint arXiv:2411.04444*.

McConnell, S. (2004). *Code Complete: A Practical Handbook of Software Construction*. Microsoft Press, Redmond, WA, 2nd edition.

Mens, T. and Demeyer, S. (2008). *Software Evolution*. Springer, Berlin.

Piastou, M. (2024). Overcoming challenges in applying ai guidance to complex and legacy codebases. *Journal of Artificial Intelligence Research*.