Análise de Redes de Colaboração em Comunidades de Desenvolvimento de Software: Um Estudo Baseado em Dados do GitHub

Victor Debortoli Landim[1], **Hudson Silva Borges**[1]

[1]FACOM – Universidade Federal de Mato Grosso do Sul (UFMS)
Campo Grande – MS – Brasil

{victor.landim,hudson.borges}@ufms.br

Abstract. *With the continuous evolution of collaborative development platforms, new challenges arise in understanding how different forms of interaction shape the dynamics between developers and communities throughout the software development lifecycle. This study investigates the structure of collaboration within contemporary development communities, focusing on interactions recorded on GitHub. By collecting data through the GraphQL API and employing an ETL process to migrate information from MongoDB to Neo4j, a graph is constructed to integrate developers, artifacts, and contribution events. Network analysis algorithms—such as centrality measures, community detection, and structural metrics—reveal, for instance, that some activities are highly concentrated among a few contributors, that there are intermediaries between subgroups, and that dense communities form around Pull Requests. The results indicate that graph-based modeling offers an analytical perspective capable of uncovering technical and social patterns that characterize collaboration in distributed projects, thus contributing to broader investigations on organization and collective behavior within software development ecosystems.*

Resumo. *Com a evolução contínua das plataformas de desenvolvimento colaborativo, surge o desafio de compreender como diferentes formas de interação moldam a dinâmica entre desenvolvedores e comunidades ao longo do ciclo de desenvolvimento de software. Este trabalho investiga como se estrutura a colaboração em comunidades contemporâneas de desenvolvimento, com foco nas interações registradas no GitHub. A partir da coleta de dados via API GraphQL e de um processo de ETL que migra informações do MongoDB para o Neo4j, é construído um grafo que integra desenvolvedores, artefatos e eventos de contribuição. Os algoritmos de análise de redes — centralidades, detecção de comunidades e métricas estruturais — revelam, por exemplo, que algumas atividades estão fortemente concentradas em poucos colaboradores, que há intervenientes entre subgrupos e que existem comunidades densas em torno de Pull Requests. Os resultados indicam que a modelagem orientada a grafos fornece uma perspectiva analítica capaz de revelar padrões técnicos e sociais que caracterizam a colaboração em projetos distribuídos, contribuindo para investigações mais amplas sobre a organização e o comportamento coletivo em ecossistemas de desenvolvimento de software.*

1. Introdução

Sistemas de controle de versão passaram de ferramentas técnicas para ambientes colaborativos ricos, incluindo interações sociais, dinâmicas comunitárias e coordenação distribuída [Lima et al. 2014].

Com o suporte de plataformas como o GitHub, que integram versionamento, comunicação assíncrona e registro minucioso de atividades, formou-se um ecossistema que gera grandes quantidades de dados estruturados e semiestruturados em torno do desenvolvimento de software. A variedade crescente dessas atividades, como abertura de issues, code reviews, discussões, reações, acompanhamento de projetos, gerou um interesse maior em entender não só o código gerado, mas também as relações que sustentam a colaboração em grande escala [Lima et al. 2014];[Borges et al. 2019].

Nesse sentido, análises fundamentadas em grafos surgem como alternativas apropriadas para mapear e explorar a natureza relacional dos dados gerados por plataformas de desenvolvimento colaborativo, conforme demonstrado por [Lima et al. 2014] ao caracterizarem o Github como uma rede social complexa. Se considerarmos atores e entidades no contexto de um repositório do GitHub, torna-se possível representar de forma mais rigorosa a estrutura colaborativa observada no projeto.

Atores correspondem aos usuários que participam do repositório, realizando ações como, por exemplo, criar pull requests, submeter commits ou interagir de forma mais social, como comentando em issues ou reagindo a discussões. foram adicionados novas funcionalidades Entidades representam os artefatos centrais do fluxo de desenvolvimento, como pull requests, commits, issues e discussions, que constituem os objetos sobre os quais as ações dos atores incidem.

Essa distinção entre atores e entidades possibilita descrever, de maneira estruturada, a complexidade das interações nos repositórios e sustenta a construção de um grafo capaz de capturar relações técnicas e sociais de forma integrada.

Estudos da literatura já utilizaram modelagens baseadas em grafos para investigar dependências de projeto e interações entre usuários, como demonstrado nos trabalhos de [Fu et al. 2021] e [Thung et al. 2013]. Estes estudos se baseiam em conjuntos de dados antigos, usam APIs desatualizadas e se concentram em contextos que não refletem adequadamente a dinâmica contemporânea das comunidades de desenvolvimento, visto que nos últimos anos foram adicionadas novas funcionalidades ao Github, como reações e discussões, que ampliaram o escopo das interações sociais na plataforma.

Este trabalho busca superar os dados desatualizados ao utilizar dados contemporâneos e incorporar tipos de interações que passaram a compor o ecossistema do GitHub nos últimos anos. Esses dados, coletados por meio da GraphQL API, permitem a realização de consultas mais detalhadas e estruturadas, ampliando o escopo analítico em relação aos estudos anteriores.

Os repositórios possuem uma métrica quantificada de estrelas (*stars*), onde os desenvolvedores indicam seu interesse ou satisfação com o projeto. As estrelas de um repositório podem ser consideradas como um indicador para sua popularidade, se correlacionadas com outras métricas como quantidade de *forks*, linguagem de programação ou domínio de aplicação [Borges et al. 2016]. Dessa forma, foi realizado um estudo

empírico, considerando apenas o número de estrelas dos repositórios, foram escolhidos 5 repositórios, variando desde projetos com muitas estrelas a projetos com poucas estrelas, com o propósito de investigar a atividade em comunidades distintas.

Com uma organização e tratamento dos dados que utiliza grafos construídos no Neo4j, é possível visualizar essas ligações entre atores, eventos e artefatos de desenvolvimento descobrindo padrões estruturais que não são evidentes por métodos de análise mais tradicionais.

Este trabalho procura responder às questões: (i) De que maneira, como comunidades internas, elas se desenvolvem e se organizam em torno do fluxo de trabalho que se baseia em Pull Requests? (ii) De que forma se estruturam as interações entre desenvolvedores em diversos projetos? (iii) Quem são os principais protagonistas na colaboração técnica?

Para isso, é proposta uma pipeline de coleta, transformação e modelagem dos dados, que serve de base para a aplicação de clássicos algoritmos de análise de redes, como centralidade de grau, centralidade de intermediação (*betweenness centrality*) e detecção de comunidades com algoritmo de Louvain, com o intuito de investigar aspectos como centralidade, intermediação e agrupamentos naturais entre colaboradores.

Os achados indicam a presença de colaborações densas e heterogêneas, em que poucos atores concentram a maior parte das interações significativas, enquanto outros desempenham papéis estratégicos de ponte entre subgrupos.

A identificação de comunidades revela agrupamentos que se alinham bem às dinâmicas internas dos projetos. Além disso, a modelagem baseada em grafos também se provou eficiente em capturar de forma significativa a estrutura dos dados que compõem os repositórios: por meio da modelagem, foi possível integrar diferentes tipos de entidades (usuários, pull requests, commits, issues, reações, discussões) e os múltiplos relacionamentos entre eles (criação, revisão, comentário, reação, etc.), mantendo a precisão semântica de cada ação.

O restante deste trabalho está organizado da seguinte forma. A Seção 2 apresenta o referencial teórico que fundamenta a pesquisa. A Seção 3 apresenta os trabalhos relacionados ao tema. A Seção 4 descreve a metodologia adotada, incluindo coleta, tratamento e modelagem dos dados. A Seção 5 detalha o desenvolvimento da ferramenta, componentes que compõem sua implementação e a construção da modelagem do grafo. A Seção 6 discute os resultados obtidos a partir das análises realizadas. Por fim, a Seção 7 reúne as conclusões e aponta possibilidades para pesquisas futuras.

2. Referencial Teórico

Esta seção descreve as bases conceituais relacionadas às plataformas de desenvolvimento colaborativo, às técnicas de mineração de repositórios, aos modelos de representação por grafos e às ferramentas que possibilitam a análise estrutural dos dados. O objetivo é fornecer o suporte teórico necessário para contextualizar as decisões metodológicas adotadas e esclarecer os princípios que orientam as etapas de coleta, modelagem e interpretação dos resultados.

2.1. Github e suas APIs para coleta de dados

O GitHub é atualmente uma das principais plataformas de hospedagem, versionamento e colaboração em projetos de código-fonte. Os recursos do GitHub abrangem mecanismos colaborativos que permitem que milhares de usuários interajam, revisem e aprimorem projetos colaborativamente [Lima et al. 2014]. É uma plataforma digital amplamente utilizada para o desenvolvimento colaborativo de software, que combina funcionalidades de hospedagem de código com recursos de interação social, formando um ambiente que é tanto técnico quanto comunitário [Dabbish et al. 2012];[Borges et al. 2016].

Segundo a documentação oficial, o GitHub funciona como uma plataforma em nuvem que integra os mecanismos de versionamento do Git a um ambiente colaborativo estruturado, permitindo que desenvolvedores armazenem, editem e sincronizem arquivos de código-fonte por meio de repositórios remotos [GitHub 2025c].

De modo geral, o GitHub funciona como um repositório online em que colaboradores, seja individualmente ou em equipe, podem armazenar, compartilhar e desenvolver projetos de maneira cooperativa. Cada projeto é mantido em um repositório, que pode conter código-fonte, documentação e outros arquivos necessários para um sistema. O GitHub também possui rastreabilidade das modificações no repositório e registra todas as interações com os conteúdos armazenados, permitindo que seja possível identificar quem realizou as alterações, quando foram alteradas e por quê.

No entanto, o GitHub não se limita às funcionalidades de um sistema tradicional de controle de versões. A plataforma permite que desenvolvedores interajam de maneira social, possibilitando que sigam outros perfis (*follow*) e fóruns de discussão integrados no próprio repositório. Também possibilita que usuários favoritem (*star*) projetos que consideram interessantes, acompanhem repositórios (*watch*), discutam problemas ou melhorias por meio de Issues e contribuam ativamente com alterações por meio de pull-requests [Borges e Valente 2022].

O GitHub disponibiliza mecanismos formais para a coleta de dados públicos por pesquisadores e profissionais, permitindo o acesso estruturado a informações sobre repositórios, contribuições, interações e histórico de desenvolvimento. Essa extração é realizada por meio de APIs (*Application Programming Interfaces*) públicas. Elas são oferecidas em dois modelos: a REST API, voltada para consultas diretas baseadas em endpoints, e a GraphQL API, que possibilita consultas mais flexíveis e orientadas a esquemas. Isso favorece a obtenção eficiente de grandes volumes de dados analisáveis.

A REST API adota o modelo de serviços web baseados em HTTP e recursos RESTful. Com esse método, é possível realizar requisições para diversos endpoints que representam entidades como usuários, repositórios, commits, issues, pull requests, etc. Cada requisição segue um formato padrão (método HTTP, URL, token de acesso e parâmetros) e devolve uma resposta no formato JSON [GitHub 2025b].

Por outro lado, como definido na documentação oficial da GitHub GraphQL API, a GraphQL API representa uma abordagem mais recente e flexível, permitindo ao usuário definir exatamente quais campos deseja obter em cada requisição, evitando o excesso de dados e multiplicidade de chamadas. Conforme a documentação oficial do GitHub, ambas APIs retornam dados adicionais se o usuário estiver autenticado, além de aumentar o limit rate para as requisições [GitHub 2025a].

As APIs possuem um sistema para garantir disponibilidade, estabilidade e o uso adequado, conforme definido na documentação. Porém, esse comportamento limita a extração de grandes quantidades de informações, dificultando a aquisição dos dados para o estudo. Dessa forma, torna-se evidente a necessidade de recorrer a abordagens de mineração de repositórios para garantir uma coleta abrangente e consistente de informações.

2.2. Mineração de Repositórios (MSR)

A Mineração de Repositórios de Software (MSR, do inglês *Mining Sofware Repositories*) refere-se a um campo de estudo que investiga os grandes volumes de dados gerados no processo de desenvolvimento de software. O objetivo é extrair conhecimento, padrões e evidências que auxiliem a compreensão e a melhoria dos processos, produtos e equipes envolvidas no desenvolvimento de software [Barros et al. 2021]. Essa abordagem parte do princípio de que os repositórios de softwares contêm uma grande quantidade de informações valiosas sobre o ciclo de vida dos projetos, as práticas dos desenvolvedores e a evolução do código.

Segundo estudos recentes, como o de [Souza et al. 2020] e [Gesse Junior 2023], a MSR tem se mostrado eficaz na produção de insights sobre a qualidade de software, manutenção de sistemas legados e comportamento coletivo em ambientes de desenvolvimento distribuído. Ademais, a área tem crescido em relevância acadêmica, com conferências dedicadas como a International Conference on Mining Software Repositories[1], que reúne pesquisadores e profissionais para discutir avanços metodológicos e aplicações práticas.

2.3. Grafos

Grafos são estruturas matemáticas fundamentais para representar relações entre entidades [David e Jon 2010]; sua aplicação em contextos sociais e computacionais tem se mostrado extremamente eficaz para a análise de interações complexas, como as que ocorrem em comunidades de desenvolvimento de software [Thung et al. 2013][Fu et al. 2021].

Um grafo é composto por um conjunto de vértices (ou nós) e de arestas (ou conexões) que ligam pares de vértices. A Figura 1 ilustra um exemplo de relação entre usuários que reagiram e comentaram em Issues, em que os círculos representam visualmente os vértices e as arestas representam as conexões.

Na modelagem desse projeto, os nós representam as entidades (como desenvolvedores, repositórios, eventos), enquanto as arestas simbolizam interações entre os nós. O processo de modelagem pertinente a este trabalho será detalhado na Seção 5.

[1] https://www.msrconf.org/

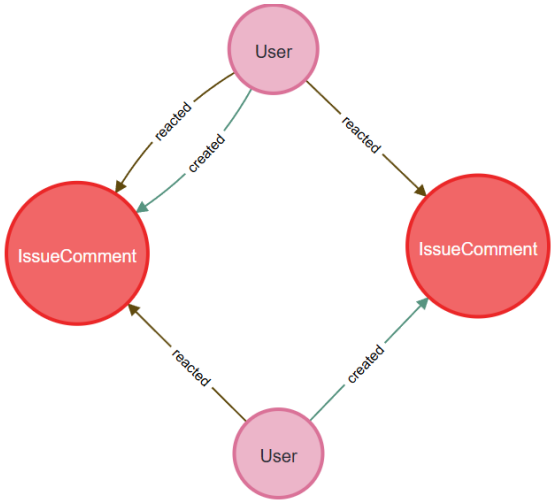

Figura 1. Representação de nodos e relações
Fonte: De autoria própria

A escolha pelo uso de grafos decorre do fato de que os dados extraídos por meio das APIs do GitHub são interconectados [Gousios 2013], característica que possibilita representar os dados a partir de um modelo orientado a grafo. Os grafos permitem representar de forma explícita e eficiente as conexões entre as entidades, possibilitando a aplicação de algoritmos clássicos de análise de redes. Métricas como grau de centralidade, intermediação (*betweeness*), proximidade e detecção de comunidades são particularmente úteis para identificar nós influentes, agrupamentos naturais de colaboração e padrões de comportamento coletivo [Laranjeira e Cavique 2018];[Fu et al. 2021].

Embora o uso de grafos para modelagem de interações no GitHub já tenha sido discutido em pesquisas anteriores como no trabalho de [Thung et al. 2013] e [Fu et al. 2021], as quais constroem grafos de interação a partir de eventos do GitHub, analisando propriedades estruturais, comunidades e usuários com papel central na difusão de informações. O presente trabalho adota uma perspectiva distinta ao considerar um conjunto mais amplo e heterogêneo de entidades e relações, abrangendo não apenas interações diretas entre usuários, mas também eventos associados a pull requests.

Essa abordagem metodológica visa entender melhor a dinâmica colaborativa que se manifesta nesses processos entre desenvolvedores e como isso contribui para o estabelecimento de padrões sociais e técnicos nos repositórios que estão sendo estudados. Os estudos relacionados que abordam essas características são detalhados na Seção 3.

2.4. Ferramentas relevantes

Nesta subseção serão apresentadas as ferramentas utilizadas para viabilizar o desenvolvimento deste trabalho, abrangendo desde a extração e manipulação dos dados até a construção do grafo e a aplicação das análises realizadas. O objetivo é descrever, de forma objetiva, os instrumentos técnicos que possibilitaram a coleta estruturada das informações, o processamento adequado dos registros e a execução dos métodos analíticos empregados na investigação.

2.4.1. Bancos de dados de grafos

Bancos de dados de grafos (GDB) são sistemas de gerenciamento de dados que utilizam a teoria dos grafos para representar entidades e seus relacionamentos. Diferentemente dos bancos relacionais, que estruturam em tabelas, os bancos de grafos armazenam informações como uma rede de nós e arestas [Robinson et al. 2015].

Existe atualmente uma variedade de ferramentas que implementam o modelo de grafos, cada uma oferecendo diferentes arquiteturas, linguagens de consulta e capacidades de processamento.

Diante desse cenário, foi conduzido um levantamento das caracteristicas de alguns GDBs para identificar qual ferramenta é adequada para atender às necessidades específicas deste trabalho, considerando principalmente a possibilidade de executar algoritmos clássicos de análise de grafos e criação de elementos com propriedades.

Neo4j - Segundo a documentação oficial, Neo4j[2] é um banco de dados de grafos bastante utilizado atualmente, garantindo ampla documentação e apoio da comunidade. Sua modelagem *Labeled Property Graph Model* (LPG Model) é uma das principais características da ferramenta, permitindo a criação de nós e arestas com propriedades. Além disso, esse GDB utiliza Cypher, uma linguagem descritiva empregada para realizar consultas em banco de dados de grafos de propriedade, possibilitando que usuários desenvolvam buscas complexas em dados conectados de maneira intuitiva e visual, focando nos padrões dos nós e relações de um grafo. O Neo4j disponibiliza também uma interface gráfica acessível por meio do Neo4j Browser, executado localmente junto ao servidor. Essa interface permite a realização de consultas em Cypher, a visualização interativa do grafo e a inspeção dos nós, relações e propriedades armazenados no banco de dados, servindo como ferramenta de apoio para verificação, exploração e validação dos dados carregados. Para viabilizar essas análises avançadas em um contexto de pesquisa ou desenvolvimento, a plataforma integra bibliotecas especializadas que expandem significativamente suas capacidades analíticas.

Como informado na documentação oficial do Neo4j na seção de introdução ao *Graph Data Science* [3] (GDS), é uma biblioteca fundamental para a aplicação de algoritmos de grafos (como os de centralidade, detecção de comunidades e busca de caminhos). O que permite a extração de insights complexos sobre a estrutura e o comportamento da rede de dados.

Adicionalmente, o framework *Awesome Procedures On Cypher*[4] (APOC) oferece um vasto leque de procedimentos e funções customizadas que estendem a linguagem de consulta Cypher. Isso proporciona aos pesquisadores a flexibilidade necessária para realizar tarefas que vão desde a importação/exportação de dados até a execução de transformações e invocações de APIs externas, agilizando a fase de pré-processamento e análise de dados em grande escala.

[2]https://neo4j.com/

[3]https://neo4j.com/docs/getting-started/gds/#gds-get-started

[4]https://neo4j.com/labs/apoc/

JanusGraph - De acordo com a documentação oficial, JanusGraph[5] é um banco de dados de grafos open-source, projetado para crescimento horizontal com grandes volumes de dados. Permite armazenar e consultar grafos com centenas de bilhões de nós e arestas, distribuídos por clusters de máquinas. Também é compatível com a Apache TinkerPop[6]/Gremlin[7] para linguagem de consulta de grafos [JanusGraph 2025].

MemGraph - Conforme especificado na documentação oficial, o MemGraph[8] é um banco de dados de grafos open-source, com arquitetura in-memory-first e também persistente, focado em cargas de trabalho com alta taxa de ingestão, latência baixa e análises em tempo real. Suporta o modelo de grafos de propriedade e utiliza a linguagem Cypher para realizar consultas nos grafos [MemGraph 2025].

A partir do levantamento realizado, a ferramenta selecionada foi o Neo4j, principalmente devido à sua configuração simplificada e ao ecossistema sólido que a acompanha. A plataforma conta com uma comunidade extensa e ativa, o que favorece a resolução de problemas e o acesso a boas práticas de modelagem. Além disso, o modelo baseado em propriedades oferecido pelo Neo4j permite a inclusão direta de atributos em nós e arestas, característica essencial para análises que exigem granularidade informacional. Somam-se a isso bibliotecas dedicadas a algoritmos clássicos de grafos, como o Neo4j Graph Data Science (GDS), que amplia significativamente o potencial analítico da solução e reforça sua adequação às demandas do estudo.

2.4.2. Github Proxy Server

As APIs REST e GraphQL do GitHub impõem limites de aproximadamente 5.000 requisições por hora por token e utilizam mecanismos de detecção de abuso que restringem requisições paralelas, o que pode resultar em bloqueios temporários durante a extração de dados em larga escala [Borges e Valente 2022]. Por causa dessas restrições, a coleta dos dados necessários para este estudo demandou a utilização do Github Proxy Server[9]. O uso dessa ferramenta mostrou-se essencial para viabilizar a coleta de dados exigida pelo escopo do trabalho em extrair todas as informações de diversos repositórios, assegurando consistência e confiabilidade na extração dos registros analisados.

2.4.3. Extração, Transformação e Carga

O processo de ETL (Extração, Transformação e Carga) é uma etapa fundamental em projetos que envolvem movimentação ou reestruturação de dados entre diferentes sistemas. Consiste em extrair informações de uma fonte original, tratá-las para que assumam um formato consistente e adequado ao novo ambiente e, por fim, carregá-las para o destino final. Embora seja uma técnica tradicional em engenharia de dados, seu princípio é sim-

[5]https://janusgraph.org/
[6]https://tinkerpop.apache.org/
[7]https://tinkerpop.apache.org/gremlin.html
[8]https://memgraph.com/
[9]https://github.com/gittrends-app/github-proxy-server

ples: preparar os dados para que possam ser reutilizados em outro contexto sem perda de integridade ou significado.

No contexto deste trabalho, o ETL foi utilizado para viabilizar a migração de informações de um banco de dados orientado a documentos (MongoDB)[10] para um banco orientado a grafos (Neo4j). Como as duas estruturas adotam modelos conceituais distintos, foi necessário aplicar etapas de transformação capazes de reinterpretar documentos, coleções e relações implícitas em forma de elementos conectados. Esse procedimento garantiu que os dados mantivessem coerência no novo modelo, preservando vínculos essenciais e favorecendo a análise baseada em conexões.

O processo foi implementado por meio de scripts em Python[11] desenvolvidos especificamente para esta finalidade. Esses scripts automatizaram a leitura dos dados na fonte, aplicaram as regras de conversão adequadas e realizaram a inserção estruturada no Neo4j. Dessa maneira, o ETL cumpriu o papel de ferramenta intermediária, permitindo que informações originalmente não estruturadas, fossem reorganizadas de forma clara e consistente para o ambiente orientado a grafos utilizado no estudo.

3. Trabalhos Relacionados

A investigação sobre colaboração em plataformas de desenvolvimento, especialmente no GitHub, tem sido abordada sob diferentes perspectivas metodológicas e níveis de granularidade. Estudos clássicos analisam o fenômeno da colaboração social no desenvolvimento de software em larga escala, explorando estruturas globais de conexão entre desenvolvedores e projetos. No atual estudo, essas abordagens são revisadas de modo a contextualizar a evolução das técnicas de modelagem e análise de redes aplicadas ao desenvolvimento colaborativo, permitindo situar nossa proposta frente aos avanços e limitações identificados na literatura.

A pesquisa *Understanding the user interactions on GitHub: a social network perspective* de [Fu et al. 2021] trata do problema de como as interações de usuários na plataforma GitHub se organizam em nível de rede social por meio de um grafo de interações entre usuários e repositórios no GitHub, especialmente sob a ótica de ”furos estruturais”(*structural holes*) em redes sociais.

O estudo de [Fu et al. 2021] propõe construir um grafo de interação a partir de três tipos principais de eventos (*Watch, IssueComment, PullRequest*) para representar interações entre usuários e repositórios durante o ano de 2019. Nesta rede, aplicaram métricas de centralidade e detectaram “*spanners*” de buracos estruturais — atores que conectam comunidades diferentes — usando teoria de buracos estruturais (*structural hole theory*) de Burt [Burt 1992]. Os resultados indicaram que a rede é muito esparsa, com pouca reciprocidade, muitos usuários com grau baixo, e que os atores que conectam comunidades diferentes têm grau e betweenness significativamente maiores, ou seja, desempenham papel de ligação entre comunidades mais que os usuários “comuns”.

Comparando com o presente estudo, destacam-se diferenças e convergências. Em termos de convergência, ambos utilizam modelagem em grafos para capturar interações em GitHub, aplicam métricas de rede como centralidade e detecção de comunidades e

[10]https://www.mongodb.com/
[11]https://www.python.org/

visam compreender cooperação e estrutura social técnica em projetos de software. No entanto, o presente estudo difere em vários aspectos, tais como:

1. O escopo temporal é mais amplo, abrangendo dados históricos completos de cinco repositórios ao invés de um único ano;
2. A modelagem é mais detalhada, incluindo múltiplos tipos de nós (atores, *commits*, *pull requests, issues*, discussões) e relações que capturam diversas formas de interação colaborativa;
3. O presente estudo introduz uma relação agregada específica (COLLAB_PR) para analisar colaboração em pull requests com pesos diferenciados, permitindo uma análise mais refinada do envolvimento dos desenvolvedores.

Por fim, o presente estudo foca na análise de subgrafos específicos para entender dinâmicas particulares de colaboração, enquanto [Fu et al. 2021] adotam uma visão mais geral da rede social. No estudo *Network Structure of Social Coding in GitHub*, desenvolvido por [Thung et al. 2013], os autores buscam compreender como desenvolvedores e repositórios se conectam dentro desse ecossistema, de modo a produzir evidências que auxiliem o aprimoramento de ferramentas voltadas à eficiência do trabalho colaborativo e ao estímulo de novas interações entre participantes.

Para atingir esse objetivo, [Thung et al. 2013] coletaram dados referentes a aproximadamente 100 mil projetos e 30 mil desenvolvedores. A partir desse conjunto, construíram dois modelos de rede. O primeiro, denominado rede de projetos, representa cada repositório como um nó, conectando-os quando compartilham desenvolvedores em comum. O segundo, a rede de desenvolvedores, utiliza programadores como nós, conectando-os quando colaboram em projetos iguais.

Os resultados demonstram que a rede de projetos apresenta distribuição do tipo lei de potência, indicando forte assimetria na popularidade dos repositórios. Em contraste, a rede de desenvolvedores não segue o mesmo padrão, uma vez que a concentração de contribuições em projetos altamente colaborativos distorce a distribuição.

Ambos os trabalhos investigam o GitHub por meio de análise de redes, mas enquanto [Thung et al. 2013] oferecem uma visão macro do ecossistema ao conectar desenvolvedores e projetos em larga escala, o estudo atual adota uma perspectiva aprofundada em colaboração em *pull requests*, modelando interações detalhadas dentro de repositórios específicos para revelar dinâmicas internas de colaboração.

4. Metodologia

A metodologia adotada é definida como um estudo empírico, cujas etapas estão representadas na Figura 2. A principal proposta consiste em analisar um conjunto de projetos previamente estabelecido, a fim de compreender e aprofundar o conhecimento sobre suas característica. Para isso, são estabelecidos e selecionados repositórios que apresentam variados níveis de popularidade; em seguida, procede-se com as etapas de coleta dos dados, modelagem do grafo, transformação das informações e análise dos resultados.

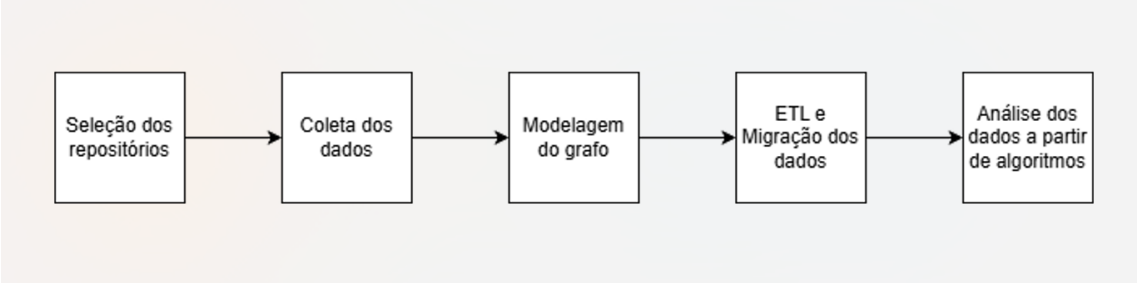

Figura 2. Diagrama do fluxo da pesquisa
Fonte: De autoria própria

Para este trabalho, o critério de variedade buscou contemplar tanto cenários com elevado tráfego de contribuições quanto ambientes com menos dados. Os projetos selecionados são apresentados na Subseção 5.2.

Fez-se necessária a coleta dos dados a partir da API GraphQL, utilizando a ferramenta Github Proxy Server com o objetivo de mitigar as limitações de rate-limit e detecção de abuso. A extração dos dados foi realizada em formato JSON bruto, preservando a estrutura original fornecida pela API, e armazenada em um banco de dados NoSQL MongoDB. A escolha pelo MongoDB deveu-se à sua flexibilidade para lidar com documentos JSON, facilitando a manipulação e transformação dos dados posteriormente.

Após a coleta dos dados, estes foram analisados com o objetivo de criar uma modelagem no contexto de grafos. Esse processo envolveu a identificação das principais entidades (nós) e suas interações (arestas), definindo propriedades relevantes para cada elemento do grafo. Esses atributos representam informações intrínsecas aos elementos do grafo, tais como nome e login de desenvolvedores, data de criação de *pull requests*, descrição de *issues*, entre outros metadados disponibilizados pela plataforma. O intuito dessa análise foi garantir que a estrutura do grafo refletisse adequadamente as dinâmicas colaborativas presentes nos repositórios, permitindo a aplicação de métodos tradicionais de análise de grafos.

Neste trabalho, os dados foram migrados para o Neo4j (versão 2025.08.0, edição: Community). Essa escolha técnica fundamentou-se na natureza intrinsecamente relacional dos dados de colaboração em projetos de software. Diferentemente de bancos de dados relacionais tradicionais, bancos de dados orientados a grafos representam diretamente os nós e relações, facilitando consultas complexas sobre conexões entre entidades [Chang et al. 2022]. De acordo com Fernandes e Bernardinho (2018, apud [Chang et al. 2022]), o Neo4j é um sistema transacional que mapeia dados em nós e arestas em vez de tabelas, sendo um dos bancos de grafos mais empregados globalmente em diversas áreas. Por essas características, o Neo4j foi utilizado como GDB para esse estudo.

Para modelar o domínio no Neo4j, são definidos os elementos principais do grafo com base na colaboração em projetos Github. Dessa forma, elementos como:

- Atores - são os usuários que interagem dentro do repositório, podendo realizar diversas ações, como a criação de issues, pull-requests, comentários, reações, entre

outras;

- Repositórios - Representam os projetos que estão hospedados na plataforma;
- Commits - Registros de alterações no código-fonte;
- *Pull-requests* - Solicitações de integração de mudanças propostas por colaboradores;
- *Issues* - Tópicos para discussão, relato de *bugs* ou sugestões de melhorias;
- *Discussions* - Fóruns de discussão para debates mais amplos em uma determinada comunidade;
- *DiscussionComments* - Comentários feitos dentro das discussões;
- *TimelineItem* - Este elemento possui uma modelagem distinta dos outros nodos, porque representa diversos eventos que ocorrem dentro do repositório, como comentários de issue, fechamento de discussão, entre outros. Possui relações entre outros nodos, como Actor, Issue e Pull-request;
- *Release* - Versões oficiais do software disponibilizadas no repositório.

Esses elementos foram categorizados como tipos distintos de nós no Neo4j, permitindo a diferenciação clara entre os diversos elementos envolvidos na dinâmica colaborativa dos repositórios.

Todas as ações realizadas dentro de um repositório do GitHub são representadas como conexões no grafo, permitindo descrever de forma clara as interações entre seus componentes. Uma ação é definida como uma ligação direcionada entre dois elementos: o primeiro indica quem realizou a ação, e o segundo indica o que foi afetado por ela, como apresentado na Figura 3, que ilustra a criação de dois pull requests por um único usuário.

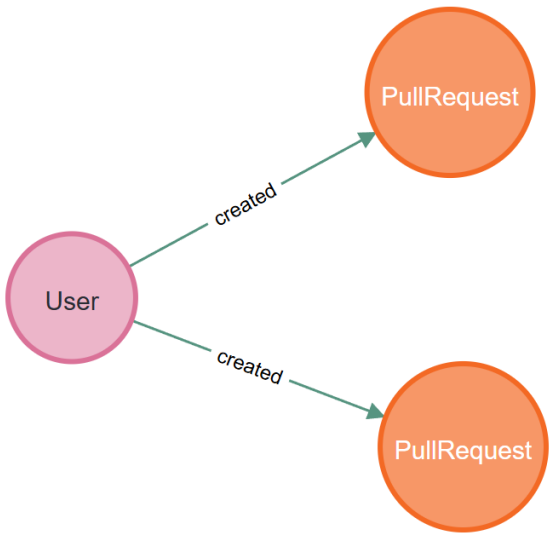

Figura 3. Exemplo de relacionamento entre usuário e Pull Request
Fonte: De autoria própria

Além da criação de pull requests, diversos outros tipos de relacionamentos foram modelados para capturar a variedade de interações presentes no GitHub. Ações como abrir issues, enviar commits, participar de discussões, revisar contribuições ou reagir a comentários foram representadas como ligações direcionadas entre o ator e a entidade correspondente.

Alguns elementos, como estrelas (*stars*) e observadores (*watchers*), não foram representados como nós independentes, mas sim como relacionamentos, pois caracterizam interações simples. Essa distinção permite preservar a estrutura do grafo sem introduzir nós redundantes e garante que a modelagem reflita adequadamente o papel de cada tipo de interação.

O processo de migração proposto foi implementado por meio de scripts personalizados em Python. Para isso, foram utilizadas bibliotecas como o pymongo[12] (para acesso ao MongoDB) e o driver oficial do Neo4j para Python (que possibilita criar nós e relacionamentos programaticamente). O processo de ETL consistiu em extrair cada documento JSON do MongoDB, converter os campos em propriedades dos nós e relacionamentos conforme o modelo de grafo definido e inserir esses elementos no Neo4j por meio de lote de dados.

A inserção dos itens no Neo4j foi inicialmente realizada por meio de uma abordagem individual, utilizando o driver oficial. Nessa abordagem, cada objeto era criado por meio de uma única query em Cypher, inserindo-se um item por vez, o que se mostrou excessivamente lento dada a volumetria dos dados. Diante dessa limitação, adotou-se uma solução de inserção em lote ("batch"), explorando o comando UNWIND do Cypher, que permite receber uma lista de objetos como parâmetro e inseri-los em uma única operação.

Neste trabalho foram adotadas técnicas consolidadas de análise de grafos com o propósito de investigar a estrutura e a dinâmica dos cinco repositórios selecionados, como centralidade de grau, intermediação e detecção de comunidades com algoritmo de Louvain. A escolha dessa abordagem baseia-se na natureza da base de dados, caracterizada por um grafo direcionado com diversas tipologias de nós e múltiplos tipos de relacionamentos, e no objetivo de capturar tanto os aspectos estruturais (como centralidade, conectividade e componentes) quanto os aspectos sociais (como colaboração e influência) dos atores envolvidos nos repositórios.

A análise de redes sociais (SNA) provê um arcabouço teórico e metodológico adequado à investigação de sistemas com muitas entidades interligadas, sendo amplamente aplicada em contextos de engenharia de software. Um levantamento sistemático identificou que "a SNA em projetos de desenvolvimento de software oferece métodos granulares para análise de redes sociais de times, indo além das ferramentas tradicionais de gestão de projetos" [Schreiber e Zylka 2020].

Adicionalmente, estudos sobre sistemas de software como redes complexas apontam que métricas de grafos tradicionais (grau, entrelaçamento, centralidade, modularidade) são úteis para caracterizar a topologia de sistemas de software, suas dependências ou interações [Li et al. 2020]. Com base nessas referências, foram aplicados os seguintes algoritmos clássicos de análise de grafos:

- Centralidade de grau, utilizada para quantificar o nível de atividade de cada ator no grafo de interações. Desenvolvedores com alto grau de entrada ou saída indicam perfis com maior engajamento no grafo, neste contexto representando usuários que realizam ações frequentes;
- Centralidade de intermediação (*betweenness*), empregada para identificar atores que atuam como pontes entre subgrupos ou entre diferentes áreas de interação,

[12]https://pypi.org/project/pymongo/

sendo esses usuários cruciais para a coesão e fluxo de informações dentro do repositório. Usuários com alto nível de intermediação podem influenciar significativamente a dinâmica colaborativa;

- Detecção de comunidades (algoritmo de Louvain), utilizada para identificar agrupamentos naturais de atores que colaboram mais intensamente entre si, revelando subgrupos dentro do repositório que podem representar equipes de trabalho, áreas de interesse ou clusters específicos.

5. Desenvolvimento da ferramenta

Esta seção descreve, com viés técnico, o processo de construção da infraestrutura e das rotinas implementadas para suportar a coleta, transformação, carga e análise dos dados extraídos dos cinco repositórios GitHub. Apresentam-se a arquitetura adotada, decisões de modelagem, bibliotecas e ferramentas empregadas e os principais desafios técnicos enfrentados.

5.1. Visão geral da arquitetura

A arquitetura geral do sistema é estruturada em três componentes principais que operam de forma encadeada:

- Coleta: GitHub Proxy Server, responsável por orquestrar requisições às APIs do GitHub e mitigar limites de rate-limit;
- ETL / Migração: *scritps* em Python (pymongo + driver oficial Neo4j) que realizam limpeza, transformação e carga em lote no Neo4j;
- Análise e pós-processamento: como as bibliotecas APOC para procedimentos utilitários e Graph Data Science (GDS) para execução de algoritmos de rede.

O primeiro componente é o GitHub Proxy Server, responsável por interagir com as APIs do GitHub e controlar o ritmo das requisições, funcionando como ponto de entrada dos dados. Esse componente fornece as informações coletadas diretamente para a etapa seguinte.

O segundo componente corresponde ao processo de ETL, implementado por *scripts* em Python que utilizam pymongo e o driver oficial do Neo4j. Ele recebe os dados brutos oriundos do *proxy*, realiza limpeza e transformação conforme o modelo de grafo definido e executa a carga em lote no Neo4j. Dessa forma, atua como ponte entre a coleta e o armazenamento estruturado.

O terceiro componente integra o ambiente de análise, no próprio Neo4j, utilizando bibliotecas como APOC e Graph Data Science (GDS). Após a carga dos dados, essa camada aplica procedimentos utilitários e algoritmos de rede, produzindo métricas e insights sobre o grafo.

A Figura 4 ilustra a arquitetura geral do sistema, destacando os fluxos de dados entre os componentes e as principais tecnologias empregadas em cada etapa.

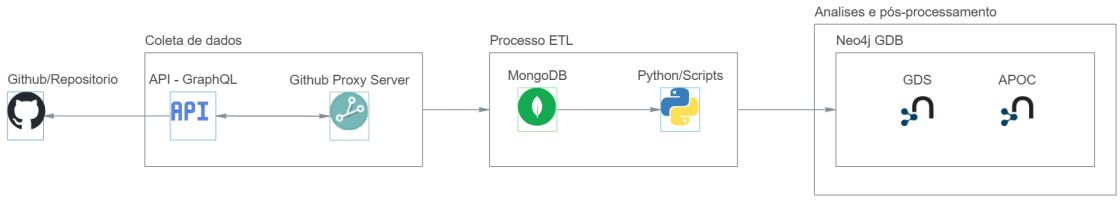

Figura 4. Arquitetura geral do sistema
Fonte: De autoria própria

5.2. Seleção e coleta dos dados

A seleção dos projetos analisados foi conduzida de forma intencional, visando contemplar diferentes níveis de visibilidade, maturidade e dinâmica colaborativa no GitHub. O objetivo foi incluir repositórios que permitissem observar padrões diversos de interação entre usuários e artefatos do desenvolvimento. Para isso, foram escolhidos projetos de grande porte e ampla adoção, exemplos de porte intermediário com comunidades ativas e repositórios de menor escala relativa. O conjunto final de repositórios selecionados inclui:

- React[13] - Projeto hospedado no repositório do Facebook no GitHub, conta atualmente com cerca de 241.000 estrelas, refletindo sua grande popularidade e adoção na comunidade de desenvolvimento. A linguagem principal utilizada no projeto é o JavaScript, compondo aproximadamente 66,7%;
- Filament[14] - Repositório reúne 27,7 mil estrelas, o que evidencia sua elevada popularidade na comunidade de desenvolvimento open-source. A linguagem predominante do projeto é PHP, representando cerca de 86% do código-fonte;
- fastapi_mcp[15] - Repositório mantido pela organização tadata-org no GitHub, apresenta aproximadamente 11,1 mil estrelas, o que indica um nível considerável de interesse da comunidade de desenvolvimento no projeto. A linguagem principal utilizada no código-fonte é o Python, representando 100% da base de código listada no repositório;
- hyprdots[16] - Projeto mantido por Prasanth Rangan no GitHub, apresenta aproximadamente 8,6 mil estrelas. A linguagem predominante do projeto é Shell, representando cerca de 90,6% do código-fonte;
- cmake_template[17] - Repositório hospedado sob a organização cpp-best-practices no GitHub, acumula aproximadamente 1,5 mil estrelas. A linguagem principal do projeto é CMake, representando cerca de 76,5% do código-fonte.

Após a seleção dos repositórios, foi realizada a coleta dos dados públicos disponíveis no GitHub. Para viabilizar a extração em escala e contornar limitações operacionais impostas pela API, empregou-se o GitHub Proxy Server, conforme proposto por

[13]https://github.com/facebook/react
[14]https://github.com/filamentphp/filament
[15]https://github.com/tadata-org/fastapi_mcp
[16]https://github.com/prasanthrangan/hyprdots
[17]https://github.com/topics/cmake-template

[Borges e Valente 2022]. Essa ferramenta atua como camada intermediária para orquestrar requisições, gerenciar múltiplos *tokens* de acesso, distribuir carga entre *workers* e mitigar mecanismos de *abuse detection*, assegurando a obtenção eficiente das informações necessárias à modelagem do grafo.

5.3. ETL e migração para Neo4j

A migração dos dados para o Neo4j foi implementada em Python, utilizando as bibliotecas pymongo e o driver oficial da plataforma. O processo foi estruturado em dois módulos principais. O primeiro módulo é responsável pela criação dos nós, definidos a partir da modelagem conceitual do domínio. Nessa etapa, determina-se previamente quais coleções do MongoDB serão convertidas em tipos de nós relevantes. Como exemplo, a coleção *Actor* é transformada em nós do tipo *Actor*, preservando os atributos presentes nos documentos originais.

A Figura 5 apresenta as propriedades de um nó ator no Neo4j, em que algumas dessas propriedades são de origem direta da coleta pela API do GitHub, como a `bio`, que representa a biografia descrita pelo usuário em seu perfil, `typename`, que indica o tipo do ator (User, bot ou Organization). Por outro lado, existem propriedades que foram criadas durante o processo de ETL, como as propriedades de `elementID` e `ID`, resultantes do processo de inserção dos dados no GDB. Também é possível observar propriedades, como `collab_pr_degree` e `community_louvain`, que foram geradas a partir da execução de algoritmos de grafos. As propriedades resultantes dos algoritmos serão exploradas na seção 6,

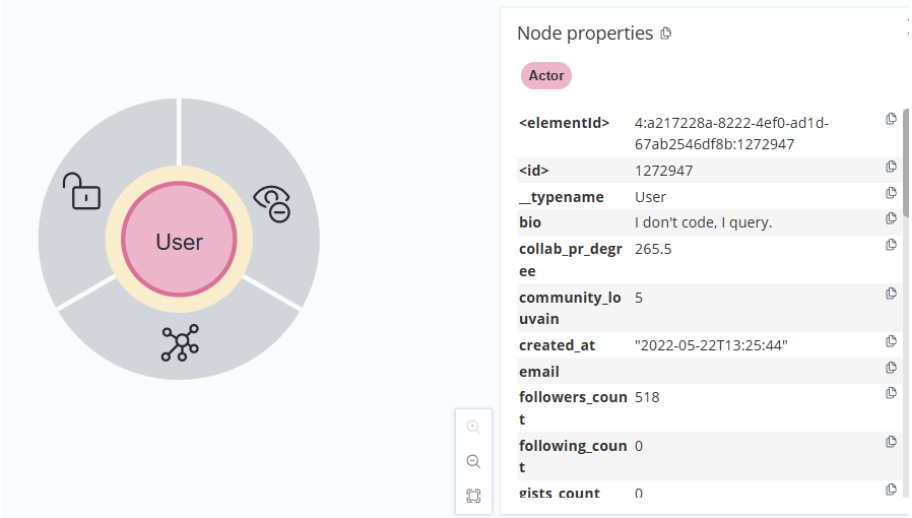

Figura 5. Propriedades de um nó ator no Neo4j
Fonte: De autoria própria

Após a criação dos nós, realiza-se a indexação, etapa fundamental para garantir o desempenho das operações no Neo4j. Estudos como o de [Mpinda et al. 2015] demonstram que a presença de índices reduz significativamente o número de nós examinados em consultas, acelerando o tempo de resposta. Para isso, foi implementado um método que cria índices por meio de comandos nativos do Cypher, utilizando o atributo *Id* de cada tipo de nó.

O segundo módulo trata da criação das relações, igualmente definida pelo modelo de dados. Cada relação é criada com os atributos necessários, como a data de criação no caso de relações associadas a issues. Na figura 6, é a representação visual de uma relação entre uma Issue e um usuário, e também apresenta a propriedade `createdAt`, referenciando a data de criação da Issue da relação created.

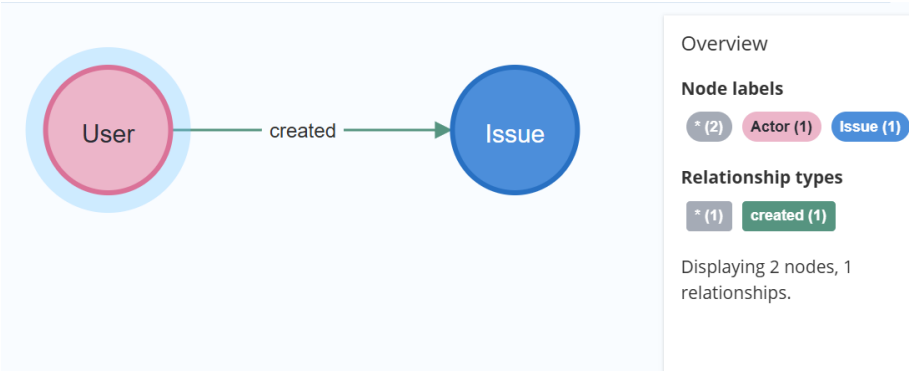

Figura 6. Atributo de uma relação created no Neo4j
Fonte: De autoria própria

Durante o processo, verificou-se que a inserção individual de relações gerava significativa degradação de desempenho, especialmente devido ao grande volume de dados. Para mitigar esse problema, adotou-se a técnica de inserção em lote por meio do comando UNWIND do Cypher, que permite enviar conjuntos de registros em uma única transação.

A partir da validação dos dados no Neo4j Browser, por meio de consultas específicas, foi possível confirmar a eficiência da migração a partir do gráfico resultante. As verificações incluíram a comparação direta entre elementos. dos documentos presentes no MongoDB e os nós e relações correspondentes no Neo4j após a migração. Essa etapa permitiu assegurar que os atributos essenciais foram preservados, que os rótulos e índices foram configurados corretamente e que as relações refletiam fielmente a modelagem definida.

A Figura 7 representa o schema final dos dados migrados para o Neo4j, evidenciando os principais tipos de nós e suas arestas. O modelo ilustra como as entidades do domínio (atores, repositórios, commits, pull requests, issues, discussões e timeline items) estão relacionadas entre si, refletindo a dinâmica colaborativa dos repositórios analisados.

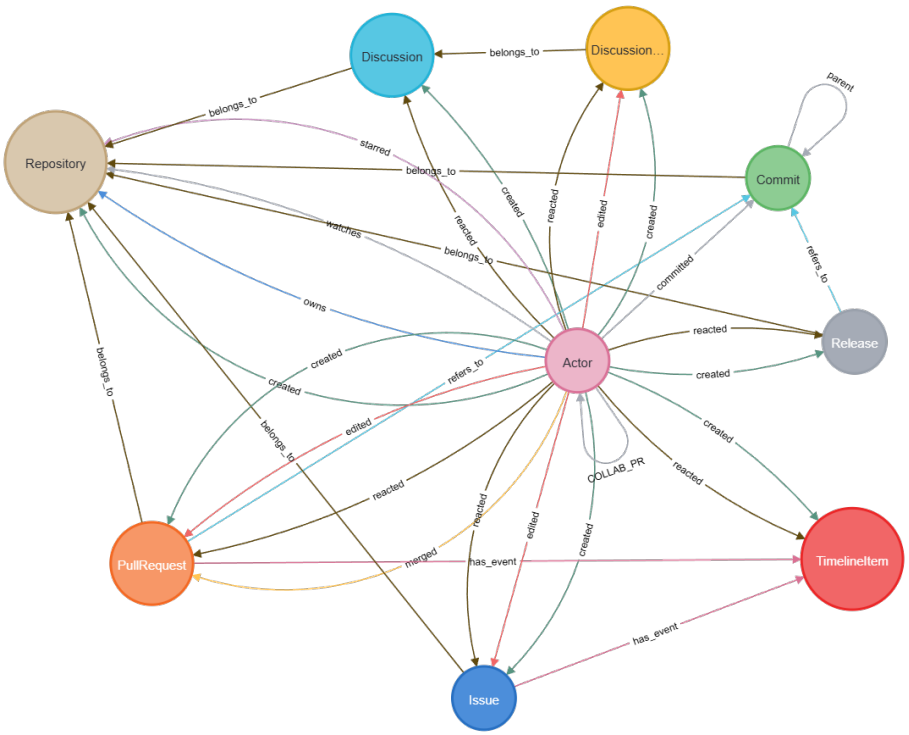

Figura 7. Modelo Padrão do schema dos dados
Fonte: De autoria própria

5.4. GDS e APOC

A ferramenta desenvolvida integra os recursos oferecidos pelo APOC (*Awesome Procedures On Cypher*) e pela *Graph Data Science Library* (GDS), ambos componentes nativos e disponibilizados pelo Neo4j. Essas bibliotecas ampliam significativamente as capacidades do banco de dados, permitindo tanto operações avançadas de pré-processamento quanto a execução de algoritmos de análise estrutural em larga escala.

Um exemplo representativo do uso do APOC na ferramenta é o uso do procedimento `apoc.periodic.iterate`, que permite aplicar operações de forma incremental, evitando sobrecarga de memória em grandes conjuntos de dados. Esse procedimento se fez necessário para criação de relações que não foram modeladas para o gráfico final, mas especificamente criadas para análises posteriores, como será detalhado na subseção 6.2.

A GDS constitui o núcleo analítico da ferramenta, sendo responsável pela execução dos principais algoritmos de análise de grafos utilizados neste trabalho. A biblioteca foi empregada para aplicar algoritmos clássicos de métricas estruturais, possibilitando a avaliação quantitativa dos padrões de colaboração entre os atores. Além disso, o GDS permite a projeção de subgrafos em memória, os quais podem ser configurados conforme o modelo conceitual definido e utilizados como base para a aplicação dos algoritmos.

6. Resultados

Nesta seção apresentam-se os resultados obtidos a partir do processamento e análise dos dados extraídos dos cinco repositórios selecionados. Inicialmente caracteriza-se o dataset

em termos de escopo, volume e composição (número de nós por tipo, número de arestas por relação, distribuição por repositório e proporção de eventos por categoria). Em seguida, descrevem-se os modelos de grafos criados no Neo4j — o grafo original (resultado da migração dos dados) e o grafo projetado a partir dos dados originais para análises específicas.

6.1. Caracterização do dataset

A caracterização do dataset evidencia a dimensão e a heterogeneidade estrutural dos dados extraídos dos cinco repositórios analisados. Conforme a Tabela 1, foram identificados 1.272.947 nós, distribuídos entre diferentes tipos de entidades relacionadas à atividade colaborativa do GitHub. Observa-se predominância dos nós do tipo TimelineItem (796.232), que representam eventos de atividade e compõem a maior parte do registro histórico dos repositórios. Em seguida, destacam-se os nós do tipo Actor (353.028), correspondendo aos usuários que realizaram ações observadas na plataforma. Os demais tipos apresentam volumes menores, porém relevantes para a modelagem: Commit (51.157), PullRequest (24.933), DiscussionComment (22.292), Issue (15.885), Discussion (8.559), Release (856) e Repository (5).

Rótulo do nó	Quantidade
TimelineItem	796.232
Actor	353.028
Commit	51.157
PullRequest	24.933
DiscussionComment	22.292
Issue	15.885
Discussion	8.559
Release	856
Repository	5

Tabela 1. Quantidade de nós por rótulo

As relações, quantificadas na Tabela 2, totalizam 2.372.611 arestas, refletindo a intensa conectividade entre os elementos. As relações has_event (816.131) e created (776.991) são as mais frequentes, evidenciando o grande volume de eventos vinculados a issues, pull requests e discussões, bem como a ampla participação de atores criando entidades no sistema. Destacam-se também as relações starred (284.946) e reacted (214.851), que refletem o componente social da plataforma. Relações estruturais como belongs_to (98.749), parent (61.834) e committed (50.786) descrevem o fluxo técnico de desenvolvimento. Interações como merged (17.591), refers_to (15.552), edited (11.358) e watches (6.956) complementam o conjunto. A relação owns apresenta apenas 5 ocorrências, coerente com o fato de existirem apenas cinco repositórios no conjunto analisado.

Tabela 2. Distribuição das relações por tipo, nó de origem e nó de destino

Relação	Nó de Origem	Nó de Destino	Quantidade
created	Actor	TimelineItem	704.788
has_event	PullRequest	TimelineItem	610.213
starred	Actor	Repository	284.946
has_event	Issue	TimelineItem	205.918
reacted	Actor	TimelineItem	131.823
parent	Commit	Commit	61.834
belongs_to	Commit	Repository	51.157
committed	Actor	Commit	50.786
reacted	Actor	Issue	40.296
created	Actor	PullRequest	24.855
reacted	Actor	PullRequest	22.663
belongs_to	DiscussionComment	Discussion	22.292
created	Actor	DiscussionComment	22.262
merged	Actor	PullRequest	17.591
belongs_to	Issue	Repository	15.885
created	Actor	Issue	15.694
refers_to	PullRequest	Commit	14.773
reacted	Actor	Release	11.559
belongs_to	Discussion	Repository	8.559
created	Actor	Discussion	8.531
watches	Actor	Repository	6.956
reacted	Actor	DiscussionComment	6.792
edited	Actor	PullRequest	5.523
edited	Actor	Issue	3.089
edited	Actor	DiscussionComment	2.746
reacted	Actor	Discussion	1.718
created	Actor	Release	856
belongs_to	Release	Repository	856
refers_to	Release	Commit	779
owns	Actor	Repository	5
created	Actor	Repository	5

6.2. Colaboração em Pull Requests

A relação *COLLAB_PR* foi criada para sintetizar interações relevantes que ocorrem em torno de Pull Requests, utilizando pesos diferenciados para refletir a importância das ações desempenhadas pelos desenvolvedores. Enquanto a criação de um PR indica autoria e iniciativa técnica, ações como merge e, principalmente, revisão (review) representam etapas decisivas no fluxo de integração e manutenção do código.

Atribuir pesos maiores a revisões e merges e pesos menores a reações permite diferenciar participação superficial de interação efetiva no processo de colaboração. Dessa forma, *COLLAB_PR* atua como uma relação agregada que expressa a intensidade de interação entre pares de atores, acumulando ao longo do tempo o peso de suas contribuições compartilhadas. A confecção dessa relação foi realizada a partir da consulta

Cypher presente na Figura 8.

```
CALL apoc.periodic.iterate(
"
  MATCH (pr:PullRequest)
  WITH pr, COUNT{ (pr)←[:created|reacted|merged]-(:Actor) } AS p
  WHERE p ⩾ 2

  MATCH (pr)-[:has_event]→(t:TimelineItem {__typename:'PullRequestReview'})

  MATCH (pr)←[r1:created|merged|reacted]-(a1:Actor)
  MATCH (t)←[r2:created|reacted]-(a2:Actor)
  WHERE id(a1) < id(a2)

  WITH a1, a2,
       CASE type(r1)
           WHEN 'created' THEN 1
           WHEN 'merged'  THEN 2
           WHEN 'reacted' THEN 0.5
       END AS w1,
       CASE type(r2)
           WHEN 'created' THEN 2.0
           WHEN 'reacted' THEN 0.5
       END AS w2

  RETURN a1, a2, (w1 + w2) AS weight
",
"
  MERGE (a1)-[r:COLLAB_PR]-(a2)
  ON CREATE SET r.weight = weight
  ON MATCH SET r.weight = r.weight + weight
",
{batchSize: 5000, parallel: false}
);
```

Figura 8. Consulta para criação da relação COLLAB-PR
Fonte: De autoria própria

O processo inicia buscando Pull Requests cujo nível de atividade seja suficientemente significativo, definido pela presença mínima de dois eventos associados a atores distintos. A partir desse conjunto filtrado, são recuperados os eventos de revisão correspondentes, representados como TimelineItems do tipo PullRequestReview.

Em seguida, a consulta percorre as interações existentes entre atores e Pull Requests, considerando ações de criação, reação e merge no próprio Pull Request, bem como ações de criação e reação dentro do respectivo processo de revisão. Para cada par de atores envolvidos, aplica-se uma regra ponderada que expressa a intensidade colaborativa de cada tipo de ação: atividades como criação ou merge do Pull Request são interpretadas como contribuições de maior relevância, enquanto reações recebem peso reduzido. No caso das revisões, atribuem-se pesos ainda mais elevados para a criação de uma revisão, refletindo sua complexidade e impacto no fluxo colaborativo.

Após o cálculo da soma ponderada que representa o grau de colaboração entre dois atores, o resultado é atribuído a um parâmetro da relação denominado ”weight”, que determina o peso de uma relação entre um nó e outro. O procedimento final realiza a inserção ou atualização de uma relação específica entre eles no grafo, denominada COLLAB_PR. Quando a relação ainda não foi estabelecida, ela é criada com o peso calculado; caso já exista, seu peso é acumulado, refletindo a intensidade colaborativa observada ao

longo de múltiplos Pull Requests.

O resultado sintetiza, de forma quantitativa, o nível de colaboração entre atores no contexto de desenvolvimento colaborativo, permitindo análises posteriores sobre padrões de interação, centralidade e comportamento coletivo nos repositórios analisados.

A construção de uma projeção específica para analisar a colaboração entre desenvolvedores mostrou-se necessária devido à natureza heterogênea e altamente conectada do grafo original. Esse grafo é composto por diversos tipos de nós e múltiplas relações de interação. Embora o grafo completo represente a totalidade do ecossistema de desenvolvimento, sua densidade e diversidade de relações introduzem ruído analítico. Operações como atividades automatizadas, criação de eventos auxiliares e interações superficiais tendem a inflar métricas de centralidade e distorcer a interpretação sobre colaboração real entre os participantes. Assim, tornou-se fundamental destacar um grafo de colaborações que efetivamente representa a participação técnica entre pessoas.

A Figura 9 apresenta a consulta Cypher utilizada para projetar um grafo de colaboração, evidenciando como os nós de atores e as relações de colaboração são selecionados e estruturados para a análise. Esta consulta define o subconjunto de dados utilizado para investigar as interações técnicas entre desenvolvedores, cujo resultado é a criação de um grafo em memória empregado em análises subsequentes de centralidade e padrões de colaboração.

```
CALL gds.graph.project(
    'GraphCollabPR',
    'Actor',
    {
        COLLAB_PR: {
            orientation: 'UNDIRECTED',
            properties: 'weight'
        }
    }
)
```

Figura 9. Consulta para projeção do grafo de colaboração
Fonte: De autoria própria

A Figura 10 mostra uma parcela do grafo projetado *GraphCollabPR*, resultante da query 9. Nesse grafo, os nós representam atores (usuários) do mesmo repositório, e as arestas ponderadas indicam o nível de colaboração entre eles, permitindo visualizar as conexões e suas intensidades na relação *COLLAB_PR*. Os valores presentes nas arestas refletem o peso acumulado das interações colaborativas, permitindo identificar os pares de desenvolvedores com maior intensidade de colaborações em Pull Requests. Um valor mais alto indica uma colaboração frequente ou significativa entre os dois atores conectados.

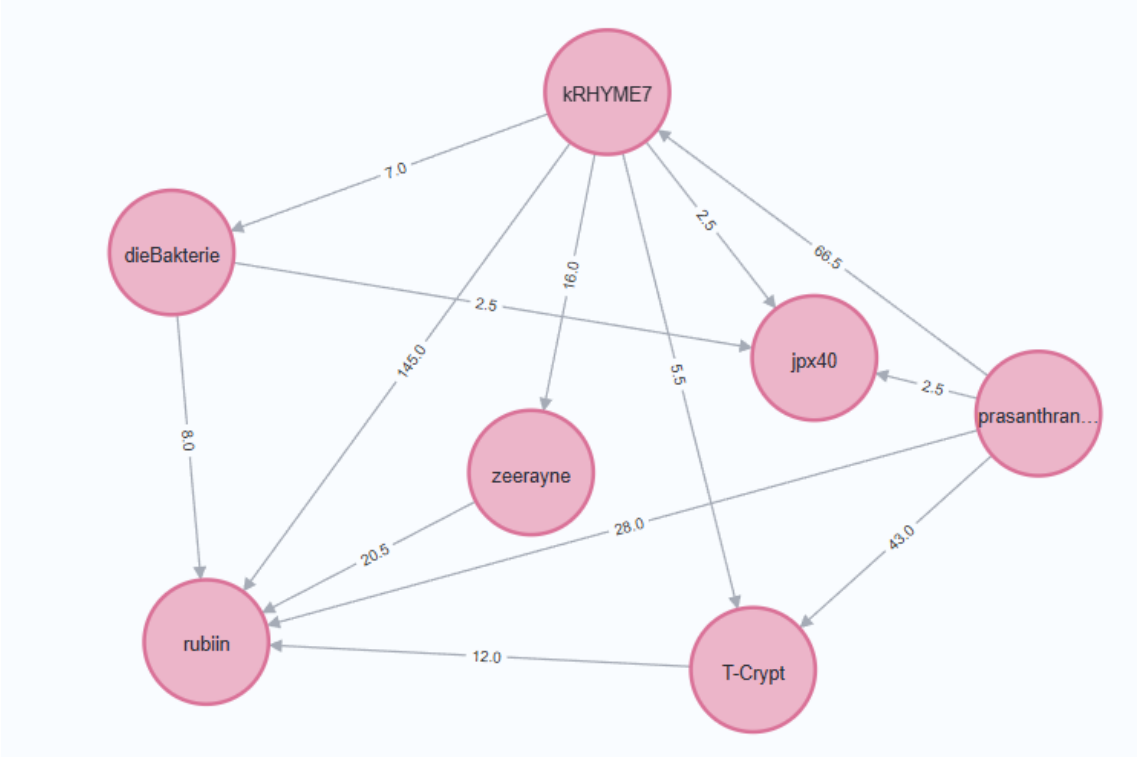

Figura 10. Grafo de colaboração COLLAB-PR no Neo4j Browser
Fonte: De autoria própria

6.2.1. Centralidade de Grau

O resultado do cálculo do algoritmo de centralidade, representado na Tabela 3, mostra a presença de um núcleo dominante de colaboradores: sebmarkbage apresenta o maior score (24.997), seguido por gaearon (23.819,5), acdlite (18.982) e bvaughn (16.612). Esses valores indicam que esses atores mantêm relações de colaboração recorrentes e de alta intensidade com diversos participantes, refletindo envolvimento contínuo tanto na autoria quanto na revisão e integração de código. Por agregar pesos às diferentes ações, a métrica prioriza pares que mantêm interações frequentes e de maior impacto, fornecendo um retrato objetivo dos participantes mais ativos no fluxo de integração de código.

Tabela 3. Top 5 atores por centralidade de grau ponderada no grafo COLLAB-PR

Ator	Score de Centralidade
sebmarkbage	24997.0
gaearon	23819.5
acdlite	18982.0
bvaughn	16612.0
rickhanlonii	9890.0

Contudo, é importante comparar esses resultados com outras medidas estruturais: a centralidade de intermediação (betweenness centrality) pode evidenciar atores que fun-

cionam como pontes entre subgrupos, enquanto a centralidade de autovetor (*eigenvector centrality*) destaca nós conectados a atores influentes. Além disso, repositórios de grande porte tendem a dominar a projeção e inflar scores de colaboradores muito ativos nesses projetos, o que pode enviesar interpretações quando se busca avaliar influência numa amostra heterogênea.

6.2.2. Centralidade de intermediação

A Tabela 4 apresenta os cinco atores com maior pontuação na métrica de centralidade de intermediação (*betweenness centrality*) no grafo COLLAB-PR. Essa métrica identifica atores que atuam como pontes entre diferentes subgrupos dentro do grafo de colaboração.

Tabela 4. Top 5 atores por centralidade de intermediação

Ator	Score de Centralidade de Intermediação
gaearon	789946,366788099
dudenamedjune	767512,7298553201
leonardodino	316514,5163682817
acdlite	292031,3173253745
danharrin	273987,0573055274

Ao executar o algoritmo de centralidade de intermediação sem considerar pesos, foi analisado apenas a estrutura das ligações entre os desenvolvedores, baseando-nos exclusivamente na presença das relações de colaboração formadas a partir dos *Pull Requests*. Assim, o algoritmo avalia como o grafo está organizado em termos de conexões e caminhos possíveis, sem que diferenças numéricas nos pesos das arestas influenciem a identificação dos atores que funcionam como pontos de ligação entre grupos distintos.

Na Tabela 4, os resultados revelam que “gaearon”, “dudenamedjune”, “leonardodino”, “acdlite” e “danharrin” ocupam posições centrais como pontos de passagem entre grupos de colaboradores. Essa métrica não identifica necessariamente os desenvolvedores com maior volume de interações, mas sim aqueles que conectam diferentes subgrupos dentro da comunidade, permitindo que informações, práticas e revisões circulem entre segmentos que, de outra forma, estariam mais isolados.

Esses resultados reforçam que a estrutura de colaboração observada não é homogênea ou totalmente descentralizada. A rede apresenta um pequeno conjunto de atores que desempenham papéis-chave na mediação entre grupos, contribuindo para a coesão do ecossistema e para o fluxo contínuo de trabalho. Essa interpretação complementa métricas como centralidade de grau, oferecendo uma visão mais profunda sobre como a comunidade se organiza e se comunica.

Os resultados dos dois algoritmos aplicados, centralidade de grau ponderada e centralidade de intermediação não ponderada, ao grafo COLLAB-PR revelam diferenças relevantes sobre o papel desempenhado pelos desenvolvedores na rede de colaboração. A centralidade de grau ponderada identifica os atores com maior volume e intensidade de interações diretas, considerando que cada aresta possui um peso proporcional à força da colaboração (autoria, merge e revisão de Pull Requests). Nesse cenário, sebmark-

bage, gaearon, acdlite e bvaughn aparecem entre os nomes mais centrais, o que indica participação direta e frequente no fluxo de trabalho do projeto. Esses atores mantêm um grande número de vínculos fortes, refletindo contribuições contínuas e recorrentes com múltiplos colaboradores.

Por outro lado, a centralidade de intermediação não ponderada revela um conjunto de atores cuja importância não decorre do volume de interações, mas da posição estrutural que ocupam na rede. Nesse caso, gaearon permanece como figura central, porém a lista passa a incluir desenvolvedores como dudenamedjune e leonardodino, que não aparecem entre os maiores graus ponderados. Esses atores atuam como pontos de ligação entre regiões distintas do grafo, conectando subgrupos que, de outra forma, permaneceriam isolados. Já sebmarkbage, apesar de liderar o grau ponderado, não aparece entre os primeiros colocados em intermediação, o que sugere que sua colaboração é intensa, porém concentrada em um cluster específico da comunidade, com menor alcance entre grupos.

A comparação mostra que os dois algoritmos capturam dimensões diferentes da colaboração. O grau ponderado destaca quem contribui com maior intensidade e frequência, enquanto a intermediação evidencia quem conecta grupos distintos e garante a fluidez estrutural da rede. Em conjunto, essas métricas revelam tanto os atores mais ativos quanto aqueles que exercem funções estratégicas de ponte dentro do ecossistema.

6.2.3. Algoritmo de Louvain

O algoritmo de Louvain foi aplicado ao grafo *GraphCollabPR* para identificar comunidades de desenvolvedores que colaboram mais intensamente entre si. A Tabela 5 apresenta as cinco maiores comunidades detectadas dentro do grafo projetado de colaboração em PullRequest, juntamente com o número de membros em cada grupo.

Tabela 5. Top 5 comunidades detectadas pelo algoritmo de Louvain

id da Comunidade	Número de Membros
880	568
1358	327
309	272
1699	222
95	193

A aplicação do algoritmo Louvain sobre o grafo GraphCollabPR resultou em uma divisão em comunidades. Nelas se destacam cinco grandes grupos: a id 880 com 568 atores, a id 1358 com 327, a id 309 com 272, a id 1699 com 222 e a id 95 com 193 membros. Esses números indicam que existem grupos colaborativos relativamente expressivos dentro da rede de desenvolvedores que participam de Pull Requests, revisões e interações de código. A existência dessas cinco comunidades de tamanho considerável sugere que a colaboração não está uniformemente distribuída, mas sim estruturada em núcleos de atores que interagem com frequência entre si.

Por meio de consultas específicas, foi possível mapear quais atores com maio-

res scores de centralidade de grau e de intermediação pertencem a quais comunidades e repositórios, conforme ilustrado nas Figuras 11 e 12,

```
MATCH (a:Actor)
WHERE a.login IN ['sebmarkbage','gaearon','acdlite','bvaughn','rickhanlonii']
OPTIONAL MATCH (a)-[:created|merged|reacted]→(pr:PullRequest)-[:belongs_to]→(repo:Repository)
WITH a, a.community_louvain AS comunidade, collect(DISTINCT repo.name) AS all_repos
WITH a, comunidade, [r IN all_repos WHERE r IS NOT NULL] AS repos
RETURN a.login AS actor, comunidade, repos
ORDER BY actor;
```

	actor	comunidade	repos
1	"acdlite"	1358	["react"]
2	"bvaughn"	1358	["react"]
3	"gaearon"	1358	["react"]
4	"nickhanlonii"	1358	["react"]
5	"sebmarkbage"	1358	["react"]

Figura 11. Consulta para identificar comunidades dos atores com maior centralidade de grau

Fonte: De autoria própria

```
MATCH (a:Actor)
WHERE a.login IN ['gaearon','dudenamedjune','leonardodino','acdlite','danharrin']
OPTIONAL MATCH (a)-[:created|merged|reacted]→(pr:PullRequest)-[:belongs_to]→(repo:Repository)
WITH a, a.community_louvain AS comunidade, collect(DISTINCT repo.name) AS all_repos
WITH a, comunidade, [r IN all_repos WHERE r IS NOT NULL] AS repos
RETURN a.login AS actor, comunidade, repos
ORDER BY actor;
```

	actor	comunidade	repos
1	"acdlite"	1358	["react"]
2	"danharrin"	95	["filament"]
3	"dudenamedjune"	880	["react"]
4	"gaearon"	1358	["react"]
5	"leonardodino"	880	["react"]

Figura 12. Consulta para identificar comunidades dos atores com maior centralidade de intermediação

Fonte: De autoria própria

Os atores com maiores scores de centralidade de grau se concentram no mesmo repositório, React, e pertencem à comunidade de ID 1358, o que indica que o volume intenso de colaboração está estruturado em torno desse projeto dominante. Por outro lado, na métrica de centralidade de intermediação, observa-se um padrão mais variado. Danharrin, pertencente à comunidade de id 95 do repositório filament, ocupa posição de destaque como intermediador. Gaearon e acdlite, ambas da comunidade de id 1358 do repositório react, mantêm papel de ponte dentro da comunidade central. E leonardodino e dudenamedjune, pertencentes à comunidade de id 880 também do react, conectam subgrupos dentro desse mesmo repositório. Esse arranjo sugere que, enquanto a colaboração em volume está dominada pelo núcleo react-1358, a função de ligação estruturante entre

diferentes grupos se distribui entre comunidades distintas, incluindo uma fora do repositório com maior fluxo de dados.

6.2.4. Considerações sobre as colaborações em pull requests

A projeção do subgrafo GraphCollabPR permitiu concentrar a análise exclusivamente nos atores envolvidos em interações relevantes com Pull Requests, aplicando pesos que refletem a intensidade da colaboração. Essa especialização tornou possível observar a estrutura colaborativa sob uma ótica mais precisa do fluxo de trabalho, em que as ligações representam contribuições efetivas de revisão e aprovação, atividades centrais em projetos baseados em controle de versão. As métricas aplicadas ao subgrafo mostraram que as interações relacionadas a PRs não estão distribuídas de forma homogênea entre os desenvolvedores, mas sim fortemente concentradas em determinados atores que sustentam boa parte da atividade de revisão e integração. Isso reforça a ideia de que grandes projetos podem depender de um conjunto de colaboradores que assumem papéis mais intensivos ou especializados.

A análise de centralidade de grau ponderado destacou atores diretamente ligados ao maior volume de colaborações, todos vinculados ao repositório principal do conjunto analisado. Esses desenvolvedores desempenham funções de alta participação na revisão e integração de código, criando um núcleo de forte densidade no subgrafo. Em contraste, a métrica de centralidade de intermediação revelou atores que, embora não figurem entre os maiores em volume total, assumem papel estrutural na ligação entre diferentes regiões da rede. A presença de atores de outros repositórios, como danharrin, entre os maiores intermediadores demonstra que, mesmo em repositórios menores, há colaboradores que exercem papéis estratégicos na conectividade geral das equipes. Esse contraste entre volume de interação e posição estrutural fortalece a interpretação de que diferentes métricas capturam dimensões distintas da atuação dos desenvolvedores.

A aplicação do algoritmo de detecção de comunidades (Louvain) ao subgrafo revelou grupos de colaboração bem definidos. As cinco maiores comunidades identificadas representam conjuntos densos de interação, associados principalmente ao repositório de maior porte, mas com estrutura interna diferenciada. O fato de atores centrais, tanto em grau quanto em intermediação, pertencerem a comunidades distintas mostra que a estrutura colaborativa não é monolítica: há subgrupos com papéis específicos e padrões de atuação próprios.

Esses achados demonstram que o estudo de subgrafos específicos, como o *GraphCollabPR*, oferece uma visão mais detalhada de diferentes camadas da colaboração dentro de um ecossistema de desenvolvimento. A mesma abordagem pode ser aplicada a outras atividades relevantes, como: Issues, Commits ou Discussões, para revelar padrões complementares de interação e organização. A análise modular por subgrafos permite identificar papéis, núcleos de colaboração, intermediadores estruturais e agrupamentos naturais que não se evidenciam na análise global da rede. Esse tipo de segmentação analítica abre caminho para estudos mais precisos sobre dinâmicas sociais e técnicas em plataformas como o GitHub, permitindo compreender como diferentes formas de contribuição ajudam a estruturar e sustentar projetos de grande escala.

7. Conclusões

Estudos sobre as interações em comunidades de desenvolvimento contemporâneo indicam que o GitHub gera um ecossistema intrincado, em que ações técnicas e sociais se misturam. Os gráficos, juntamente com uma pipeline de coleta por meio do GraphQL, processamento em MongoDB e modelagem no Neo4j, demonstraram ser eficazes para capturar a dinâmica relacional entre atores, artefatos e eventos. A transição para um modelo orientado a grafos permitiu a utilização de algoritmos clássicos de análise de redes, o que trouxe à tona padrões como a colaboração concentrada em um pequeno número de desenvolvedores, a presença de intermediários estruturais e a existência de comunidades densas que se alinham aos fluxos de trabalho baseados em Pull Requests.

Os resultados indicam que a colaboração não é homogênea, mas sim estruturada em grupos que sustentam a maior parte da revisão, integração e coordenação técnica. A identificação de comunidades e estruturas de intermediação indica que o desenvolvimento colaborativo não é apenas o resultado de contribuições intensivas, mas também de conexões estratégicas que sustentam a coesão da rede. Dessa forma, a pesquisa corrobora que a modelagem baseada em grafos é uma ferramenta sólida para elucidar as interações sociais e técnicas em projetos de software modernos, permitindo investigações mais detalhadas sobre colaboração em larga escala.

Trabalhos futuros podem estender a análise para além de Pull Requests, explorando subgrafos centrados em *Issues*, *Commits* ou *Discussions*, permitindo avaliar o impacto de diferentes pesos e combinações na identificação de vínculos colaborativos. O modelo atual também pode ser estendido com análises temporais, possibilitando observar a evolução das estruturas de colaboração ao longo do tempo e identificar mudanças no papel dos atores. Além disso, pode-se considerar a ampliação do conjunto de repositórios para incluir projetos com maior grau de interconexão, nos quais usuários atuam simultaneamente em múltiplos repositórios, podendo revelar dinâmicas mais complexas de interação, especialmente aquelas que emergem entre comunidades distintas.

Essas extensões reforçam que o modelo desenvolvido neste trabalho representa apenas um primeiro passo, mas já evidencia o potencial das técnicas de análise de grafos para revelar dimensões sociais no ecossistema de desenvolvimento colaborativo.

Referências

[Barros et al. 2021] Barros, D., Horita, F., Wiese, I., e Silva, K. (2021). A mining software repository extended cookbook: Lessons learned from a literature review. In *Proceedings of the XXXV Brazilian Symposium on Software Engineering*, pages 1–10.

[Borges et al. 2019] Borges, H., Brito, R., e Valente, M. T. (2019). Beyond textual issues: Understanding the usage and impact of GitHub reactions. In *Proceedings of the XXXIII Brazilian symposium on software engineering*, pages 397–406.

[Borges et al. 2016] Borges, H., Hora, A., e Valente, M. T. (2016). Understanding the factors that impact the popularity of github repositories. In *2016 IEEE international conference on software maintenance and evolution (ICSME)*, pages 334–344. IEEE.

[Borges e Valente 2022] Borges, H. S. e Valente, M. T. (2022). GitHub proxy server: A tool for supporting massive data collection on GitHub. In *Proceedings of the XXXVI Brazilian Symposium on Software Engineering*, pages 370–375.

[Burt 1992] Burt, R. S. (1992). *The social structure of competition*. Cambridge, MA: Harvard University Press.

[Chang et al. 2022] Chang, V., Songala, Y. K., Xu, Q. A., e Liu, B. S.-C. (2022). Scientific data analysis using neo4j. In *FEMIB*, pages 75–84.

[Dabbish et al. 2012] Dabbish, L., Stuart, C., Tsay, J., e Herbsleb, J. (2012). Social coding in GitHub: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on computer supported cooperative work*, pages 1277–1286.

[David e Jon 2010] David, E. e Jon, K. (2010). Networks, crowds, and markets: reasoning about a highly connected world.

[Fu et al. 2021] Fu, E., Zhuang, Y., Zhang, J., Zhang, J., e Chen, Y. (2021). Understanding the user interactions on GitHub: a social network perspective. In *2021 IEEE 24th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 1148–1153. IEEE.

[Gesse Junior 2023] Gesse Junior, R. R. (2023). Mineração de repositórios para análise de ciclos de software.

[GitHub 2025a] GitHub (2025a). GitHub graphql api documentation. `https://docs.github.com/en/graphq`. Acesso em: 27 out. 2025.

[GitHub 2025b] GitHub (2025b). GitHub rest api documentation. `https://docs.github.com/en/rest`. Acesso em: 27 out. 2025.

[GitHub 2025c] GitHub (2025c). GitHub — getting started. `https://docs.github.com/pt/get-started/start-your-journey/about-github-and-git`. Acesso em: 20 nov. 2025.

[Gousios 2013] Gousios, G. (2013). The ghtorent dataset and tool suite. In *2013 10th Working conference on mining software repositories (MSR)*, pages 233–236. IEEE.

[JanusGraph 2025] JanusGraph (2025). Janusgraph documentation. `https://docs.janusgraph.org/`. Acesso em: 29 out. 2025.

[Laranjeira e Cavique 2018] Laranjeira, P. A. e Cavique, L. (2018). Métricas de centralidade em redes sociais.

[Li et al. 2020] Li, H., Wang, T., Xu, X., Jiang, B., Wei, J., e Wang, J. (2020). Modeling software systems as complex networks: analysis and their applications. *Mathematical Problems in Engineering*, 2020(1):5346498.

[Lima et al. 2014] Lima, A., Rossi, L., e Musolesi, M. (2014). Coding together at scale: GitHub as a collaborative social network. In *Proceedings of the International AAAI Conference on Web and Social Media*, volume 8. AAAI.

[MemGraph 2025] MemGraph (2025). Memgraph documentation. `https://memgraph.com/docs`. Acesso em: 29 out. 2025.

[Mpinda et al. 2015] Mpinda, S. A. T., Ferreira, L. C., Ribeiro, M. X., e Santos, M. T. P. (2015). Evaluation of graph databases performance through indexing techniques. *International Journal of Artificial Intelligence & Applications (IJAIA)*, 6(5):87–98.

[Robinson et al. 2015] Robinson, I., Webber, J., e Eifrem, E. (2015). *Graph databases: new opportunities for connected data*. "O'Reilly Media, Inc.".

[Schreiber e Zylka 2020] Schreiber, R. R. e Zylka, M. P. (2020). Social network analysis in software development projects: A systematic literature review. *International Journal of Software Engineering and Knowledge Engineering*, 30(03):321–362.

[Souza et al. 2020] Souza, B. M., Junqueira, A. L. C., Testoni, G. R., Terra, A., e Sirqueira, T. M. F. (2020). Mineração de repositório de software: Investigando a qualidade do software em projetos de código aberto. *ANALECTA-Centro Universitário Academia*, 5(5).

[Thung et al. 2013] Thung, F., Bissyande, T. F., Lo, D., e Jiang, L. (2013). Network structure of social coding in GitHub. In *2013 17th European conference on software maintenance and reengineering*, pages 323–326. IEEE.