FACULDADE DE CIÊNCIAS E TECNOLOGIA

DEPARTAMENTO DE INFORMÁTICA

MIGUEL VIEIRA DE ALMEIDA

Licenciatura em Engenharia Informática

INTEGRAÇÃO DA INTELIGÊNCIA ARTIFICIAL NO PROCESSO DE REVISÃO DE CÓDIGO NO CICLO DE VIDA DE DESENVOLVIMENTO DE SOFTWARE

Plano de Dissertação
MESTRADO EM ENGENHARIA INFORMÁTICA

Universidade NOVA de Lisboa

Draft: *8 de janeiro de 2026*

FACULDADE DE
CIÊNCIAS E TECNOLOGIA

INTEGRAÇÃO DA INTELIGÊNCIA ARTIFICIAL NO PROCESSO DE REVISÃO DE CÓDIGO NO CICLO DE VIDA DE DESENVOLVIMENTO DE SOFTWARE

MIGUEL VIEIRA DE ALMEIDA

Licenciatura em Engenharia Informática

Orientador: João Sousa
Consultor Sénior, Processware

Coorientador: Jorge Cruz
Professor, NOVA University Lisbon

Plano de Dissertação
MESTRADO EM ENGENHARIA INFORMÁTICA

Universidade NOVA de Lisboa
Draft: *8 de janeiro de 2026*

Resumo

Por fazer...

Abstract

To be Made

Índice

Índice de Figuras

Índice de Tabelas

Siglas

API Application Programming Interface *(p. 13)*
AST Abstract Syntax Tree *(pp. 8, 15)*

CI Continuous Integration *(pp. 7, 13)*
CR Code Review *(pp. iii, 1–10, 12–14)*

Diff Diferença de código entre versões (*diff*) *(pp. 3, 4, 9–11, 13, 15)*
DL Deep Learning *(pp. 8, 9)*

IDE Integrated Development Environment *(pp. iii, 15)*

LLM Large Language Model *(pp. iii, 1–4, 8–12, 14–16)*

ML Machine Learning *(pp. 7, 8)*

NLP Natural Language Processing *(p. 8)*

PR Pull Request *(pp. 1, 3, 4, 6, 9–15)*

1 Introdução

Este capítulo tem como objetivo introduzir o tema e objetivos desta dissertação. É apresentada a motivação para a realização do projeto, o contexto em que o problema abordado se insere e os objetivos a serem alcançados.

1.1 Motivação

O aumento da complexidade do software moderno e o ritmo acelerado de desenvolvimento tornam a CR uma etapa essencial no ciclo de vida do software. Este processo garante a qualidade, segurança e conformidade do código com as boas práticas de engenharia e com os padrões internos definidos pelas organizações [17, 4].

Contudo, a CR continua a ser uma tarefa exigente e demorada, que frequentemente exige múltiplos revisores para evitar a introdução de erros [10]. Em ambientes empresariais com grande volume de commits e Pull Requests (PRs), os revisores lidam diariamente com alterações extensas que exigem análise cuidadosa. Isto pode originar atrasos, inconsistências e risco de falhas quando as mudanças chegam ao produto final [13, 4].

No contexto específico desta dissertação, estes desafios tornam-se evidentes. A Processware mantém uma base de código ampla, distribuída por vários repositórios e com múltiplas tecnologias — como C, C++, C#, Vue, SQL — coexistindo com código legacy. Esta diversidade aumenta a complexidade da revisão e exige conhecimentos técnicos específicos por parte dos revisores [21]. Além disso, diferentes equipas podem aplicar práticas distintas, dificultando a uniformização de padrões internos e levando a revisões repetitivas e demoradas [10].

Com o avanço recente dos LLMs, já amplamente demonstrado em ferramentas como ChatGPT, Gemini ou GitHub Copilot, tornou-se evidente que estes modelos têm capacidade para compreender e analisar código-fonte, identificar problemas e sugerir melhorias [4, 12]. Assim, a adoção de LLMs na revisão de código surge como uma oportunidade para reduzir esforço manual, aumentar a consistência e melhorar a eficiência do processo [13, 21].

1.2 Contexto

Esta dissertação enquadra-se no conjunto de soluções desenvolvidas pela Processware para melhorar o processo interno de revisão de código.

A Processware é uma empresa tecnológica com presença internacional, especializada no desenvolvimento de soluções para otimização de processos e gestão operacional em ambientes empresariais complexos. A sua atividade centra-se na transformação digital de operações através da combinação de engenharia de processos e plataformas tecnológicas proprietárias, apoiando organizações na melhoria da eficiência, controlo e escalabilidade das suas operações.

No centro da sua oferta encontra-se a plataforma O2P, uma solução baseada na cloud que permite a digitalização, monitorização e otimização de processos operacionais em tempo real, integrando-se com sistemas existentes e suportando diferentes contextos tecnológicos. A empresa atua em sectores com elevada complexidade operacional, caracterizados por múltiplas tecnologias, elevados volumes de dados e necessidade de elevada fiabilidade.

Neste contexto, a Processware mantém uma base de código extensa e heterogénea, distribuída por vários repositórios e tecnologias, o que coloca desafios significativos ao nível da manutenção e da revisão de código. A necessidade de garantir qualidade, consistência e alinhamento com boas práticas internas torna o processo de CR um elemento crítico no seu ciclo de desenvolvimento de software.

Atualmente, o processo de CR é totalmente manual e depende fortemente da experiência dos revisores. O elevado volume de alterações, a variedade tecnológica e a coexistência de código moderno com código legacy tornam a revisão uma tarefa exigente e pouco escalável [13]. Esta situação resulta em tempos de análise elevados e inconsistências entre equipas, dificultando a padronização de práticas internas [21].

Dado este cenário, torna-se pertinente explorar soluções baseadas em inteligência artificial, capazes de apoiar a análise de código e contribuir para um processo de CR mais eficiente, coerente e alinhado com as necessidades atuais de desenvolvimento de software. Estudos recentes mostram que LLMs podem atuar como revisores assistentes, melhorar a qualidade dos comentários, aumentar a uniformidade e acelerar a deteção de problemas [17, 19].

1.3 Definição do Problema

O processo de CR na Processware é atualmente manual, exigindo que os revisores analisem cada alteração submetida nos diferentes repositórios e tecnologias utilizadas pela empresa. Consequentemente, esta abordagem apresenta várias limitações. Numa primeira instância, a revisão é uma tarefa demorada e dependente das competências técnicas dos revisores, o que pode levar ao aumento da probabilidade de ocorrerem erros e inconsistências [10]. A coexistência de múltiplas tecnologias e de código legacy também dificultam

na identificação de problemas numa forma sistemática [17].

Além disso, revisores poderão adotar critérios distintos, dependendo da PR em questão, resultando em comentários heterogéneos e dificultando a uniformização de práticas internas. Estudos recentes mostram que esta falta de consistência é um problema comum em ambientes empresariais e afeta tanto a qualidade das revisões de código como a eficiência deste processo [21]. Outro problema recorrente é a necessidade de realizar comentários repetitivos, como correções de estilo, validações simples ou alertas sobre riscos conhecidos, que consomem tempo e poderiam ser automatizados [10].

Com os avanços recentes em LLMs, surgiu a oportunidade de automatizar grande parte deste processo. Estes modelos têm demonstrado capacidade para analisar código, identificar problemas e gerar comentários úteis [20]. No entanto, existem registos que apontam limitações: a utilidade dos comentários varia significativamente, alguns modelos tendem a gerar falsos positivos e a eficácia depende da clareza e pertinência das recomendações produzidas [13][19]. Importa ainda referir que LLMs genéricos não compreendem as práticas e padrões internos de cada organização, reduzindo a sua aplicabilidade prática durante o processo de CR.

Assim, o problema central que esta dissertação procura resolver consiste na ausência de um sistema capaz de apoiar a CR na Processware, produzindo comentários automáticos, alinhados com as boas práticas internas, reduzindo esforço repetitivo e contribuindo para um processo mais consistente, eficiente e escalável. Este desafio está alinhado com a necessidade de adaptar LLMs a contextos organizacionais específicos, de forma a tirar partido do seu potencial no processo de CR [14][4].

1.4 Objetivos

O principal objetivo desta dissertação é desenvolver um sistema capaz de apoiar o processo de CR na Processware, fornecendo comentários automáticos consistentes com as boas práticas internas e ajudando a reduzir o esforço associado a tarefas repetitivas. Pretende-se criar uma solução que aumente a eficiência, a uniformidade e a qualidade das revisões de código realizadas na empresa.

Para atingir este propósito, foram definidos os seguintes objetivos:

1. **Identificação e formalização de boas práticas internas:** analisar o código existente e recolher conhecimento junto da equipa responsável pelo processo de CR para mapear os padrões, convenções e recomendações atualmente aplicados neste processo.

2. **Análise inteligente de alterações de código:** desenvolver um sistema capaz de interpretar Diferença de código entre versões (*diff*)s (Diffs) e compreender o contexto das alterações submetidas, tendo em conta as várias tecnologias utilizadas na Processware.

3. **Geração de comentários consistentes e acionáveis:** criar um mecanismo que produza comentários claros, concisos e alinhados com as boas práticas internas, ajudando os revisores a detetar problemas recorrentes e a manter a consistência entre PRs.

4. **Integração fluida no fluxo de desenvolvimento:** garantir que a solução pode ser incorporada no processo atual de revisão de código, permitindo que os comentários sejam disponibilizados durante a análise de PRs.

5. **Avaliação da eficácia e utilidade prática:** validar o sistema através de métricas objetivas e subjetivas, garantindo que a solução contribui para revisões mais eficientes e uniformes.

Este conjunto de objetivos visa assegurar que o sistema desenvolvido traz valor real ao processo de CR da Processware, promovendo maior qualidade, consistência e produtividade no desenvolvimento de software.

1.5 Solução Proposta

A solução proposta consiste no desenvolvimento de um sistema de apoio à CR baseado em LLMs, capaz de analisar automaticamente alterações submetidas nas PRs e gerar comentários alinhados com as boas práticas internas da Processware. O sistema pretende atuar como um assistente de revisão, reduzindo tarefas repetitivas, promovendo maior consistência entre equipas e aumentando a eficiência do processo.

A proposta assenta nos seguintes componentes principais:

1. **Extração e pré-processamento das alterações de código:** o sistema recebe o Diff da PR e transforma-o numa representação adequada para análise, garantindo que apenas as alterações relevantes são processadas.

2. **Interpretação das mudanças com suporte de LLMs:** um modelo de linguagem é utilizado para compreender o contexto das alterações, identificar possíveis problemas e reconhecer padrões que violam boas práticas previamente definidas.

3. **Geração de comentários automáticos:** com base na análise realizada, o sistema produz comentários claros, concisos e acionáveis, seguindo as recomendações internas da Processware e facilitando o trabalho dos revisores humanos.

4. **Módulo de personalização de boas práticas:** permite adaptar regras e orientações às necessidades da empresa, garantindo que os comentários automáticos refletem padrões internos e preferências específicas de diferentes equipas.

5. **Integração com o fluxo de desenvolvimento existente:** o sistema é incorporado diretamente no processo atual da Processware, permitindo que os comentários sejam adicionados nas PRs através das ferramentas já utilizadas pela equipa.

6. **Monitorização e melhoria contínua:** a solução recolhe métricas e feedback dos utilizadores para identificar oportunidades de melhoria, ajustando o comportamento do sistema de forma progressiva ao longo do tempo.

Em conjunto, estes componentes constituem uma solução prática e extensível, preparada para apoiar revisores humanos e tornar o processo de CR mais eficiente, consistente e alinhado com as necessidades reais da empresa.

No próximo capítulo serão apresentadas as várias tecnologias a serem utilizadas para o desenvolvimento deste projeto, bem como a análise de trabalhos que sejam considerados relevantes para o enquadramento deste tema de dissertação.

2

Conceitos

Este capítulo tem como objetivo apresentar os principais conceitos teóricos e técnicos que sustentam o desenvolvimento desta dissertação. São abordados os fundamentos relacionados com o processo de CR no contexto do desenvolvimento de *software*, bem como a evolução das abordagens de automação aplicadas a este processo.

2.1 *Code Review* no Desenvolvimento de *Software*

CR é uma prática fundamental no desenvolvimento de software que consiste na análise sistemática do código-fonte por um ou mais programadores, com o objetivo de identificar defeitos, melhorar a qualidade do código e garantir a conformidade com boas práticas e padrões definidos pela organização [2, 15]. Esta atividade é normalmente realizada antes da integração de alterações no ramo principal do projeto, assumindo um papel central nos fluxos de desenvolvimento modernos baseados em PRs.

Historicamente, a revisão de código surgiu como uma prática formal, associada a inspeções estruturadas, como as inspeções de Fagan, que demonstraram benefícios significativos na deteção precoce de erros e na redução de custos associados à correção de defeitos em fases avançadas do desenvolvimento [5]. Com a evolução das metodologias ágeis e das plataformas colaborativas de desenvolvimento, a revisão de código tornou-se um processo mais leve e contínuo, integrado no ciclo diário de desenvolvimento de software [15].

Os principais objetivos da revisão de código incluem a deteção de erros lógicos e defeitos funcionais, a melhoria da legibilidade e manutenibilidade do código, a verificação da conformidade com padrões de estilo e boas práticas, bem como a mitigação de riscos relacionados com segurança e desempenho [11]. Para além dos aspetos técnicos, a revisão de código desempenha também um papel relevante na partilha de conhecimento entre membros da equipa, promovendo a disseminação de boas práticas e contribuindo para a uniformização do código ao longo do projeto [2].

Apesar dos seus benefícios comprovados, a revisão de código é uma atividade exigente

do ponto de vista cognitivo e temporal, dependendo fortemente da experiência e disponibilidade dos revisores. Em projetos de grande escala, caracterizados por elevados volumes de alterações e diversidade tecnológica, este processo pode tornar-se difícil de escalar, originando atrasos, inconsistências e variabilidade na qualidade das revisões realizadas [2, 11].

2.2 Automação do Processo de *Code Review*

Com o aumento da complexidade dos sistemas de software e do volume de alterações submetidas diariamente, surgiu a necessidade de melhorar o processo de CR através de mecanismos de automação. O principal objetivo destas abordagens é reduzir o esforço manual associado a tarefas repetitivas, melhorar a deteção precoce de defeitos e apoiar os revisores humanos na análise das alterações de código [2].

As primeiras tentativas de automação do CR basearam-se essencialmente em ferramentas de análise estática de código. Estas ferramentas analisam o código-fonte sem necessidade de execução, recorrendo a regras pré-definidas para identificar problemas como violações de estilo, padrões perigosos, possíveis erros de execução ou más práticas comuns [10]. Linters e analisadores estáticos tornaram-se parte integrante dos fluxos de desenvolvimento modernos, sendo frequentemente integrados em pipelines de Continuous Integration (CI).

Apesar dos benefícios associados à sua utilização, estas abordagens apresentam limitações significativas. Estudos mostram que ferramentas de análise estática tendem a gerar um elevado número de falsos positivos, o que pode levar à sua desvalorização por parte dos programadores [7]. Além disso, estas ferramentas possuem uma compreensão limitada do contexto em que o código é desenvolvido, não conseguindo capturar a intenção do programador, decisões de design ou requisitos específicos do domínio da aplicação [8].

Outra limitação relevante prende-se com a rigidez das regras utilizadas. A adaptação destas ferramentas a práticas internas ou a estilos específicos de uma organização requer frequentemente configurações manuais complexas, dificultando a sua adoção em ambientes empresariais com grande diversidade tecnológica [10]. Como consequência, a automação tradicional do CR tende a ser eficaz na deteção de problemas simples e bem definidos, mas menos adequada para identificar defeitos mais complexos ou fornecer comentários contextualizados e acionáveis.

Estas limitações motivaram a investigação de abordagens mais avançadas para a automação do processo de revisão de código, explorando técnicas de Machine Learning (ML) capazes de aprender padrões diretamente a partir de grandes volumes de código-fonte e dados históricos de desenvolvimento [1]. Estas abordagens abriram caminho para soluções mais flexíveis e contextuais, que serão discutidas nas secções seguintes.

2.3 Aprendizagem Automática e *Deep Learning* em *Code Review*

As limitações das abordagens tradicionais de automação do CR motivaram a utilização de técnicas de ML e, posteriormente, de Deep Learning (DL). Ao contrário das ferramentas baseadas em regras fixas, estas abordagens permitem aprender padrões diretamente a partir de grandes volumes de código-fonte e dados históricos de desenvolvimento, reduzindo a necessidade de configuração manual e aumentando a capacidade de generalização [1].

Os primeiros trabalhos nesta área exploraram modelos estatísticos e técnicas de aprendizagem supervisionada para identificar padrões de defeitos, prever a probabilidade de erros e priorizar avisos gerados por ferramentas automáticas [8]. Estas abordagens demonstraram que informação histórica, como alterações anteriores e decisões tomadas em revisões passadas, pode ser utilizada para melhorar a eficácia do processo de CR.

Com o avanço de técnicas de DL, surgiram modelos capazes de representar o código-fonte de forma mais expressiva, explorando a sua estrutura sintática e semântica. Trabalhos baseados em redes neuronais profundas passaram a utilizar representações como Abstract Syntax Tree (AST), grafos de fluxo de controlo e sequências de *tokens*, permitindo capturar relações mais complexas entre diferentes componentes do código base [16]. Estes modelos mostram melhorias significativas em tarefas como deteção de defeitos, sugestões de correções e identificação de padrões recorrentes no código.

Outro avanço relevante foi a introdução do conceito de *naturalness* do software, que assume que o código-fonte apresenta regularidades semelhantes às da linguagem natural. Estudos demonstraram que código que se desvia destes padrões tende a estar mais associado a defeitos, o que abriu caminho à aplicação de Natural Language Processing (NLP) à análise de código [6]. Esta perspetiva reforçou a adequação de técnicas originalmente desenvolvidas para processamento de linguagem natural ao domínio do desenvolvimento de software.

Apesar dos progressos alcançados, as abordagens baseadas em ML e DL apresentavam ainda algumas limitações práticas. Muitos modelos exigiam grandes volumes de dados rotulados, eram dispendiosos do ponto de vista computacional e apresentavam dificuldades em lidar com múltiplas linguagens de programação ou contextos organizacionais específicos [1]. Além disso, a integração destas soluções nos fluxos de desenvolvimento reais permanecia um desafio.

Estas limitações criaram as condições para a adoção de LLMs, capazes de aprender representações genéricas de código e de transferir conhecimento entre diferentes tarefas e linguagens. A utilização de LLMs no processo de CR será abordada na secção seguinte.

2.4 Modelos de Linguagem de Grande Escala

Os LLMs representam a evolução mais recente no campo da aprendizagem profunda aplicada à linguagem. São modelos baseados na arquitetura *Transformer*, treinados com grandes quantidades de dados textuais e, no contexto relevante para esta dissertação,

também em código-fonte, que adquiriram capacidades generativas e de compreensão contextual notáveis [18, 4].

Ao contrário dos modelos anteriores de DL em que é necessário algum tipo de calibração ou ajustes específicos para cada tarefa, os LLMs modernos não necessitam de tanta informação ou exemplos para conseguir devolver uma resposta. Isto significa que podem realizar novas tarefas com base apenas em instruções de linguagem natural, sem necessidade de treino adicional extensivo [3]. Esta característica é fundamental para a sua aplicação prática em ambientes empresariais, onde a diversidade de tarefas e a necessidade de rápida adaptação são elevadas.

Dois aspetos tornam os LLMs particularmente adequados para a análise de código:

1. **Treino Multimodal (Texto e Código):** Modelos como o Codex (base do GitHub Copilot), o GPT-4, ou modelos *open-source* como o CodeLLaMA, foram treinados em vastos *corpora* que incluem tanto linguagem natural como código-fonte de múltiplas linguagens de programação. Isto permitiu-lhes aprender não só a sintaxe, mas também padrões semânticos, *idioms* comuns e até associações entre comentários (especificações em linguagem natural) e a sua implementação [4, 17].

2. **Compreensão de Contexto de Longo Alcance:** A arquitetura *Transformer*, através do seu mecanismo de *attention*, permite que o modelo pondere relações entre *tokens* distantes no texto de entrada. Isto é crucial para compreender um Diff de código, onde uma alteração numa linha pode ter implicações em funções ou módulos referenciados noutras partes do ficheiro [18].

Estas capacidades transformam os LLMs de meros geradores de texto em ferramentas potencialmente capazes de atuar como *assistentes de programação inteligentes*, compreendendo a intenção por trás de alterações de código e avaliando-as num contexto mais amplo do que as ferramentas de análise estática baseadas em regras.

2.5 Modelos de Linguagem de Grande Escala Aplicados à Revisão de Código

A aplicação de LLMs à automação da revisão de código é uma área de investigação e desenvolvimento industrial em rápido crescimento. Estes modelos são utilizados para automatizar ou auxiliar várias sub-tarefas do processo de CR:

- **Geração de Descrições e Sumários:** Automatizar a criação de descrições concisas para PRs, resumindo as alterações realizadas, o que ajuda os revisores a compreender rapidamente o contexto [9].

- **Deteção de Defeitos e *Code Smells*:** Identificar potenciais *bugs*, vulnerabilidades de segurança, más práticas de desempenho ou violações de princípios de design (como

SOLID) com base no contexto aprendido, superando em alguns casos a precisão de analisadores estáticos tradicionais para certos tipos de problemas [10, 13].

- **Geração de Comentários de Revisão:** Esta é a aplicação mais diretamente relacionada com o objetivo desta dissertação. Os LLMs analisam o Diff e geram comentários que podem variar desde sugestões de estilo (formatação, nomenclatura) até a questões de lógica, sugerindo implementações alternativas ou apontando casos extremos não tratados [20, 14].
- **Resposta a Comentários:** Alguns sistemas exploram a capacidade dos LLMs de atuar como autores, respondendo automaticamente a comentários dos revisores, esclarecendo decisões ou comprometendo-se com correções [9].

Estudos e Resultados Chave: Trabalhos recentes demonstram o potencial desta abordagem. Por exemplo, estudos que utilizam modelos como o GPT-3.5/4 ou o CodeLLaMA mostraram que os comentários gerados podem ser considerados úteis por programadores numa percentagem significativa dos casos, por vezes equivalentes a comentários de revisores humanos para categorias específicas de problemas [10, 13]. No entanto, a literatura também aponta desafios consistentes: a utilidade é altamente variável, há uma tendência para gerar falsos positivos ou sugestões genéricas, e os modelos podem "alucinar", sugerindo problemas inexistentes ou correções incorretas [10, 21].

A eficácia destes sistemas depende criticamente da formulação do *prompt* (a instrução dada ao modelo), do contexto fornecido (o Diff, mas também potencialmente ficheiros relevantes, descrição do PR, etc.) e da existência de um mecanismo para guiar o modelo com o conhecimento específico da organização - uma lacuna que esta dissertação pretende colmatar.

2.6 Limitações atuais e Desafios

Apesar do potencial demonstrado, a integração prática de LLMs no processo de CR em ambientes empresariais específicos, como o da Processware, enfrenta vários desafios significativos:

1. **Falta de Contexto Organizacional Específico:** LLMs genéricos são treinados em código público e conhecimentos gerais. Não possuem conhecimento inerente sobre as **boas práticas internas**, convenções de *codebase*, padrões de arquitetura específicos ou regras de negócio de uma empresa. Um comentário genericamente correto pode ser irrelevante ou mesmo contrário aos padrões internos da organização [14, 17].
2. **Variabilidade e Inconsistência na Qualidade:** A utilidade dos comentários gerados pode flutuar amplamente. O modelo pode produzir uma análise perspicaz num caso e, no seguinte, gerar comentários vagos, incorretos ou focados em aspetos triviais, minando a confiança dos programadores [10, 13].

3. **Alucinações e Falsos Positivos:** LLMs podem identificar "problemas" que não existem, sugerir correções sintaticamente inválidas ou semanticamente erradas, ou atribuir incorretamente a causa de um defeito. Esta falta de fiabilidade exige uma supervisão humana cuidadosa, podendo anular os ganhos de eficiência [21, 12].

4. **Custo e Latência:** A execução de LLMs de grande porte para analisar cada PR pode ter custos computacionais e financeiros não triviais, além de introduzir latência no processo, especialmente para Diffs grandes ou análises complexas.

5. **Integração no Fluxo de Trabalho (*Workflow*):** Incorporar uma ferramenta baseada em LLMs de forma não intrusiva e que realmente agregue valor - em vez de gerar ruído - é um desafio de engenharia. A ferramenta deve complementar, e não substituir ou atrapalhar, o julgamento humano e a dinâmica colaborativa da revisão [19].

Este conjunto de limitações define claramente o espaço para a contribuição proposta nesta dissertação. O problema central não é *se* os LLMs podem gerar comentários, mas **como fazê-lo de forma consistente, alinhada com o contexto específico da Processware, e suficientemente fiável para ser integrada no fluxo de desenvolvimento da empresa**. A solução proposta, ao focar-se na personalização com base nas boas práticas internas e na integração fluida, aborda diretamente as lacunas 1 e 5, enquanto os objetivos de avaliação (Objetivo 5) visam mitigar os riscos apontados nas lacunas 2, 3 e 4.

3

Tecnologias

Este capítulo tem como objetivo apresentar as principais tecnologias relevantes para o desenvolvimento da solução proposta nesta dissertação. São abordadas as plataformas de controlo de versões e revisão de código, as ferramentas de revisão automática existentes e as abordagens baseadas em LLMs, bem como as tecnologias utilizadas para a extração, processamento e integração de alterações de código em sistemas reais.

3.1 Plataformas de Controlo de Versões e Revisão de Código

As plataformas de controlo de versões desempenham um papel central no desenvolvimento moderno de software, fornecendo mecanismos para gerir alterações de código, promover a colaboração entre programadores e suportar processos de revisão antes da integração de novas funcionalidades no produto em desenvolvimento. Atualmente, o sistema de controlo de versões distribuído *Git* é amplamente adotado na indústria e serve de base às principais plataformas de desenvolvimento colaborativo, suportando os fluxos de CR característicos do desenvolvimento moderno [15, 2].

Com base no *Git*, têm vindo a surgir plataformas que integram funcionalidades adicionais orientadas para a colaboração e revisão de código, como é o caso do *GitHub*, do *GitLab* e do *Bitbucket*. Estas plataformas introduziram o conceito de PRs (ou *merge request*), que permite submeter um conjunto de alterações para análise, discussão e validação antes da integração das alterações no ramo principal do repositório. Este modelo tornou-se um elemento central dos fluxos de desenvolvimento contemporâneos, particularmente em equipas distribuídas e ambientes empresariais de grande escala, estando associado a melhorias na qualidade do código e na deteção precoce de defeitos [11].

O processo de CR nestas plataformas é suportado por um conjunto de funcionalidades específicas, incluindo comentários *inline* associados a linhas ou blocos de código, mecanismos de aprovação ou rejeição de alterações e integração com sistemas automáticos de validação, como testes e analisadores estáticos. Estas capacidades permitem que os revisores forneçam feedback contextualizado e registem decisões de forma estruturada, promovendo a qualidade e rastreabilidade do processo de desenvolvimento.

Para além da interface de utilizador, estas plataformas disponibilizam interfaces de programação (Application Programming Interface (API)) e mecanismos de notificação, como *webhooks*, que permitem a integração de ferramentas externas no fluxo de CR. Através destas interfaces, é possível aceder programaticamente às alterações submetidas, obter o Diff associado a um PR e adicionar comentários de forma automática. Estas características tornam as plataformas de controlo de versões pontos naturais de integração para sistemas de apoio à CR baseados em inteligência artificial.

Neste contexto, a escolha de uma plataforma de controlo de versões não influencia apenas a gestão do código-fonte, mas também condiciona as possibilidades de automação e integração de soluções inteligentes no processo de CR. Assim, compreender as capacidades oferecidas por estas plataformas é essencial para o desenvolvimento de sistemas que visem apoiar ou automatizar a revisão de código de forma eficaz e alinhada com os fluxos de trabalho existentes.

3.2 Ferramentas de Revisão Automática Tradicionais

Com o aumento da complexidade dos sistemas de software e do volume de alterações submetidas para revisão, surgiram ferramentas de apoio automático ao processo de CR, com o objetivo de reduzir o esforço manual e melhorar a deteção precoce de problemas. Estas ferramentas baseiam-se, maioritariamente, em técnicas de análise estática de código, analisando o código-fonte sem necessidade de execução e recorrendo a um conjunto de regras ou heurísticas pré-definidas, com origem em abordagens clássicas de inspeção de software [5].

Entre as ferramentas mais utilizadas encontram-se os *linters* e analisadores estáticos, como o SonarQube, o ESLint, o PMD ou o Checkstyle, que são capazes de identificar violações de estilo, más práticas recorrentes, potenciais erros de execução e problemas de manutenibilidade. Estas ferramentas são frequentemente integradas em pipelines de CI, permitindo que os problemas detetados sejam sinalizados automaticamente durante o processo de desenvolvimento.

Uma das principais vantagens destas abordagens reside no seu comportamento determinístico e na capacidade de fornecer feedback rápido e consistente. As regras utilizadas são bem definidas, o que facilita a compreensão dos avisos gerados e a sua repetibilidade ao longo do tempo. Além disso, a integração destas ferramentas nos fluxos de desenvolvimento é, em geral, simples e bem suportada pelas plataformas de controlo de versões e CI.

No entanto, as ferramentas de revisão automática tradicionais apresentam limitações relevantes, especialmente em ambientes empresariais complexos. Estudos empíricos demonstram que a utilização de regras fixas dificulta a adaptação a contextos organizacionais específicos e pode originar um número elevado de avisos irrelevantes, levando frequentemente à sua desvalorização por parte dos programadores [7, 8].

Outra limitação significativa prende-se com a compreensão reduzida do contexto em que o código é desenvolvido. As ferramentas de análise estática não conseguem, em geral, captar a intenção do programador, avaliar decisões de design ou interpretar o impacto de uma alteração no contexto mais amplo do sistema. Assim, embora sejam eficazes na deteção de problemas bem definidos e repetitivos, revelam-se menos adequadas para fornecer comentários contextualizados e acionáveis durante o processo de CR.

Estas limitações motivaram a exploração de abordagens mais flexíveis e contextuais, capazes de compreender o código de forma mais abrangente e de gerar feedback alinhado com o contexto específico de cada projeto. As ferramentas baseadas em LLMs, discutidas na secção seguinte, surgem como uma resposta a estas necessidades.

3.3 Ferramentas baseadas em LLMs para Revisão de Código

3.3.1 Enquadramento Geral

Os LLMs baseiam-se em arquiteturas de atenção que permitem capturar dependências complexas em sequências de texto, tendo demonstrado resultados relevantes em múltiplas tarefas de engenharia de software [18, 3, 4]. Estes modelos exploram regularidades estatísticas presentes no código-fonte, frequentemente descritas como a sua *naturalidade*, permitindo capturar padrões semânticos e estruturais relevantes [6, 1].

No contexto de CR, a utilização de LLMs permite ultrapassar algumas das limitações das abordagens tradicionais baseadas em regras fixas. Estes modelos são capazes de analisar alterações de código de forma mais flexível, considerando o contexto envolvente, a intenção do programador e a estrutura global do código. Como resultado, torna-se possível gerar comentários mais expressivos, próximos do feedback fornecido por revisores humanos, e apoiar tarefas como a deteção de erros lógicos, a identificação de problemas de legibilidade e a sugestão de melhorias de design.

A aplicação de LLMs à revisão de código tem sido explorada tanto em ambientes de investigação como em soluções comerciais, refletindo o interesse crescente em integrar estas tecnologias nos fluxos de desenvolvimento existentes [17]. No entanto, a diversidade de abordagens adotadas evidencia que o domínio se encontra ainda em evolução.

3.3.2 Assistentes de CR Comerciais

Nos últimos anos, várias soluções comerciais têm incorporado LLMs como forma de apoiar tarefas relacionadas com a revisão de código. Estas ferramentas são, em geral, integradas em plataformas de controlo de versões ou disponibilizadas como serviços externos, sendo capazes de analisar alterações submetidas em PRs e gerar comentários automáticos.

Estudos recentes indicam que estas ferramentas podem reduzir o esforço manual dos revisores e melhorar a eficiência do processo de CR, embora a qualidade e relevância dos comentários gerados varie significativamente entre soluções [20, 13]. Em contextos

industriais, observa-se que a falta de adaptação ao contexto organizacional e às práticas específicas de cada projeto constitui uma limitação relevante [14, 19].

3.3.3 Ferramentas Integradas em IDEs

Para além das soluções integradas em plataformas de controlo de versões, os LLMs têm sido amplamente explorados em ferramentas integradas em IDEs. Estas ferramentas fornecem feedback em tempo real durante o processo de escrita de código, incluindo sugestões de melhoria, explicações de código e deteção de potenciais problemas.

Apesar do impacto positivo na produtividade individual, estas soluções não substituem o processo colaborativo de revisão formal, uma vez que o seu foco se encontra predominantemente no apoio ao programador individual e não nos fluxos estruturados de validação coletiva [10].

3.3.4 Limitações das Soluções Existentes

Apesar do potencial demonstrado pelas ferramentas baseadas em LLMs, estudos empíricos evidenciam limitações relacionadas com a consistência do feedback, a previsibilidade do comportamento dos modelos e a sua integração nos fluxos de desenvolvimento existentes [13, 19]. Adicionalmente, questões de privacidade, custos computacionais e controlo do comportamento dos modelos continuam a constituir desafios relevantes em ambientes empresariais.

3.4 Tecnologias para Extração e Processamento de Alterações de Código

A análise automática de alterações de código requer o acesso estruturado às modificações submetidas pelos programadores. Este acesso é geralmente realizado através das interfaces disponibilizadas pelas plataformas de controlo de versões, que permitem obter informação detalhada sobre *commits*, ficheiros modificados e diferenças associadas a PRs.

As alterações são frequentemente representadas sob a forma de Diffs, embora possam ser transformadas em representações sintáticas, como ASTs, ou híbridas, dependendo dos objetivos da análise. A escolha da representação influencia diretamente a eficácia das técnicas aplicadas, conforme discutido na literatura sobre análise automática de código [1, 16].

3.5 Tecnologias para Integração de LLMs

A integração de LLMs em sistemas de apoio à revisão de código é normalmente realizada através de arquiteturas baseadas em serviços intermediários, responsáveis pela orquestração da extração de dados, processamento das alterações e geração de feedback automático.

Estas arquiteturas permitem maior flexibilidade e controlo sobre o comportamento do sistema [9, 19].

No entanto, esta integração levanta desafios técnicos relacionados com latência, custos computacionais e robustez do sistema, exigindo mecanismos adicionais de controlo e monitorização [10].

3.6 Síntese e Considerações Finais

Neste capítulo foram apresentadas as principais tecnologias relevantes para o desenvolvimento de sistemas de apoio à revisão de código. Foram analisadas as plataformas de controlo de versões, as ferramentas tradicionais de revisão automática e as abordagens baseadas em LLMs.

Adicionalmente, foram discutidas as tecnologias necessárias para a extração, processamento e integração de alterações de código em sistemas reais. Este enquadramento tecnológico permitiu identificar limitações nas soluções existentes e fundamentar as decisões técnicas subjacentes à solução proposta, cuja arquitetura e implementação são apresentadas no capítulo seguinte.

4

Solução Proposta

Bibliografia

[1] M. Allamanis et al. «A Survey of Machine Learning for Big Code and Naturalness». Em: *ACM Computing Surveys* 51.4 (2018), pp. 1–37 (ver pp. 7, 8, 14, 15).

[2] A. Bacchelli e C. Bird. «Expectations, Outcomes, and Challenges of Modern Code Review». Em: *Proceedings of the 35th International Conference on Software Engineering (ICSE)* (2013), pp. 712–721 (ver pp. 6, 7, 12).

[3] T. B. Brown et al. «Language Models Are Few-Shot Learners». Em: *Advances in Neural Information Processing Systems (NeurIPS)* (2020) (ver pp. 9, 14).

[4] X. Chen et al. «Large Language Models for Software Engineering: A Survey». Em: *arXiv* (2023). URL: https://arxiv.org/abs/2312.15223 (ver pp. 1, 3, 9, 14).

[5] M. E. Fagan. «Design and Code Inspections to Reduce Errors in Program Development». Em: *IBM Systems Journal* 15.3 (1976), pp. 182–211 (ver pp. 6, 13).

[6] A. Hindle et al. «On the Naturalness of Software». Em: *Proceedings of the 34th International Conference on Software Engineering (ICSE)* (2012), pp. 837–847 (ver pp. 8, 14).

[7] B. Johnson et al. «Why Don't Software Developers Use Static Analysis Tools to Find Bugs?» Em: *Proceedings of the 35th International Conference on Software Engineering (ICSE)* (2013), pp. 672–681 (ver pp. 7, 13).

[8] S. Kim e M. D. Ernst. «Which Warnings Should I Fix First?» Em: *Proceedings of the 6th Joint Meeting on Foundations of Software Engineering (FSE)* (2016), pp. 45–55 (ver pp. 7, 8, 13).

[9] J. Li et al. «CodeAgent: A Multi-Agent Framework for Automated Code Review». Em: *arXiv* (2024). URL: https://arxiv.org/abs/2402.02172 (ver pp. 9, 10, 16).

[10] D. Liang et al. «Code Review Automation: Strengths and Weaknesses». Em: *arXiv* (2024). URL: https://arxiv.org/abs/2401.05136 (ver pp. 1–3, 7, 10, 15, 16).

[11] S. McIntosh et al. «An Empirical Study of the Impact of Modern Code Review Practices on Software Quality». Em: *Empirical Software Engineering* 19.6 (2014), pp. 1819–1850 (ver pp. 6, 7, 12).

[12] T. Nguyen et al. «A Comparison of LLM-Assisted Development Tools». Em: *arXiv* (2025). URL: https://arxiv.org/abs/2501.00000 (ver pp. 1, 11).

[13] F. Rahman et al. «An Empirical Study of LLM-Based Automated Code Review». Em: *arXiv* (2024). URL: https://arxiv.org/abs/2412.18531 (ver pp. 1–3, 10, 14, 15).

[14] G. Research. «AI-Assisted Assessment of Coding Practices in Industrial Code Review». Em: *Google Publications* (2024). URL: https://research.google/pubs/ai-assisted-assessment-of-coding-practices-in-industrial-code-review/ (ver pp. 3, 10, 15).

[15] P. C. Rigby e C. Bird. «Convergent Contemporary Software Peer Review Practices». Em: *Proceedings of the 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)* (2013), pp. 202–212 (ver pp. 6, 12).

[16] M. Tufano et al. «Deep Learning Similarities from Different Representations of Source Code». Em: *Proceedings of the 26th IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)* (2019) (ver pp. 8, 15).

[17] M. Tufano et al. «Large Language Models in Software Engineering: A Systematic Literature Review». Em: *ACM TOSEM* (2024). URL: https://nzjohng.github.io/publications/papers/tosem2024_5.pdf (ver pp. 1–3, 9, 10, 14).

[18] A. Vaswani et al. «Attention Is All You Need». Em: *Advances in Neural Information Processing Systems (NeurIPS)* (2017) (ver pp. 9, 14).

[19] Y. Xie et al. «Rethinking Code Review Workflows with LLM Assistance». Em: *arXiv* (2025). URL: https://arxiv.org/html/2505.16339v1 (ver pp. 2, 3, 11, 15, 16).

[20] L. Zhang et al. «AI-powered Code Review with LLMs: Early Results». Em: *arXiv* (2024). URL: https://arxiv.org/abs/2404.18496 (ver pp. 3, 10, 14).

[21] K. Zhong et al. «Evaluating AI-Assisted Code Review in Industry». Em: *arXiv* (2024). URL: https://arxiv.org/abs/2508.18771 (ver pp. 1–3, 10, 11).

A

Appendix 1

B

Appendix 2

I

Anexo 1