Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Explorando Large Language Models para a Geração de Requisitos de Software a partir de Issues em Projetos de Código Aberto

Guilherme Pereira Paiva

Dissertação apresentada como requisito parcial para
conclusão do Mestrado em Informática

Orientadora
Prof.a Dr.a Edna Dias Canedo

Brasília
2025

Ficha catalográfica elaborada automaticamente,
com os dados fornecidos pelo(a) autor(a)

Paiva, Guilherme Pereira
P149e Explorando Large Language Models para a Geração de Requisitos de Software a partir de Issues em Projetos de Código Aberto / Guilherme Pereira Paiva; orientador Edna Canedo. Brasília, 2025.
56 p.

Dissertação(Mestrado em Informática) Universidade de Brasília, 2025.

1. Engenharia de Requisitos. 2. Large Language Models. 3. Engenharia de Prompt. 4. Análise de Qualidade Automatizada. 5. Software de Código Aberto. I. Canedo, Edna, orient. II. Título.

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Explorando Large Language Models para a Geração de Requisitos de Software a partir de Issues em Projetos de Código Aberto

Guilherme Pereira Paiva

Dissertação apresentada como requisito parcial para
conclusão do Mestrado em Informática

Prof.a Dr.a Edna Dias Canedo (Orientadora)
CIC/UnB

Prof. Dr. Geraldo Pereira Rocha Filho
Universidade Estadual do Sudoeste da Bahia (UESB)

Prof. Dr. Rodrigo Pereira dos Santos
Universidade Federal do Estado do Rio de Janeiro (UNIRIO)

Prof.a Dr.a Cláudia Nalon
Coordenadora do Programa de Pós-graduação em Informática

Brasília, 25 de Agosto de 2025

Dedicatória

À minha esposa, pelo amor, paciência e apoio incondicional em todos os momentos;

Aos meus pais e às minhas irmãs, pelo incentivo, confiança e carinho ao longo de toda a jornada;

À minha orientadora, Edna Dias Canedo, pela sábia orientação, pelo comprometimento e pelas valiosas contribuições que nortearam cada etapa deste trabalho.

Agradecimentos

Agradeço a todos que, de alguma forma, contribuíram para a realização deste estudo:

À banca examinadora pelas valiosas sugestões que enriqueceram este trabalho;

Aos colegas e aos amigos do programa de pós-graduação pelo ambiente acolhedor e pelas discussões enriquecedoras;

À Universidade de Brasília, por oferecer infraestrutura e apoio à pesquisa;

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), pelo apoio financeiro concedido para a realização desta pesquisa;

Por fim, registro minha gratidão a Deus, fonte de força e esperança em todos os momentos.

Resumo

Contexto: A Engenharia de Requisitos (ER) em projetos de *Open-Source Software* (OSS) sofre com a informalidade e o grande volume de *issues*, gerando lacunas entre solicitações de usuários e artefatos formais exigidos por normas como a ISO/IEC/IEEE 29148:2018. **Objetivo:** Avaliar empiricamente a eficácia de *Large Language Models* (LLMs) — modulada por diferentes estratégias de engenharia de *prompts* — na geração automática de requisitos de software a partir de títulos de *issues*. **Método:** Foram coletados 150 títulos de *issues* de cinco repositórios OSS altamente ativos; esses títulos foram processados pelos LLMs `o3-mini` e `DeepSeek R1`, combinados com três estilos de *prompt* (Zero-shot, Few-shot e Expert Identity), resultando em 900 requisitos avaliados por um LLM-juiz (`Qwen QwQ-32b`) segundo as métricas Não Ambiguidade, Verificabilidade e Singularidade derivadas da norma ISO/IEC/IEEE 29148. **Resultados:** Ambos os LLMs produziram requisitos de alta qualidade (médias $> 4{,}2$ numa escala 1–5), mas com variação significativa; a estratégia `Few-shot` elevou consistentemente a Singularidade, enquanto a `Expert Identity` melhorou a Verificabilidade ao custo de requisitos menos singulares, evidenciando *trade-offs* dependentes do modelo e do *prompt*. **Conclusão:** LLMs são assistentes promissores para automatizar etapas críticas da ER em OSS, porém sua eficácia exige *prompts* cuidadosamente projetados e supervisão humana para balancear atributos de qualidade concorrentes e assegurar requisitos claros, verificáveis e atômicos.

Palavras-chave: Engenharia de Requisitos, Large Language Models, Engenharia de Prompt, Análise de Qualidade Automatizada, Software de Código Aberto

Abstract

Context: Requirements Engineering (RE) in Open Source Software (OSS) projects suffers from informality and a large volume of issues, creating gaps between user requests and formal artifacts required by standards such as ISO/IEC/IEEE 29148:2018. **Objective:** To empirically evaluate the effectiveness of Large Language Models (LLMs) — modulated by different prompt engineering strategies — in automatically generating software requirements from issue titles. **Method:** 150 issue titles were collected from five highly active OSS repositories; these titles were processed by the LLMs o3-mini and DeepSeek R1, combined with three prompt styles (Zero-shot, Few-shot, and Expert Identity), resulting in 900 requirements evaluated by an LLM-judge (Qwen QwQ-32b) according to the metrics Unambiguity, Verifiability, and Singularity, derived from the ISO/IEC/IEEE 29148 standard. **Results:** Both LLMs produced high-quality requirements (averages $>$ 4.2 on a 1–5 scale), but with significant variation; the Few-shot strategy consistently increased Singularity, while Expert Identity improved Verifiability at the cost of less singular requirements, highlighting model and prompt dependent trade-offs. **Conclusion:** LLMs are promising assistants for automating critical stages of RE in OSS, but their effectiveness requires carefully designed prompts and human supervision to balance competing quality attributes and ensure clear, verifiable, and atomic requirements.

Keywords: Requirements Engineering, Large Language Models, Prompt Engineering, Automated Quality Assessment, Open Source Software

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

ER Engenharia de Requisitos.

IA Inteligência Artificial.

LLMs *Large Language Models*.

OSS *Open-Source Software*.

ZSL *Zero-Shot Learning*.

Capítulo 1

Introdução

1.1 Contextualização

A Engenharia de Requisitos (ER) é reconhecida como uma fase fundamental e, simultaneamente, um dos principais gargalos no ciclo de vida do desenvolvimento de software. A qualidade desta etapa é um fator determinante para o sucesso de um projeto, pois falhas na elicitação, especificação ou gerenciamento de requisitos podem levar a atrasos, custos excessivos e ao desenvolvimento de um produto que não atende às necessidades das partes interessadas [1, 2].

Este desafio é particularmente amplificado no ecossistema dinâmico e distribuído do Software de Código Aberto – *Open-Source Software* (OSS). Nesses projetos, a elicitação de requisitos raramente segue processos formais. Em vez disso, ela depende massivamente de sistemas de rastreamento de *issues*[1] em plataformas como o GitHub, onde desenvolvedores e usuários reportam falhas, discutem melhorias e propõem novas funcionalidades de maneira informal [3, 4]. Essa realidade cria uma lacuna significativa entre, de um lado, um grande volume de solicitações concisas, ambíguas e não estruturadas e, de outro, a necessidade de requisitos de software claros, verificáveis e singulares, conforme preconizado por normas como a ISO/IEC/IEEE 29148:2018 [5].

Neste cenário, os *Large Language Models* (LLMs) surgem como uma tecnologia promissora. Com sua capacidade avançada de compreender e gerar texto de forma análoga à humana [6], esses modelos de Inteligência Artificial (IA) oferecem um potencial sem precedentes para automatizar a tarefa de "traduzir" as solicitações informais dos usuários em artefatos de requisitos bem formados, um objetivo central para a modernização da ER [7]. A capacidade dos LLMs de realizar tarefas complexas com pouca ou nenhuma supervisão contorna a dependência de grandes conjuntos de dados rotulados, que eram uma limitação de abordagens de IA anteriores [8].

[1] `https://docs.github.com/pt/issues`

Contudo, a eficácia dos LLMs não é incondicional. A qualidade da saída gerada é diretamente influenciada pela entrada fornecida, conhecida como *prompt*. A prática de projetar e refinar esses *prompts* — a Engenharia de *Prompts* — tornou-se uma disciplina crítica, pois funciona como uma "forma de programação" para guiar o comportamento do modelo [9]. Diferentes estratégias de *prompt* podem levar a resultados drasticamente distintos [10], tornando essencial a investigação sobre como otimizar essa interação para tarefas específicas de ER.

Diante deste contexto, o presente trabalho se propõe a investigar a aplicação de LLMs para a geração automática de requisitos de software a partir de uma das fontes mais desafiadoras e realistas do mundo OSS: os títulos de *issues* do GitHub. Mais especificamente, esta dissertação realiza uma avaliação empírica e comparativa para entender como diferentes estratégias de engenharia de *prompts* e a escolha de distintos LLMs impactam a qualidade dos requisitos gerados, oferecendo insights práticos para a automação de uma das etapas mais críticas do desenvolvimento de software.

1.2 Problema de Pesquisa

A elicitação e especificação de requisitos são universalmente reconhecidas como atividades cognitivamente desafiadoras e cruciais para o sucesso de um projeto. Uma pesquisa recente com profissionais da área de desenvolvimento, conduzida por Mesquita et al. [11], corrobora essa visão, indicando que a percepção de dificuldade está ligada a desafios de comunicação, como gerenciar o relacionamento com as partes interessadas e compreender processos de negócio complexos. Essa dificuldade inerente ao processo manual frequentemente resulta em artefatos de requisitos iniciais que são informais, ambíguos ou incompletos.

Essa fragilidade na criação dos artefatos de requisitos é corroborada por Canedo et al. [12]. A pesquisa dos autores com equipes ágeis revelou que os desafios na documentação são acentuados pela carência de profissionais com treinamento específico em requisitos e pela falta de conhecimento sobre as técnicas e métodos da ER.

Este problema se manifesta de forma aguda no ecossistema de OSS. Nesses projetos, as barreiras de comunicação são amplificadas pela natureza distribuída das equipes, e o processo de elicitação depende de um fluxo contínuo de *issues* onde as solicitações de funcionalidades são expressas de forma extremamente concisa, muitas vezes apenas em um título [3]. Portanto, existe uma lacuna pragmática entre a forma como os requisitos nascem no mundo OSS e a necessidade de artefatos que atendam a critérios de qualidade rigorosos, como os da norma ISO/IEC/IEEE 29148:2018.

Embora a literatura demonstre o potencial dos LLMs para automatizar tarefas de ER [13, 14], sua aplicação para transpor essa lacuna específica permanece pouco explorada. O desempenho dos LLMs é altamente sensível às instruções recebidas, e a engenharia de *prompts* emergiu como um fator crítico para o sucesso [9, 10]. Contudo, há pouca evidência empírica sobre como diferentes estratégias de *prompt* influenciam a qualidade dos requisitos gerados neste contexto.

A ausência de uma avaliação sistemática que compare o desempenho de diferentes LLMs e estratégias de *prompt*, utilizando métricas de qualidade padronizadas para a tarefa específica de gerar requisitos a partir de títulos de *issues* de OSS, configura o principal problema de pesquisa a ser abordado. É preciso ir além da prova de conceito e investigar a eficácia, a consistência e os *trade-offs* envolvidos nesta aplicação prática.

Portanto, esta dissertação é norteada por um objetivo central: **avaliar empiricamente a eficácia de LLMs, modulada por diferentes estratégias de engenharia de *prompts*, na geração de requisitos de software de alta qualidade a partir de títulos de *issues* de projetos OSS**.

Para alcançar este objetivo, o estudo busca responder às seguintes questões de pesquisa (QPs):

- **QP1:** Qual a eficácia dos LLMs para gerar requisitos de software a partir de títulos de *issues* de projetos OSS?
- **QP2:** Como diferentes estratégias de engenharia de *prompts* influenciam a qualidade dos requisitos gerados?
- **QP3:** Quais são as características qualitativas, os pontos fortes e fracos dos requisitos gerados por LLMs?

1.3 Justificativa

A relevância desta pesquisa ancora-se na intersecção de três pilares: um desafio prático persistente na Engenharia de Software, uma lacuna acadêmica na aplicação de tecnologias emergentes e uma necessidade metodológica de avaliação em larga escala.

Primeiramente, a **relevância prática** reside em abordar um dos gargalos mais críticos do desenvolvimento de software: a Engenharia de Requisitos. Conforme apontado por Arora et al. [7], a ER é frequentemente subdimensionada devido a restrições de tempo e recursos, apesar de ser uma fonte conhecida de falhas e retrabalho [2]. Este desafio é acentuado no contexto do *Open-Source Software*, onde a elicitação ocorre de forma distribuída e informal, predominantemente através de um volume massivo de *issues* em plataformas como o GitHub [3, 4]. A tarefa de transformar títulos de *issues* — que

são, por natureza, concisos e muitas vezes ambíguos — em requisitos de software claros, verificáveis e singulares, alinhados com padrões como a ISO/IEC/IEEE 29148:2018 [5], constitui um problema prático para a evolução de projetos OSS.

Em segundo lugar, a **relevância acadêmica** emerge da necessidade de investigar sistematicamente o potencial dos LLMs para solucionar o problema descrito. A literatura recente já demonstrou que os LLMs são promissores para automatizar diversas tarefas de ER [13, 8]. Contudo, persistem lacunas que este trabalho se propõe a preencher. Estudos anteriores não focaram adequadamente no desafio de gerar requisitos a partir de entradas tão restritas e informais como os títulos de *issues* de OSS. Além disso, embora a importância da engenharia de *prompts* seja reconhecida [9, 10], falta uma avaliação empírica comparativa em larga escala que investigue como diferentes estratégias de *prompt* e a escolha de distintos LLMs influenciam atributos de qualidade específicos. Este estudo avança ao investigar não apenas a eficácia dos modelos, mas também os *trade-offs* entre as métricas de qualidade, um aspecto prático e pouco explorado.

Finalmente, a **relevância metodológica** desta dissertação está na adoção e validação do paradigma *LLM-as-a-Judge* para a avaliação da qualidade dos artefatos gerados. A avaliação manual de um grande volume de requisitos seria impraticável. A abordagem *LLM-as-a-Judge* oferece uma alternativa escalável, consistente e de baixo custo [15]. Ao empregar um LLM avaliador fundamentado em critérios explícitos da norma ISO 29148, este trabalho não só viabiliza a análise em larga escala, mas também contribui com evidências sobre a robustez desta metodologia para mitigar vieses conhecidos no domínio específico da ER.

Portanto, esta dissertação justifica-se por fornecer evidências empíricas e insights práticos sobre a aplicação de LLMs a um problema real da ER em projetos OSS, avançando o conhecimento sobre a interação entre modelos, *prompts* e qualidade dos requisitos, ao mesmo tempo em que explora uma abordagem de avaliação automatizada promissora.

1.4 Objetivos

Com base no problema de pesquisa delineado, os objetivos deste trabalho foram estruturados da seguinte forma:

1.4.1 Objetivo Geral

O objetivo geral deste trabalho é avaliar empiricamente a eficácia de LLMs, modulada por diferentes estratégias de engenharia de *prompts*, na geração de requisitos de software de alta qualidade a partir de títulos de *issues* de projetos OSS.

1.4.2 Objetivos Específicos

Para alcançar o objetivo geral, os seguintes objetivos específicos foram definidos:

1. Coletar um conjunto de dados composto por títulos de *issues* de solicitações de funcionalidades de repositórios OSS de alta atividade no GitHub.
2. Desenvolver estratégias de engenharia de *prompts* para guiar a geração de requisitos.
3. Gerar um corpus de requisitos de software utilizando LLMs distintos em combinação com as estratégias de *prompt*.
4. Definir um protocolo de avaliação de qualidade baseado em métricas da norma ISO/IEC/IEEE 29148:2018.
5. Avaliar a qualidade dos requisitos gerados de forma automatizada, utilizando um modelo de LLM como avaliador.
6. Analisar os dados quantitativos e qualitativos para responder às questões de pesquisa, comparando a eficácia dos LLMs e das estratégias de *prompt* e identificando os pontos fortes e fracos das saídas geradas.

1.5 Resultados Esperados e Contribuições

A execução desta pesquisa visa gerar um conjunto de contribuições teóricas, práticas e metodológicas para a área de Engenharia de Software. Os principais resultados e contribuições esperados são:

1. **Um conjunto de dados público e anotado:** Disponibilização de um dataset contendo 150 títulos de *issues* de projetos OSS e os 900 requisitos de software gerados. Cada requisito é acompanhado por pontuações de qualidade e pela justificativa textual do LLM-juiz, servindo como um recurso valioso para futuras pesquisas.
2. **Evidência empírica sobre a eficácia de LLMs:** Uma avaliação quantitativa e qualitativa que responde à questão de quão eficazes são os LLMs para a tarefa de gerar requisitos a partir de entradas concisas, oferecendo uma visão realista de suas capacidades e limitações atuais.
3. **Análise comparativa de estratégias de *prompt*:** Um estudo aprofundado sobre como diferentes estratégias de engenharia de *prompts* impactam a qualidade dos requisitos. A pesquisa identifica os *trade-offs* gerados por cada estratégia, fornecendo orientações práticas para a escolha do *prompt* mais adequado.

4. **Validação de uma metodologia de avaliação escalável:** Demonstração e validação do uso do paradigma *LLM-as-a-Judge* para avaliação de qualidade em ER, incluindo evidências de sua robustez contra vieses conhecidos e sua utilidade para análises em larga escala.

1.6 Método de Pesquisa

Para alcançar os objetivos propostos, esta pesquisa adota um desenho empírico e quantitativo, aderindo a padrões para pesquisa em Engenharia de Software definidos pela ACM SIGSOFT [16]. Especificamente, o trabalho combina os padrões:

- **Mineração de Repositórios**[2]: Para a extração sistemática de solicitações de funcionalidades (*issues*) de repositórios de software de código aberto no GitHub, formando a base de dados para o estudo.
- **Ciência de Dados**[3]**:** Para o desenho, condução e avaliação dos experimentos de geração automática de requisitos, incluindo a aplicação de modelos, análise estatística e interpretação dos resultados.

O fluxo metodológico, detalhado no Capítulo 3, foi executado em três fases principais. Na **Coleta de Dados**, foram minerados 150 títulos de *issues* de cinco repositórios OSS de alta relevância. Na **Geração de Requisitos**, estes títulos foram processados por dois LLMs (`o3-mini` e `DeepSeek R1`) com três estratégias de *prompt* distintas, resultando em 900 requisitos. Finalmente, na **Avaliação de Qualidade**, foi empregado o paradigma *LLM-as-a-Judge* com o modelo `Qwen QwQ-32b` para avaliar cada requisito segundo as métricas de Não Ambiguidade, Verificabilidade e Singularidade da norma ISO/IEC/IEEE 29148:2018.

Os resultados obtidos a partir do processo de geração de requisitos, bem como os *scripts* utilizados, foram disponibilizados publicamente na plataforma de arquivamento e versionamento *Zenodo*[4]. Essa iniciativa visa garantir a transparência, a disponibilidade e a replicabilidade das etapas realizadas.

1.7 Estrutura do Trabalho

Esta dissertação está organizada em cinco capítulos, além deste, que se desdobram da seguinte forma:

[2]https://www2.sigsoft.org/EmpiricalStandards/docs/standards?standard=RepositoryMining
[3]https://www2.sigsoft.org/EmpiricalStandards/docs/standards?standard=DataScience
[4]`https://zenodo.org/records/13655334`

- **Capítulo 2: Fundamentação Teórica** — Revisa os conceitos essenciais de Engenharia de Requisitos, qualidade de requisitos segundo a norma ISO/IEC/IEEE 29148:2018, LLMs, engenharia de *prompts* e o paradigma *LLM-as-a-Judge*. Também analisa os trabalhos relacionados para posicionar esta pesquisa na literatura atual;
- **Capítulo 3: Configuração do Estudo** — Descreve em detalhes o método adotado, incluindo os procedimentos para coleta de dados, a configuração dos LLMs e *prompts* para a geração de requisitos, e o protocolo de avaliação automatizada da qualidade;
- **Capítulo 4: Resultados e Discussão** — Apresenta e analisa os resultados quantitativos e qualitativos obtidos. Cada questão de pesquisa é abordada, discutindo a eficácia dos LLMs, a influência dos *prompts* e as características dos requisitos gerados. Também analisa criticamente as limitações do estudo e as potenciais ameaças à sua validade, com o objetivo de contextualizar o alcance dos resultados;
- **Capítulo 5: Conclusão** — Recapitula as contribuições da dissertação, sintetiza as conclusões em resposta às questões de pesquisa, e delineia as implicações dos resultados e as direções para trabalhos futuros.

Capítulo 2

Fundamentação

Este capítulo estabelece as bases teóricas para a presente dissertação. Inicia-se discutindo a extração de requisitos a partir de artefatos de software em projetos de código aberto, definindo em seguida os critérios de qualidade para requisitos com base na norma ISO/IEC/IEEE 29148:2018 [5]. Subsequentemente, são apresentados LLMs, com foco em técnicas de engenharia de prompts. Por fim, aborda-se o paradigma emergente *LLM-as-a-Judge* para avaliação em larga escala e analisam-se os trabalhos relacionados para identificar a lacuna de pesquisa que este trabalho se propõe a preencher.

2.1 Fundamentação Teórica

2.1.1 Requisitos de Software

A ER é um processo sistemático e disciplinado fundamental no ciclo de vida do desenvolvimento de software. Ela abrange o conjunto de atividades focadas em descobrir, analisar, documentar e verificar as funcionalidades e restrições de um sistema [1]. Uma ER eficaz é crucial para o sucesso de um projeto, pois estabelece uma base sólida que alinha as expectativas das partes interessadas (*stakeholders*) com o produto final, mitigando riscos de falhas, atrasos e custos excessivos [2].

O processo de ER é composto por um conjunto de atividades inter-relacionadas e iterativas. Embora possam ser apresentadas de forma sequencial para fins didáticos, na prática, essas atividades são frequentemente sobrepostas e revisitadas ao longo do projeto, em um ciclo contínuo de refinamento [1]. As principais atividades incluem:

- **Elicitação e Análise:** A fase de descoberta, onde os requisitos são identificados por meio da interação com *stakeholders* e da análise do domínio do problema. Esta é uma das fases mais críticas e desafiadoras, pois envolve a tradução de necessidades, muitas vezes vagas ou implícitas, em conceitos concretos [1, 2];

- **Especificação:** A atividade de converter os requisitos elicitados em um formato padronizado, claro e inequívoco. O resultado é um documento de requisitos que serve como um contrato entre clientes e desenvolvedores, guiando o projeto, desenvolvimento e testes subsequentes [1];
- **Validação:** O processo de verificação para garantir que os requisitos especificados realmente definem o sistema que o cliente deseja e necessita. Esta fase busca identificar problemas como omissões, inconsistências e ambiguidades antes que eles se propaguem para as fases posteriores do desenvolvimento [1];
- **Gerenciamento de Requisitos:** Dado que os requisitos são raramente estáticos, esta atividade lida com as mudanças inevitáveis que ocorrem durante o ciclo de vida do projeto. Inclui o controle de alterações, a análise de impacto e a manutenção da rastreabilidade entre os requisitos e outros artefatos do sistema [1].

Desafios na Engenharia de Requisitos

A elicitação e a especificação de requisitos são processos inerentemente complexos e repletos de desafios. Um dos obstáculos mais significativos reside na comunicação e na própria natureza da linguagem natural. Requisitos são frequentemente expressos de forma vaga, ambígua, inconsistente ou incompleta, o que pode levar a múltiplas interpretações e, consequentemente, a falhas no desenvolvimento [2].

Outros desafios importantes incluem [1, 2]:

- **Dificuldade na Articulação:** Muitas vezes, os *stakeholders* não sabem exatamente o que querem de um sistema ou têm dificuldade em articular suas necessidades de forma clara e precisa.
- **Requisitos Conflitantes:** Diferentes *stakeholders* podem ter necessidades e prioridades divergentes, resultando em requisitos que entram em conflito entre si.
- **Conhecimento Implícito:** Especialistas de domínio possuem conhecimento tácito sobre seus processos de trabalho, que consideram tão fundamental a ponto de não o mencionarem, levando a omissões críticas.
- **Ambiente Dinâmico:** Fatores de negócio, econômicos e políticos podem mudar durante o processo de análise, alterando a importância de certos requisitos ou introduzindo novos.

Essas dificuldades ressaltam a importância de técnicas e ferramentas que possam auxiliar na transformação de entradas informais e ambíguas em requisitos bem definidos.

Além disso, a Engenharia de Requisitos enfrenta desafios como tempo, restrições orçamentárias, ferramentas inadequadas e uma abordagem que prioriza especialmente a implementação, levando a problemas como requisitos inconsistentes, incompletos e incorretos em estágios posteriores de desenvolvimento [7].

Elicitação de Requisitos em Projetos OSS

Os desafios da ER são particularmente amplificados no contexto de projetos OSS. Estes projetos são caracterizados por um desenvolvimento distribuído, colaborativo e, muitas vezes, menos formal, onde a comunidade de desenvolvedores e usuários desempenha um papel central [3]. Nesses ambientes, a elicitação de requisitos raramente segue um processo estruturado de entrevistas ou *workshops*.

Em vez disso, a elicitação em projetos OSS depende fortemente de ferramentas de comunicação e documentação online, principalmente dos sistemas de rastreamento de issues (*issue trackers*), como os encontrados em plataformas como GitHub, Jira e Bugzilla [4]. Um *issue tracker* é um sistema que permite aos usuários e desenvolvedores reportar *bugs*, solicitar novas funcionalidades, fazer perguntas e discutir aspectos técnicos do projeto. Cada entrada (*issue*) funciona como um tópico de discussão que pode evoluir para um requisito de software [3, 4].

O uso de *issue trackers* como principal fonte de requisitos introduz um conjunto único de desafios:

- **Grande Volume de Entradas:** Projetos OSS populares recebem um volume massivo de *issues* e pedidos de funcionalidades, tornando difícil para os desenvolvedores gerenciar, classificar e priorizar essas solicitações de forma eficaz [3].
- **Informalidade e Falta de Estrutura:** As *issues* são frequentemente escritas em linguagem natural informal, com vocabulário e estrutura sintática muito diferentes dos requisitos formais. Títulos de *issues*, em particular, podem ser extremamente concisos, ambíguos ou incompletos [17].
- **Comunicação Distribuída:** A natureza distribuída e assíncrona da comunicação em *issue trackers* dificulta a obtenção de esclarecimentos e a resolução de conflitos entre os interesses dos diversos *stakeholders* [3].

Essa lacuna entre a natureza informal e massiva das solicitações em *issue trackers* de OSS e a necessidade de requisitos de software claros, singulares e verificáveis cria uma oportunidade significativa para a automação. A capacidade de transformar automaticamente entradas concisas e informais, como os títulos de *issues*, em requisitos bem formados poderia otimizar drasticamente o processo de ER em projetos OSS, motivando a exploração de tecnologias como LLMs para esta tarefa.

2.1.2 Qualidade em Requisitos de Software

A norma ISO/IEC/IEEE 29148:2018 [5] é o padrão internacional que estabelece diretrizes para os processos e produtos da Engenharia de Requisitos. Ela fornece um arcabouço robusto para a definição da qualidade de um requisito, detalhando um conjunto de características que cada declaração deve possuir para ser considerada apropriadamente elaborada.

Segundo a norma, um requisito bem escrito deve, primeiramente, seguir boas práticas de redação. Ele deve ser expresso em voz ativa e utilizar declarações positivas, evitando negações como "o sistema não deve". O uso de termos vagos deve ser evitado, e a terminologia específica do domínio deve ser formalmente definida e aplicada de maneira consistente. Estruturalmente, uma declaração de requisito deve conter um sujeito claro (por exemplo, "o sistema", "o compilador"), um verbo modal que indique obrigatoriedade: tipicamente "deve" (*shall*), e uma descrição da funcionalidade ou restrição a ser atendida.

Além dessas diretrizes de estilo, a ISO/IEC/IEEE 29148:2018 define que cada requisito individual deve possuir um conjunto de atributos essenciais de qualidade:

- **Necessário:** O requisito define uma funcionalidade, característica ou restrição essencial. Sua ausência resultaria em uma deficiência no sistema que não poderia ser suprida por outros requisitos.
- **Apropriado:** O nível de detalhe do requisito é adequado ao nível de abstração da entidade a que se refere. Ele deve evitar impor restrições de design desnecessárias, permitindo liberdade na implementação.
- **Não Ambíguo:** O requisito é redigido de forma que só possa ser interpretado de uma única maneira. A declaração deve ser simples e de fácil compreensão para todas as partes interessadas.
- **Completo:** O requisito descreve suficientemente a funcionalidade necessária, sem que informações adicionais sejam precisas para seu entendimento. Ele deve conter toda a informação necessária para que seja possível projetá-lo e testá-lo.
- **Singular:** O requisito expressa uma única funcionalidade, característica, restrição ou fator de qualidade. Ele deve ser atômico, evitando a combinação de múltiplas necessidades em uma única declaração para facilitar o rastreamento e a verificação.
- **Viável:** O requisito pode ser implementado dentro das restrições do sistema (como custo, prazo e tecnologia) com um nível de risco aceitável.
- **Verificável:** O requisito é estruturado e redigido de modo que sua implementação no sistema possa ser verificada de forma objetiva, seja por meio de testes, inspeção,

análise ou demonstração. A verificabilidade é aprimorada quando o requisito é mensurável.

- **Correto:** O requisito representa com precisão a necessidade da entidade (cliente, usuário) da qual foi derivado.
- **Conforme:** O requisito individual segue um modelo ou estilo de escrita padronizado e aprovado pela organização ou pelo projeto, quando aplicável.

No contexto deste trabalho, a avaliação da qualidade concentra-se em um subconjunto pragmático dessas características: Não Ambiguidade, Verificabilidade e Singularidade. Essa seleção é justificada pois tais atributos podem ser avaliados de forma mais objetiva em requisitos isolados, sem a necessidade do contexto completo do projeto, como cronogramas, orçamentos e detalhes de arquitetura, que muitas vezes estão ausentes ou são pouco explícitos nos ambientes de OSS. A capacidade de gerar requisitos de qualidade é um passo fundamental para automatizar e otimizar a ER nesses ecossistemas dinâmicos [3].

2.1.3 Large Language Models

Large Language Models representam um avanço significativo no campo da Inteligência Artificial, caracterizados pela sua capacidade de compreender, gerar e interagir utilizando linguagem natural de forma sofisticada. São modelos de redes neurais com bilhões de parâmetros, treinados em vastos volumes de dados textuais, o que lhes confere uma proficiência em uma ampla gama de tarefas linguísticas [6, 18].

A principal vantagem dos LLMs reside em sua versatilidade. Eles podem atuar como "solucionadores de tarefas de propósito geral", capazes de seguir instruções humanas para executar tarefas complexas e novas, muitas vezes sem a necessidade de exemplos explícitos [18]. Essa capacidade, conhecida como "seguimento de instruções", é uma das habilidades emergentes que os distinguem de modelos de IA anteriores. Outra habilidade crucial é o raciocínio em múltiplos passos, que permite aos LLMs decompor problemas complexos em etapas intermediárias para derivar uma solução, tornando-os blocos de construção fundamentais para o desenvolvimento de agentes de IA mais gerais [18].

A Arquitetura Transformer

A viabilidade e o sucesso dos LLMs modernos são intrinsecamente ligados à arquitetura *Transformer*, introduzida por Vaswani et al. em 2017 [19]. Antes de sua criação, modelos de Redes Neurais processavam os dados sequencialmente, o que criava um gargalo computacional e dificultava a captura de dependências de longo prazo no texto.

O *Transformer* revolucionou o campo ao propor uma arquitetura que dispensa completamente a recorrência, baseando-se exclusivamente em mecanismos de atenção [19]. O coração do *Transformer* é o mecanismo de auto-atenção, que permite ao modelo ponderar a importância de todas as palavras em uma sequência em relação umas às outras, capturando o contexto de forma muito mais eficaz. Crucialmente, esse processo pode ser executado em paralelo para cada palavra, o que tornou o treinamento em *hardware* moderno muito mais eficiente. Essa característica foi o que permitiu escalar os modelos de linguagem para centenas de bilhões de parâmetros, tornando o *Transformer* a arquitetura padrão para o desenvolvimento de LLMs [6, 18].

Em sua forma original, a arquitetura consiste em um *encoder* (codificador) e um *decoder* (decodificador). O codificador é responsável por processar a sequência de entrada e criar uma representação numérica rica em contexto. O decodificador, por sua vez, utiliza essa representação para gerar a sequência de saída, um token de cada vez, de forma autorregressiva [19]. O desenvolvimento de um LLM segue um paradigma de duas etapas principais: pré-treinamento e ajuste fino [6, 18].

Na fase de pré-treinamento, o modelo é treinado em um corpus massivo e não rotulado de texto, extraído da internet e de outras fontes. O treinamento é auto-supervisionado, geralmente com objetivos como prever a próxima palavra em uma sentença. É nesta fase que o modelo adquire seu conhecimento fundamental sobre a linguagem, incluindo gramática, fatos, capacidade de raciocínio e compreensão semântica. Esse conhecimento generalista é o que permite que os LLMs executem tarefas para as quais não foram explicitamente treinados [18, 6].

Após o pré-treinamento, o modelo passa pela fase de ajuste fino. Nela, o LLM, já dotado de conhecimento geral, é treinado em um conjunto de dados menor e mais específico para aprimorar suas capacidades em tarefas particulares ou para se alinhar com as expectativas humanas. Uma forma proeminente de ajuste fino é o "ajuste de instrução", onde o modelo é treinado com exemplos de instruções e as respostas desejadas. Esse processo é fundamental para aprimorar a capacidade de seguir instruções complexas e produzir respostas mais úteis e seguras [6, 18].

As variações na arquitetura *Transformer* levaram a diferentes famílias de LLMs. As principais são:

- ***Encoder-Decoder***: Utilizam tanto o codificador quanto o decodificador. São comuns em tarefas de tradução ou sumarização.
- ***Decoder-Only***: Utilizam apenas uma pilha de decodificadores com uma máscara de atenção que garante que cada token só possa "ver" os tokens anteriores. Essa arquitetura, chamada de "decoder causal", é a base de modelos focados em geração de texto, como a série GPT [20].

- ***Encoder-Only***: Modelos como o BERT [21] utilizam apenas o codificador para criar representações profundas do texto, sendo altamente eficazes em tarefas de compreensão de linguagem, como classificação de texto e análise de sentimento [6].

Limitações e Considerações Éticas

Apesar de suas capacidades, os LLMs possuem limitações significativas. Eles podem gerar informações factualmente incorretas (fenômeno conhecido como "alucinação"), reproduzir vieses presentes nos dados de treinamento e produzir respostas tóxicas ou prejudiciais [18]. Esforços para alinhar os modelos aos valores humanos, por meio de técnicas como o Aprendizado por Reforço com Feedback Humano (*Reinforcement Learning from Human Feedback* - RLHF) [22], buscam mitigar esses problemas, mas podem introduzir um "imposto de alinhamento", onde a melhoria na segurança e conformidade pode levar a uma ligeira perda de capacidade em outras áreas [6].

As preocupações éticas e sociais incluem o potencial de uso indevido para desinformação, a perpetuação de estereótipos, questões de privacidade de dados e o considerável custo computacional e ambiental associado ao treinamento desses modelos massivos. Portanto, o desenvolvimento e a aplicação de LLMs exigem uma abordagem crítica e responsável [18].

2.1.4 Engenharia de Prompts

A interação com LLMs é mediada por instruções em linguagem natural conhecidas como *prompts*. A prática de projetar, refinar e implementar sistematicamente esses prompts para guiar o comportamento de um LLM e otimizar seus resultados em tarefas específicas é denominada Engenharia de Prompts [23]. Essa disciplina é importante, pois a qualidade, precisão e relevância da resposta de um LLM estão diretamente condicionadas à clareza e eficácia da instrução recebida. De acordo com White et al. [9], os prompts funcionam como uma "forma de programação" para instruir um LLM a executar tarefas de engenharia de software.

A necessidade dessa engenharia surge da própria natureza da linguagem natural, que pode ser inerentemente ambígua. Um prompt mal formulado pode levar a interpretações equivocadas por parte do modelo, resultando em saídas indesejadas ou desalinhadas com a intenção do usuário [10]. Portanto, a engenharia de prompts visa mitigar essa ambiguidade e a "dirigir o comportamento do modelo para os resultados desejados" [23].

Estratégias Fundamentais de Prompt

A literatura estabeleceu diversas técnicas para estruturar prompts, que variam em complexidade e na quantidade de contexto fornecido ao modelo [23, 24, 25]. Essas estratégias são frequentemente categorizadas com base no conceito de aprendizagem em contexto, onde o modelo adapta seu comportamento com base nas informações contidas no próprio prompt, sem a necessidade de re-treinamento ou ajuste fino de seus parâmetros [24].

Zero-shot Prompting É a abordagem mais direta, na qual o LLM recebe apenas uma instrução em linguagem natural descrevendo a tarefa, sem nenhum exemplo de como executá-la [24, 10]. Essa técnica depende inteiramente do conhecimento pré-treinado do modelo para interpretar a solicitação e gerar uma resposta adequada. Embora seja a mais simples de implementar, sua eficácia pode ser limitada em tarefas complexas ou que exigem um formato de saída muito específico.

Few-shot Prompting Para aprimorar a precisão e guiar o modelo de forma mais eficaz, a técnica de *few-shot prompting* é empregada. Ela consiste em incluir no prompt, além da instrução, alguns exemplos de pares "entrada-saída" que demonstram a tarefa a ser realizada [24]. Esses exemplos servem como um condicionamento, permitindo que o modelo infira o padrão e o formato esperados para a resposta. A eficácia dessa abordagem pode ser tão sensível que até mesmo a ordem dos exemplos no prompt pode influenciar significativamente o desempenho do modelo [10].

Além das estratégias básicas, técnicas mais sofisticadas foram desenvolvidas para lidar com problemas complexos e para sistematizar a interação com os LLMs.

Expert Identity Prompting Uma estratégia poderosa é a de atribuir uma persona ou identidade de especialista ao LLM. A técnica *ExpertPrompting*, proposta por Xu et al. [25], é uma estratégia de prompt aumentada que instrui o modelo a responder como um especialista em um determinado domínio. Ao condicionar a resposta a uma identidade detalhada (por exemplo, "Você é um Engenheiro de Requisitos de Software com vasta experiência..."), o objetivo é extrair o potencial latente do modelo para gerar respostas mais abrangentes, detalhadas e de maior qualidade, especialmente em tarefas que requerem conhecimento aprofundado.

Chain-of-Thought Prompting Para problemas que exigem raciocínio complexo em múltiplas etapas, a técnica de *Chain-of-Thought* (Cadeia de Pensamento) se mostrou eficaz. Ela consiste em instruir o modelo a "gerar uma série coerente de passos de raciocínio intermediários que levam à resposta final" [10]. Ao externalizar o processo de raciocínio, o LLM tende a cometer menos erros em tarefas lógicas e matemáticas, melhorando sua capacidade de resolução de problemas.

2.1.5 LLM-as-a-Judge

A avaliação da qualidade de textos gerados por LLMs, especialmente em tarefas abertas como a geração de requisitos, apresenta desafios significativos. Métricas tradicionais baseadas em correspondência, como ROUGE [26] e BLEU [27], são frequentemente inadequadas, pois não conseguem capturar atributos sutis como clareza, relevância ou adequação ao contexto [28]. Em resposta a essa lacuna, emergiu um novo paradigma de avaliação conhecido como *LLM-as-a-Judge* (LLM como Juiz), proposto por Zheng et al. [15]. A premissa central é utilizar LLMs de ponta como substitutos de avaliadores humanos para julgar a qualidade dos resultados de outros modelos [15, 29]. Essa abordagem aproveita a capacidade dos LLMs de compreender instruções complexas e raciocinar de forma análoga à humana para realizar avaliações detalhadas e contextuais, superando as limitações dos métodos de avaliação automática tradicionais [28].

A adoção do *LLM-as-a-Judge* oferece vantagens substanciais, principalmente em termos de escalabilidade, custo e consistência. A avaliação humana, embora considerada o padrão-ouro, é um processo caro, demorado e difícil de escalar, especialmente para grandes volumes de dados, como os requisitos gerados neste estudo [15]. O *LLM-as-a-Judge* automatiza esse processo, oferecendo uma alternativa econômica e ágil [29]. Além disso, um LLM-juiz é menos suscetível a fatores como fadiga ou variabilidade subjetiva que podem afetar a consistência entre múltiplos avaliadores humanos [29]. Outro benefício chave é a explicabilidade: LLMs podem ser instruídos a fornecer não apenas uma pontuação, mas também uma justificativa textual detalhada para suas avaliações, tornando o processo mais transparente e interpretável, uma capacidade que foi fundamental para a análise qualitativa realizada nesta pesquisa [15].

Quando comparado à avaliação humana, o paradigma *LLM-as-a-Judge* demonstra uma notável convergência. Estudos mostram que LLMs robustos, como o GPT-4, podem atingir níveis de concordância com as preferências humanas superiores a 80%, um patamar comparável ao nível de concordância entre os próprios humanos [15]. Essa alta correlação com o julgamento humano valida seu uso como uma alternativa eficaz para avaliar atributos complexos e subjetivos, como a qualidade de um requisito de software. Ao mimetizar o raciocínio humano e alinhar-se com suas preferências, os LLMs-juízes combinam a profundidade da avaliação especializada com a escalabilidade dos métodos automáticos [29].

Apesar de seu potencial, a abordagem *LLM-as-a-Judge* não está isenta de limitações. Uma das principais preocupações é a suscetibilidade a vieses inerentes aos modelos. Entre os mais documentados estão:

- **Viés de Posição:** A tendência de favorecer a primeira resposta apresentada em uma comparação pareada [15].

- **Viés de Verbosidade:** A propensão a atribuir pontuações mais altas a respostas mais longas, independentemente da qualidade do conteúdo [15].
- **Viés de Auto-aprimoramento:** A tendência de um LLM-juiz favorecer respostas geradas por si mesmo ou por modelos com arquitetura similar [15, 29].

Adicionalmente, os LLMs podem apresentar alucinações ou limitações em domínios de raciocínio muito específicos, o que pode levar a julgamentos incorretos [15, 29]. Para mitigar esses riscos, é necessário adotar estratégias como a randomização da ordem das respostas, o uso de prompts de avaliação claros e baseados em critérios bem definidos, como os da norma ISO/IEC/IEEE 29148:2018 [5] utilizados neste trabalho, e a seleção de um LLM-juiz arquitetonicamente distinto dos modelos avaliados. A supervisão humana, mesmo que em uma amostra, continua sendo indispensável para validar e calibrar os resultados da avaliação automatizada [29].

2.2 Trabalhos Relacionados

A aplicação de LLMs tem se expandido rapidamente na Engenharia de Software, com a ER emergindo como um campo de grande potencial. A capacidade desses modelos de compreender e gerar linguagem natural oferece soluções promissoras para automatizar tarefas tradicionalmente manuais, trabalhosas e propensas a ambiguidades [7]. Uma revisão sistemática da literatura recente confirma o crescente interesse na área, destacando que atividades como especificação, elicitação e análise de requisitos são as mais exploradas, enquanto a priorização e a rastreabilidade ainda carecem de atenção [30]. Neste contexto, a literatura tem abordado o uso de LLMs em ER a partir de diversas perspectivas, que podem ser agrupadas em automação de tarefas fundamentais, exploração da elicitação, desenvolvimento de frameworks integrados e o papel crucial do prompt engineering.

Uma vertente significativa da pesquisa foca em validar a capacidade dos LLMs para automatizar tarefas centrais de ER. No que tange à geração de artefatos, Krishna et al. [13] demonstraram empiricamente que LLMs como GPT-4 e CodeLlama podem gerar rascunhos de Documentos de Especificação de Requisitos de Software (*Software Requirement Specifications* - SRS) com qualidade comparável à de um engenheiro júnior, resultando em uma economia de tempo substancial. De forma complementar, Almonte et al. [14] investigaram a geração de Requisitos Não-Funcionais a partir de Requisitos Funcionais, concluindo, através de uma avaliação com múltiplos especialistas, que os Requisitos Não-Funcionais gerados por LLMs possuem alta validade e aplicabilidade. Outra tarefa fundamental é a classificação de requisitos. Nesse âmbito, Alhoshan et al. [8] exploraram a abordagem de *Zero-Shot Learning* (ZSL), mostrando que LLMs podem classificar

requisitos (por exemplo, funcional vs. não funcional) com desempenho aceitável sem a necessidade de grandes volumes de dados rotulados, o que representa uma alternativa escalável aos métodos supervisionados tradicionais. Esses estudos estabelecem coletivamente que LLMs são ferramentas viáveis para a produção e organização de artefatos de requisitos.

Outra área de investigação se concentra na fase inicial e interativa da ER: a elicitação de requisitos. Diversos trabalhos comparam o desempenho de LLMs com o de especialistas humanos. Hymel et al. [31] conduziram um estudo comparativo no qual os requisitos gerados por um LLM foram considerados mais alinhados e completos pelos *stakeholders* do que aqueles produzidos por especialistas humanos em uma sessão de tempo limitado, além de serem drasticamente mais rápidos e baratos de obter. De maneira similar, Ronanki et al. [32] avaliaram a qualidade de requisitos para sistemas de IA confiáveis e descobriram que os requisitos gerados pelo ChatGPT superaram os formulados por humanos em atributos como consistência e correção, apesar de não atenderem às expectativas em termos de viabilidade e clareza. Indo além da simples geração, Ataei et al. [33] propuseram o *Elicitron*, um *framework* que utiliza agentes de IA simulados para descobrir necessidades dos usuários, demonstrando um potencial para explorar casos de uso não previstos que seriam difíceis de capturar em entrevistas convencionais.

Visando uma automação mais completa, alguns pesquisadores têm proposto *frameworks* que orquestram LLMs para cobrir múltiplas fases da ER. Um exemplo proeminente é o MARE, de Jin et al. [34], um *framework* de colaboração multiagente que simula uma equipe de ER, dividindo tarefas como elicitação, modelagem, verificação e especificação entre diferentes agentes de LLM. Seus resultados indicam que essa abordagem colaborativa supera o desempenho de um único LLM. Em paralelo à geração, a avaliação da qualidade dos requisitos também tem sido automatizada. Lubus et al. [35] demonstraram que um LLM pode avaliar características de qualidade de requisitos de acordo com a norma ISO/IEC/IEEE 29148:2018 [5], identificar falhas, fornecer explicações plausíveis e sugerir melhorias. Este trabalho é particularmente relevante, pois introduz a ideia de usar um LLM não apenas como um gerador, mas também como um avaliador, um conceito alinhado à metodologia *LLM-as-a-Judge*.

Um tema transversal e crítico em toda a literatura é a importância do *prompt engineering* para otimizar o desempenho dos LLMs. A forma como as instruções são fornecidas ao modelo impacta diretamente a qualidade da saída. Ronanki et al. [10] abordaram essa questão de forma sistemática, avaliando a eficácia de cinco padrões de prompt em tarefas de classificação e rastreabilidade de requisitos. Eles concluíram que diferentes padrões são mais adequados para tarefas distintas e que a escolha do prompt é fundamental para a confiabilidade dos resultados. Este estudo estabelece um precedente metodológico im-

portante ao focar não apenas em *se* LLMs podem realizar tarefas de ER, mas em *como* otimizar sua performance através de estratégias de prompt.

2.2.1 Lacuna de Pesquisa e a Contribuição deste Trabalho

A literatura existente estabelece de forma robusta que os LLMs são capazes de executar diversas tarefas de ER, desde a classificação [8] e geração de especificações completas [13], até a elicitação de necessidades latentes [33]. Ficou claro também que o prompt engineering é um fator determinante para o sucesso [10] e que a avaliação automatizada da qualidade é uma área emergente e promissora [35].

Contudo, algumas lacunas persistem. Primeiramente, um diferencial crucial, que define o escopo do nosso trabalho, é o foco no ecossistema de Software de Código Aberto. Conforme detalhado na Subseção 2.1.1, o processo de ER em projetos OSS é caracterizado por um grande volume de entradas informais, não estruturadas e concisas provenientes de *issue trackers*. Enquanto os trabalhos revisados utilizam descrições de projetos controlados [13, 31] ou datasets de benchmark [8, 10], eles não enfrentam diretamente o desafio de processar a matéria-prima da elicitação em OSS. Nossa pesquisa ataca esse problema central: a transformação de títulos de *issues* em requisitos bem escritos, um cenário prático e pouco explorado.

Além disso, enquanto o trabalho de Ronanki et al. [10] avaliou padrões de prompt, falta um estudo empírico comparativo em larga escala que meça como estratégias de prompt distintas influenciam atributos de qualidade específicos e padronizados. O efeito interativo entre a escolha do LLM e a estratégia de prompting também permanece uma área pouco explorada, assim como a validação da abordagem *LLM-as-a-Judge* para a avaliação escalável de artefatos de requisitos neste contexto.

Este trabalho se posiciona diretamente para preencher essas lacunas. Realizamos uma avaliação empírica sistemática no contexto específico de OSS, transformando títulos de *issues* em requisitos de software. Comparamos explicitamente três estratégias de prompt em dois LLMs distintos e medimos seu impacto em três atributos de qualidade da norma ISO/IEC/IEEE 29148:2018 [5]. Ao fazer isso, não apenas demonstramos a eficácia dos LLMs, mas também revelamos os *trade-offs* dependentes do modelo e do prompt entre esses atributos, validando a metodologia *LLM-as-a-Judge* como uma abordagem escalável e consistente para a avaliação de qualidade em larga escala neste domínio. Dessa forma, nossa pesquisa avança a literatura de uma demonstração de potencial para o fornecimento de evidências documentadas sobre desempenho, efeitos colaterais e considerações práticas para a aplicação de LLMs na geração de requisitos.

2.3 Síntese do Capítulo

Este capítulo estabeleceu a fundamentação teórica que sustenta a pesquisa. Iniciou-se com a definição da Engenharia de Requisitos, detalhando suas atividades, desafios e as particularidades do processo em ambientes de Software de Código Aberto, onde a elicitação informal via *issue trackers* cria uma oportunidade para a automação. Em seguida, foram apresentados os critérios de qualidade para requisitos de software, com base na norma ISO/IEC/IEEE 29148:2018 [5], justificando a escolha das métricas de Não Ambiguidade, Verificabilidade e Singularidade para o escopo deste trabalho.

A discussão prosseguiu com a introdução aos LLMs, abordando a arquitetura *Transformer*, suas capacidades e limitações. Foi dado destaque à Engenharia de Prompts como uma disciplina crucial para otimizar a interação com esses modelos, descrevendo estratégias fundamentais. Subsequentemente, foi apresentado o paradigma *LLM-as-a-Judge* como uma solução escalável para a avaliação da qualidade de artefatos gerados, detalhando seus benefícios e vieses potenciais.

Por fim, uma análise dos trabalhos relacionados na Tabela 2.1 revela o panorama atual da aplicação de LLMs em ER, identificando a lacuna de pesquisa que esta dissertação se propõe a preencher: a avaliação empírica e comparativa de estratégias de prompt e de diferentes LLMs na tarefa de gerar requisitos a partir de entradas concisas no contexto específico de projetos OSS, utilizando uma metodologia de avaliação automatizada e escalável.

Tabela 2.1: Comparativo dos Trabalhos Relacionados

Referência	**Tecnologia utilizada**	**Etapas de ER abordadas**	**Base de dados**
Alhoshan et al. [8]	Sentence-BERT, ZSL	Classificação	PROMISE NFR, SecReq
Krishna et al. [13]	GPT-4, CodeLlama	Geração, Validação e Retificação de SRS	Benchmark humano (portal universitário)
Almonte et al. [14]	Múltiplos LLMs (GPT, Claude, Gemini, Llama)	Geração de NFRs, Validação	FR NFR Dataset (derivado do PURE)
Hymel et al. [31]	GPT-4	Elicitação (geração inicial de épicos/histórias)	Entradas de stakeholders, especialistas humanos
Ronanki et al. [32]	ChatGPT	Elicitação de requisitos	Especialistas humanos
Ataei et al. [33]	Multi-agentes LLM (GPT-4-Turbo)	Elicitação de necessidades latentes	Entrevistas humanas (para comparação)
Jin et al. [34]	Multi-agentes LLM (GPT-3.5)	Elicitação, Modelagem, Verificação, Especificação	Datasets públicos e casos de avaliação próprios
Lubus et al. [35]	LLaMA 2 como avaliador	Qualidade, Validação, Melhoria	Projeto hipotético e dataset PURE
Ronanki et al. [10]	GPT-3.5 turbo, Padrões de Prompt	Classificação e Rastreabilidade de requisitos	PROMISE NFR, PURE
Trabalho Proposto	**LLMs (o3-mini, DeepSeek R1), LLM-as-a-Judge (Qwen)**	**Geração de requisitos, Avaliação de Qualidade**	**Títulos de issues de repositórios OSS (GitHub)**

Capítulo 3

Configuração do Estudo

Este capítulo detalha a configuração metodológica adotada para responder às questões de pesquisa desta dissertação. Assumindo os conceitos apresentados no Capítulo 2, o foco aqui reside em descrever os procedimentos técnicos e as decisões operacionais tomadas em cada fase do estudo. O objetivo é fornecer uma descrição transparente e sistemática do processo, garantindo a validade, a replicabilidade e a confiabilidade dos resultados.

A pesquisa foi estruturada em três estágios principais, conforme ilustrado na Figura 3.1: (1) coleta de dados, (2) geração de requisitos e (3) avaliação de requisitos. Cada etapa foi executada de forma rigorosa, desde a extração de solicitações de funcionalidades (*feature requests*) de repositórios de software de código aberto até a análise quantitativa e qualitativa dos requisitos gerados por LLMs.

3.1 Desenho da Pesquisa

O presente estudo emprega um desenho de pesquisa empírico e quantitativo para avaliar a eficácia de técnicas de engenharia de prompts na geração de requisitos de software por LLMs. A metodologia foi concebida para examinar como diferentes estratégias de prompt e a escolha do LLM influenciam a qualidade dos artefatos de requisitos gerados a partir de solicitações de funcionalidades extraídas de projetos de software de código aberto.

A Figura 3.1 apresenta uma visão geral do fluxo metodológico, que se desdobra nas três fases sequenciais que serão detalhadas nas seções subsequentes deste capítulo.

3.2 Coleta de Dados

A primeira etapa da metodologia, conforme indicado na Figura 3.1, consistiu na construção de um conjunto de dados abrangente e relevante para a tarefa de geração de requisitos. Para isso, realizou-se a extração de solicitações de funcionalidades dos cinco projetos de

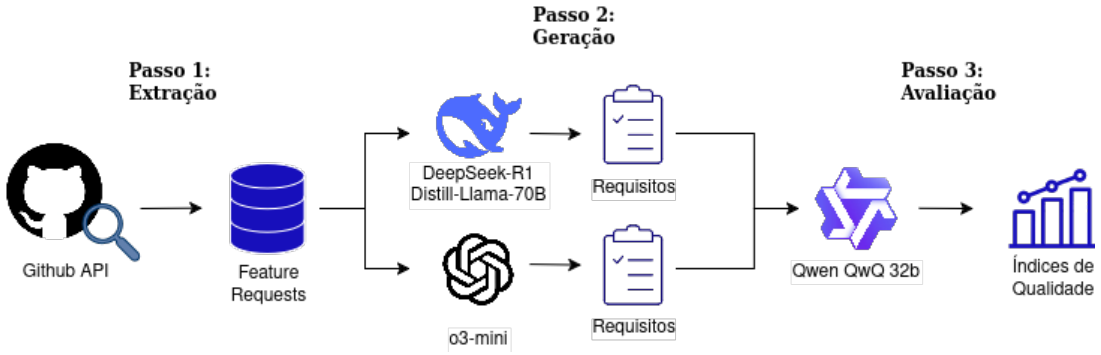

Figura 3.1: Procedimentos Metodológicos

código aberto com maior número de *issues* no GitHub: Pytorch, Flutter, Godot Engine[1], Rust e Golang[2]. A escolha desses repositórios foi baseada em seu elevado nível de atividade e relevância para a comunidade de OSS.

O processo de coleta de dados seguiu os seguintes passos:

1. **Critérios de Seleção das *issues*:** Para garantir a inclusão de solicitações de funcionalidades genuínas e com alto engajamento da comunidade, foram selecionadas as 30 *issues* abertas mais ativas[3] de cada repositório. Esta abordagem foi necessária devido à inexistência de um método programático consistente para identificar *issues* que foram fechadas por motivos de classificação incorreta, spam ou duplicação entre diferentes repositórios. As *issues* foram filtradas para incluir apenas aquelas com o rótulo de *feature request* (solicitação de funcionalidades) ou equivalente. Este processo de filtragem foi fundamental para garantir que o conjunto de dados permanecesse focado em conteúdo prático e relacionado a requisitos, eliminando discussões irrelevantes ou tarefas menos significativas.

2. **Extração e Formatação dos Dados:** Utilizando a API do GitHub, os títulos e os rótulos das *issues* foram extraídos e estruturados em um conjunto de dados padronizado. Cada registro continha metadados (ex: nome do repositório, ID da *issue*), juntamente com o conteúdo textual necessário para a geração de requisitos. Este passo visou facilitar a consistência em todo o conjunto de dados, o que foi fundamental para a etapa de engenharia de prompts e para a análise subsequente, além de garantir a reprodutibilidade dos experimentos.

[1]O projeto Godot Engine possui um repositório específico para solicitações de funcionalidades, denominado *godot-proposals*.

[2]Dados coletados em 22 de março de 2025, a partir do repositório `https://github.com/EvanLi/Github-Ranking`

[3]Ordenadas por `reactions` na API do GitHub: `https://docs.github.com/en/rest/reactions/reactions?apiVersion=2022-11-28`

Dessa forma, o conjunto de dados final foi composto por 150 solicitações de funcionalidades, coletadas igualmente dos cinco repositórios mencionados. Para a fase subsequente de geração de requisitos, o título da *issue* extraído de cada solicitação serviu como dado de entrada principal. Esta coleção curada apresenta uma gama diversificada de entradas para os LLMs, abrangendo solicitações de domínios de software variados (como aprendizado de máquina, *frameworks* de interface de usuário, desenvolvimento de jogos e programação de sistemas) e exibindo variações naturais na formulação, especificidade técnica e comprimento do título. Um excerto que ilustra a estrutura dos dados coletados é apresentado na Tabela 3.1.

Tabela 3.1: Amostra do Conjunto de Dados de Solicitações de Funcionalidades

Repositório	**Número da issue**	**Título da issue**
pytorch/pytorch	16897	Implement `numpy.random.choice` equivalent
flutter/flutter	46789	Improve the indexability (SEO) of Flutter apps on the web
godotengine/godot-proposals	6416	Add a Trait system for `GDScript`
rust-lang/rust	39915	Linking with LLD
golang/go	32204	`net/http`: support HTTP/3

3.3 Geração de Requisitos

Na segunda etapa da metodologia, foram preparados os prompts e executados dois LLMs distintos para gerar os requisitos a partir do conjunto de dados construído na fase anterior. Motivados por estudos prévios em engenharia de prompts, como os discutidos na Seção 2.1.4, os prompts foram elaborados com base em algumas das técnicas mais conhecidas, com o objetivo de avaliar seu papel na melhoria da qualidade das respostas dos LLMs.

3.3.1 Configuração dos Prompts

Foram desenvolvidos três prompts distintos, cada um baseado em uma conhecida técnica de engenharia de prompts, para investigar como diferentes abordagens afetam a qualidade e a clareza dos requisitos produzidos: *Zero-shot*, *Few-shot* e Expert Identity. Cada método representa uma estratégia diferente na forma como o modelo é preparado para interpretar e responder à entrada.

Zero-shot Este método, baseado na abordagem de *zero-shot prompting* (descrita na Seção 2.1.4), consistiu em fornecer diretamente os títulos das *issues* ao modelo,

sem detalhes ou exemplos adicionais. A estratégia baseia-se no conhecimento geral do modelo para executar a tarefa. Conforme explorado por Alhoshan et al. [8], esta técnica mostrou-se eficaz em tarefas de classificação de requisitos sem dados de treinamento rotulados. Neste trabalho, ela foi utilizada para testar o conhecimento inerente e a adaptabilidade do modelo sem contexto adicional.

Prompt - *Zero-shot*

You are given a feature request from an open-source software project's issue tracker. Generate a clear, concise, and self-contained software requirement statement that captures the intended feature.
Output only the finalized requirement statement. Do not include explanations or commentary.

Few-shot Nesta abordagem (descrita na Seção 2.1.4), alguns exemplos de requisitos previamente gerados foram incluídos no prompt para guiar o modelo. Essa técnica ajuda o modelo a inferir os requisitos da tarefa a partir de um número mínimo de exemplos, melhorando sua precisão e relevância [24, 10]. O objetivo deste prompt foi fornecer uma orientação mínima ao modelo para obter uma maior precisão com base em padrões de exemplos anteriores.

Prompt - Few-Shot

Here are examples of feature requests from an open-source software project's issue tracker along with their corresponding software requirements.
Based on these examples, analyze new feature requests and generate the requirement.
Output only the finalized requirement statement. Do not include explanations or commentary.

Feature request: Limit GPU memory usage during prediction.
Requirement: The system shall provide an option to limit GPU memory usage during prediction.

Feature request: Reset password
Requirement: The system must provide a functional "forgot password" link that allows users to reset their passwords securely.

Feature request: Make the application use vector search.
Requirement: The system shall use vector search to improve search performance.

Expert Identity A técnica de *Expert Prompting* (apresentada na Seção 2.1.4) instrui os LLMs a agirem como especialistas distintos para resolver problemas complexos, fornecendo uma Expert Identity e instruções específicas para seu domínio. Isso tende a levar a um melhor desempenho e a respostas de maior qualidade, especialmente em tarefas que exigem uma compreensão aprofundada [25].

Prompt - Expert Identity

> You are a Software Requirements Engineer with deep expertise in analyzing feature requests in open-source software projects. Your task is to generate a clear, complete, and actionable software requirement from informal feature requests found in OSS issue trackers.
> Output only the finalized requirement statement. Do not include explanations or commentary.

3.3.2 Execução dos Prompts

A geração dos requisitos foi realizada utilizando os modelos de raciocínio `OpenAI o3-mini`[4] e `DeepSeek-R1-Distill-Llama-70B`[5] [36]. Os modelos foram selecionados com base em seu desempenho médio em múltiplos *benchmarks*. O `OpenAI o3-mini` foi escolhido por sua superior relação custo-benefício entre os modelos de raciocínio de código fechado: seu custo é inferior a 10% do `OpenAI o1`, o modelo de melhor desempenho, enquanto entrega resultados comparáveis. Da mesma forma, o `DeepSeek-R1-Distill-Llama-70B` foi selecionado como o modelo de raciocínio de código aberto com a melhor relação custo-benefício, com base no tamanho de suas variantes destiladas e em métricas de desempenho de *benchmarks*[6] [37].

Para garantir um grau de reprodutibilidade, permitindo ao mesmo tempo uma diversidade na formulação das saídas, os LLMs foram configurados com parâmetros específicos. A semente aleatória (*random seed*) foi fixada em 42 para todas as tarefas de geração. Foi empregada uma temperatura de 1.0; embora temperaturas mais altas aumentem a aleatoriedade, este valor foi escolhido para encorajar os modelos a explorarem uma gama um pouco mais ampla de formulações de requisitos potenciais, com base nas diferentes solicitações de entrada e estilos de prompt, em vez de produzirem saídas excessivamente determinísticas. É importante notar que as respostas de LLMs podem exibir comportamento não determinístico mesmo com uma semente fixa, devido a fatores como otimizações subjacentes do modelo ou atualizações da API ao longo do tempo. No escopo e

[4] `https://openai.com/index/openai-o3-mini/`
[5] `https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Llama-70B`
[6] Métricas coletadas em 22 de março de 2025

cronograma deste estudo, execuções repetidas do pipeline de geração sob configurações idênticas produziram saídas altamente consistentes, demonstrando a reprodutibilidade prática de nossos resultados.

Ambos os LLMs foram avaliados utilizando todas as variantes de prompt como *prompts de sistema* e os títulos das *issues* como *prompts de usuário*. Cada variante de prompt foi pareada com cada título de *issue* e submetida a ambos os modelos, resultando em três requisitos distintos por *issue*. Este processo gerou um total de 450 requisitos por LLM, somando 900 requisitos no geral.

3.4 Avaliação dos Requisitos

3.4.1 Métricas de Avaliação

No campo da engenharia de sistemas e software, requisitos bem formulados são cruciais para o sucesso dos projetos. Conforme a norma ISO/IEC/IEEE 29148:2018 [5], os requisitos individuais devem ser: (1) necessários; (2) apropriados; (3) não ambíguos; (4) completos; (5) singulares; (6) viáveis; (7) verificáveis; (8) corretos; e (9) conformes. Uma análise mais detalhada da descrição dessas características leva à conclusão de que a **Não Ambiguidade**, a **Singularidade** e a **Verificabilidade** podem ser mais facilmente avaliadas ao se analisar requisitos de forma isolada, especialmente no contexto de OSS, onde detalhes específicos do projeto, como partes interessadas, prazos e restrições, são frequentemente ausentes. As demais características necessitam de uma compreensão mais aprofundada do contexto e das restrições específicas do projeto.

A seleção dessas três métricas é útil para avaliar os requisitos gerados e sua conformidade com a norma, pois elas representam qualidades intrínsecas de requisitos individuais e podem ser avaliadas sem o contexto completo do projeto. Cada métrica foi avaliada utilizando uma escala Likert de 1 a 5, onde 1 representa baixa qualidade e 5, alta qualidade. Para garantir uma métrica padronizada de comparação, foi adotada a escala de avaliação presente em outros estudos [13, 14].

3.4.2 Avaliação Automatizada

Para avaliar o grande volume de requisitos de software gerados neste estudo, foi adotado o paradigma *LLM-as-a-Judge*, discutido na Seção 2.1.5, utilizando o modelo `Qwen QwQ-32b` [38, 39] como avaliador. Esta metodologia oferece uma alternativa escalável e de baixo custo à avaliação humana extensiva, que seria impraticável para o grande número de saídas geradas [15].

Empregar um LLM como juiz neste contexto oferece vantagens distintas. Primeiro, promove uma consistência superior nas 900 avaliações, pois o LLM não é suscetível a fatores como fadiga ou variabilidade subjetiva que podem afetar avaliadores humanos [29]. Segundo, a abordagem oferece explicabilidade. Conforme destacado na literatura, uma vantagem chave é a capacidade de configurar o juiz para fornecer justificativas para suas avaliações [15]. Nossa implementação foi projetada para explorar essa capacidade, operacionalizando a avaliação com base nos critérios da norma ISO/IEC/IEEE 29148:2018 [5]. O juiz foi instruído a retornar uma resposta estruturada, contendo não apenas uma pontuação para cada atributo, mas também uma explicação textual. Essa justificativa é essencial para nossa análise, fornecendo os dados qualitativos necessários para interpretar os escores quantitativos.

Embora vantajosa, é essencial reconhecer e mitigar as limitações potenciais da abordagem. Juízes LLM podem exibir vieses inerentes, como o de auto-aprimoramento (favorecer saídas de modelos com arquitetura similar) [29, 15]. Para mitigar esse risco, selecionamos deliberadamente um avaliador (`Qwen QwQ-32b`) que é arquitetonicamente distinto dos LLMs testados (`OpenAI o3-mini` e `DeepSeek R1`). Adicionalmente, ao fornecer ao juiz definições claras e baseadas na norma em seu prompt, buscamos restringir seu processo de tomada de decisão e aprimorar a precisão avaliativa.

3.4.3 Prompt de Avaliação

Para guiar o LLM em seu papel de avaliador, foi elaborado um prompt específico que descreve a tarefa, os critérios e o formato de saída desejado. O prompt instrui o LLM a avaliar um requisito de software com base nos atributos de qualidade selecionados da norma ISO/IEC/IEEE 29148:2018 [5]: Não Ambiguidade, Verificabilidade e Singularidade. Para garantir a coleta sistemática de dados e facilitar a análise automatizada, o prompt instruiu explicitamente o LLM a estruturar sua resposta como uma instância *JSON* em conformidade com um esquema específico. Esse esquema inclui campos para as pontuações numéricas de cada atributo e um campo dedicado para a "explicação" textual do avaliador. O prompt completo fornecido ao LLM avaliador foi o seguinte:

Prompt - Avaliador LLM

As an experienced requirements engineer, your task is to evaluate the quality of a given software requirement based on the following criteria:

Unambiguity: The requirement is stated in such a way so that it can be interpreted in only one way. The requirement is stated simply and is easy to understand. Score

1 if the requirement is highly ambiguous and 5 if there is no room for multiple interpretations.

Verifiability: The requirement is structured and worded such that its realization can be proven (verified) to the customer's satisfaction at the level the requirements exists. Verifiability is enhanced when the requirement is measurable. Score 1 if the requirement is difficult to verify, and 5 if it is easily measurable.

Singularity: The requirement states a single capability, characteristic, constraint or quality factor. Score 1 if the requirement includes several needs or is unclear in its focus, and 5 if it is entirely singular in nature.

For each criterion, provide a score from 1 to 5, with 1 being the lowest (poor) and 5 being the highest (excellent). Structure your output as a JSON object containing the scores and a combined explanation for your ratings.

O esquema JSON que o modelo foi instruído a seguir está detalhado abaixo:

```
{
  "properties": {
    "unambiguity": {
      "description": "1-5 score for unambiguity",
      "title": "Unambiguity",
      "type": "integer"
    },
    "verifiability": {
      "description": "1-5 score for verifiability",
      "title": "Verifiability",
      "type": "integer"
    },
    "singularity": {
      "description": "1-5 score for singularity",
      "title": "Singularity",
      "type": "integer"
    },
    "explanation": {
      "description": "Combined textual explanation for the scores",
      "title": "Explanation",
```

```
      "type": "string"
    }
  },
  "required": [
    "unambiguity",
    "verifiability",
    "singularity",
    "explanation"
  ]
}
```

3.5 Estratégia de Análise dos Dados

Para abordar as questões de pesquisa, foi utilizada uma abordagem de métodos mistos para examinar a coleção de 900 requisitos gerados, cada um anotado com as pontuações da escala Likert para *Não Ambiguidade*, *Verificabilidade* e *Singularidade*, juntamente com a justificativa qualitativa do LLM juíz.

Quantitativamente, serão calculadas estatísticas descritivas (médias, medianas, desvios padrão e distribuições de frequência) e visualizadas as distribuições das pontuações, tanto de forma geral quanto dentro dos grupos experimentais definidos pelo LLM, estilo de prompt e repositório. Como os dados são ordinais e podem violar os pressupostos de normalidade, serão aplicados testes não paramétricos: o teste U de Mann-Whitney [40] para comparações entre pares e o teste de Kruskal-Wallis [41], seguido pelo procedimento post-hoc de Dunn [42] quando apropriado (ex: entre os estilos de prompt). O coeficiente de correlação de Spearman [43] será calculado para avaliar as associações entre as três métricas de qualidade.

Qualitativamente, será realizada uma análise temática do feedback narrativo do juiz, registrado no campo `explanation`, suplementada com amostras dos requisitos gerados, para descobrir pontos fortes, fracos e padrões recorrentes ligados a combinações específicas de LLM e prompt. A integração dessas percepções qualitativas com os resultados quantitativos fornecerá uma explicação mais rica dos fatores que influenciam a qualidade dos requisitos.

3.6 Síntese do Capítulo

Neste capítulo, foi detalhado o desenho metodológico da pesquisa. Iniciou-se com a descrição da arquitetura geral do estudo, fundamentada em três estágios: coleta, geração e avaliação. A etapa de coleta de dados foi descrita, especificando a seleção de cinco repositórios de OSS de alta atividade e o critério de extração de 150 títulos de *issues* que serviram como matéria-prima. Em seguida, o processo de geração de requisitos foi pormenorizado, incluindo a configuração de três estratégias de prompt (*Zero-shot*, *Few-shot*, Expert Identity) e a execução em dois LLMs distintos (`OpenAI o3-mini` e `DeepSeek R1`) com parâmetros controlados. A metodologia de avaliação foi apresentada, justificando a escolha das métricas da norma ISO/IEC/IEEE 29148:2018 e detalhando a implementação da abordagem *LLM-as-a-Judge*, incluindo o prompt de avaliação e o esquema de saída estruturada. Por fim, foi delineada a estratégia de análise de dados de métodos mistos, combinando testes estatísticos não paramétricos com análise temática para uma interpretação robusta dos resultados.

Capítulo 4

Resultados e Discussão

Este capítulo apresenta e discute os resultados obtidos a partir da avaliação de LLMs para a geração automática de requisitos de software a partir de títulos de *issues* do GitHub. O objetivo é responder às questões de pesquisa que nortearam este estudo, fornecendo uma análise aprofundada tanto quantitativa quanto qualitativa dos dados coletados.

Para responder a estas questões, foi empregada a metodologia empírica detalhada no Capítulo 3. O processo envolveu a coleta de dados de repositórios de código aberto, a geração de 900 requisitos candidatos utilizando dois LLMs e três estratégias de prompt, e a subsequente avaliação da qualidade desses requisitos através do paradigma *LLM-as-a-Judge*.

As questões de pesquisa que direcionam este trabalho são:

- **QP1:** Qual a eficácia dos LLMs para gerar requisitos de software a partir de títulos de *issues* de projetos OSS?
- **QP2:** Como diferentes estratégias de engenharia de prompts influenciam a qualidade dos requisitos gerados?
- **QP3:** Quais são as características qualitativas, os pontos fortes e fracos dos requisitos gerados pelos LLMs?

O capítulo está estruturado da seguinte forma: a Seção 4.1 oferece uma visão panorâmica da qualidade geral dos requisitos gerados. As seções subsequentes abordam cada questão de pesquisa individualmente, integrando a apresentação dos resultados com uma discussão aprofundada. A Seção 4.2 avalia a eficácia geral dos LLMs. A Seção 4.3 investiga a influência das estratégias de prompt. A Seção 4.4 explora as características qualitativas dos requisitos através de exemplos ilustrativos. A Seção 4.5 apresenta achados suplementares e suas implicações. Por fim, a Seção 4.7 sintetiza as principais conclusões do capítulo.

4.1 Visão Geral da Qualidade dos Requisitos Gerados

Uma análise inicial de todos os 900 requisitos gerados estabeleceu uma base para a compreensão da qualidade geral alcançada pelos LLMs. As principais estatísticas descritivas para os atributos de *Não Ambiguidade*, *Verificabilidade* e *Singularidade* estão sumarizadas na Tabela 4.1.

De modo geral, os requisitos gerados foram avaliados favoravelmente pelo LLM-juiz, com pontuações médias superiores a 4.25 e medianas de 4 ou 5 em todas as métricas. Este resultado sugere que os LLMs são frequentemente capazes de produzir requisitos percebidos como largamente não ambíguos, verificáveis e singulares, indicando uma aderência geral aos atributos de qualidade definidos pela norma ISO/IEC/IEEE 29148:2018. Este potencial alinha-se com as conclusões de trabalhos como [7, 44].

Tabela 4.1: Estatísticas Gerais para as Métricas de Qualidade dos Requisitos (N=900)

Estatística	**Não Ambiguidade**	**Verificabilidade**	**Singularidade**
Média	4.34	4.25	4.38
Desvio Padrão	0.69	0.84	1.15
Mínimo	2.00	1.00	1.00
Mediana	4.00	4.00	5.00
Máximo	5.00	5.00	5.00

Contudo, a variabilidade notável, especialmente no atributo de **Singularidade** ($DP = 1.15$), e as pontuações mínimas (2.0, 1.0, e 1.0) indicam deficiências ocasionais e uma inconsistência significativa. Este achado corrobora os desafios inerentes à Engenharia de Requisitos, como a dificuldade de se obter qualidade consistente, discutidos no Capítulo 2 e amplificados em ambientes OSS pela informalidade das *issues*, como apontado na Subseção 2.1.1.

A distribuição das pontuações, ilustrada na Figura 4.1, confirma essa observação. Embora as pontuações altas (4 e 5) predominem, existe uma cauda não trivial de pontuações mais baixas, especialmente para *Verificabilidade* e *Singularidade*. Isso sugere que, embora os LLMs sejam promissores, alcançar uma geração de requisitos de alta qualidade de forma consistente continua a ser um desafio, influenciado por fatores que serão explorados nas seções seguintes.

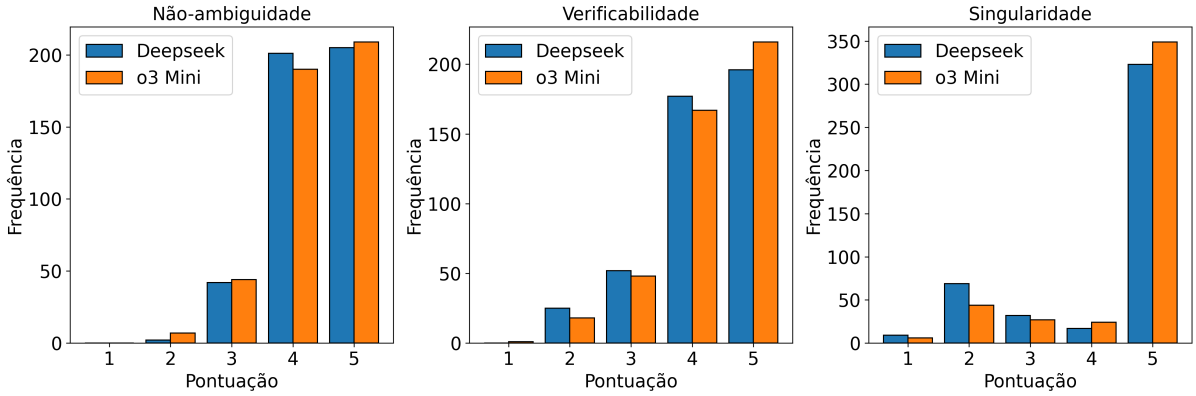

Figura 4.1: Distribuição das Pontuações de Qualidade dos Requisitos por LLM

4.2 QP1: Análise da Eficácia dos Modelos de Linguagem

Nesta seção, abordamos a primeira questão de pesquisa: **QP1: Qual a eficácia dos LLMs para gerar requisitos de software a partir de títulos de *issues* de projetos OSS?**

A comparação da eficácia dos dois LLMs avaliados, `o3-mini` e `DeepSeek`, revelou capacidades comparáveis na geração de requisitos de alta qualidade, como detalhado na Tabela 4.2. Ambos os modelos alcançaram medianas de 4 para *Não Ambiguidade* e *Verificabilidade*, e de 5 para *Singularidade*, reforçando a conclusão geral de que são ferramentas eficazes para esta tarefa.

Tabela 4.2: Estatísticas Descritivas das Métricas de Qualidade por LLM (N=450 por LLM)

	o3-mini			**DeepSeek**		
Métrica	**Média**	**Mediana**	**DP**	**Média**	**Mediana**	**DP**
Não Ambiguidade	4.34	4	0.72	**4.35**	4	0.67
Verificabilidade	**4.29**	4	0.83	4.21	4	0.86
Singularidade	**4.48**	5	1.06	4.28	5	1.23

Apesar das médias similares, a aplicação de testes U de Mann-Whitney revelou uma vantagem estatisticamente significativa para o `o3-mini` na geração de requisitos com maior **Singularidade** ($U = 108009.5$, $p = 0.023$). Na prática, isso sugere que o `o3-mini` demonstrou uma tendência ligeiramente maior a gerar requisitos atômicos, que se concentram em uma única funcionalidade. Para os atributos de *Não Ambiguidade* ($p = 0.967$) e *Verificabilidade* ($p = 0.156$), não foram encontradas diferenças significativas. Os desvios

padrão substanciais observados em ambos os modelos, no entanto, reforçam que a escolha do LLM é apenas um dos fatores que influenciam o resultado final.

Para contextualizar estes achados, é útil compará-los com as conclusões de outros estudos relevantes. Nossos resultados convergem conceitualmente com a literatura em dois pontos principais. Primeiro, os LLMs demonstram uma promessa significativa como assistentes capazes para a elaboração de artefatos de ER. Nossas altas pontuações médias alinham-se com os resultados favoráveis reportados por Krishna et al. [13], onde o CodeLlama produziu saídas comparáveis a um benchmark humano, e por Almonte et al. [14], onde os NFRs gerados alcançaram uma pontuação mediana de validade de 5.0/5.0. Em segundo lugar, e de forma crucial, todos os estudos relatam uma variabilidade significativa e fraquezas notáveis nas saídas geradas. Os desvios padrão substanciais em nossos resultados e a presença de requisitos com baixa pontuação espelham os achados de Krishna et al. [13], que notaram problemas com verbosidade e formatação, e Almonte et al. [14], que descobriram que quase 20% dos NFRs gerados tinham atributos de qualidade incompatíveis. Esta conclusão compartilhada ressalta que, embora os LLMs sejam eficazes, sua saída não é uniformemente perfeita, reforçando a necessidade de avaliação e refinamento cuidadosos.

Sumário da QP1

Os LLMs são geralmente eficazes na geração de requisitos de software a partir de títulos de *issues* de OSS, alcançando altas pontuações médias de qualidade ($> 4.2/5$) para Não Ambiguidade, Verificabilidade e Singularidade. No entanto, essa eficácia apresenta uma variabilidade significativa nas pontuações, indicando que a alta qualidade não é garantida e depende de outros fatores, como a clareza da entrada. Os dois modelos testados tiveram desempenho geral comparável, embora o `o3-mini` tenha apresentado uma vantagem pequena, mas estatisticamente significativa, na produção de requisitos com maior Singularidade.

4.3 QP2: Influência das Estratégias de Engenharia de Prompts

Esta seção investiga a segunda questão de pesquisa: **QP2: Como diferentes estratégias de engenharia de prompts influenciam a qualidade dos requisitos gerados?** A análise confirmou que a engenharia de prompts tem um impacto significativo, embora os efeitos tenham sido notavelmente dependentes do modelo, reforçando as conclusões de

[9, 10] sobre a importância do design do prompt e adicionando nuances sobre as respostas específicas de cada modelo.

4.3.1 Influência dos Prompts no Desempenho do o3-mini

Para o modelo `o3-mini` (Tabela 4.3, Figura 4.2), testes de Kruskal-Wallis indicaram efeitos significativos do prompt sobre a *Não Ambiguidade* ($H = 12.50$, $p = 0.002$) e a *Singularidade* ($H = 30.08$, $p < 0.001$). A estratégia `Few-shot` produziu os melhores resultados para ambos, alcançando a mediana mais alta (5) em todas as métricas e a média mais alta para *Singularidade* e *Não Ambiguidade*. Em contrapartida, o prompt `Expert Identity`, apesar de sua intenção de explorar um conhecimento de domínio simulado, teve um desempenho inferior, degradando particularmente a *Singularidade* (média 4.11). Isso sugere que, para o `o3-mini`, fornecer exemplos concretos (`Few-shot`) foi mais eficaz do que atribuir uma persona.

Tabela 4.3: Estatísticas Descritivas para as Pontuações do `o3-mini` por Estilo de Prompt (N=150 por estilo)

	Zero-Shot			**Few-shot**			**Expert**		
Métrica	**Média**	**Mediana**	**DP**	**Média**	**Mediana**	**DP**	**Média**	**Mediana**	**DP**
Não Ambiguidade	4.37	4	0.70	**4.43**	5	0.76	4.21	4	0.67
Verificabilidade	4.21	4	0.82	4.29	5	0.97	**4.36**	4	0.67
Singularidade	4.56	5	0.97	**4.77**	5	0.75	4.11	5	1.29

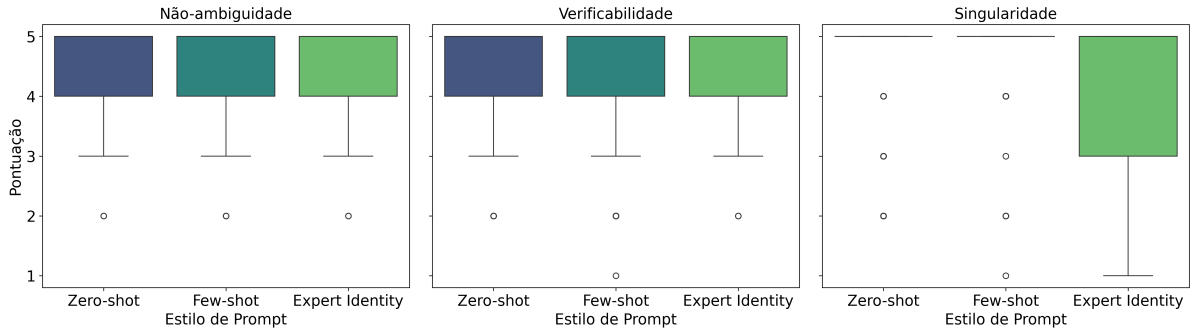

Figura 4.2: Distribuição das Pontuações de Qualidade dos Requisitos do `o3-mini` por Estilo de Prompt

4.3.2 Influência dos Prompts no Desempenho do DeepSeek R1

Para o modelo `DeepSeek R1` (Tabela 4.4, Figura 4.3), efeitos significativos do prompt foram encontrados para *Verificabilidade* ($H = 7.84$, $p = 0.020$) e *Singularidade* ($H =$

29.00, $p < 0.001$). Neste caso, o prompt `Expert Identity` melhorou significativamente a *Verificabilidade* (mediana 5, média 4.37). No entanto, de forma análoga aos resultados do `o3-mini`, o prompt `Expert Identity` levou à menor média de *Singularidade* (3.95) e à maior variabilidade (DP 1.37). Novamente, o prompt `Few-shot` foi o mais eficaz para aprimorar a *Singularidade* (média 4.69).

Este resultado evidencia um balanceamento crítico, particularmente com o prompt `Expert Identity`: tentativas de aumentar o detalhe para melhorar a *Verificabilidade* podem encorajar o modelo a combinar múltiplas facetas, reduzindo a *Singularidade*. Este fenômeno pode ser explicado pela tensão entre a instrução do prompt (atuar como um especialista para gerar um requisito "completo e prático"), e o princípio de atomicidade (Singularidade) da engenharia de requisitos.

Tabela 4.4: Estatísticas Descritivas para as Pontuações do `DeepSeek R1` por Estilo de Prompt (N=150 por estilo)

	Zero-shot			**Few-shot**			**Expert**		
Métrica	**Média**	**Mediana**	**DP**	**Média**	**Mediana**	**DP**	**Média**	**Mediana**	**DP**
Não Ambiguidade	4.31	4	0.65	4.33	4	0.74	4.42	4	0.59
Verificabilidade	4.18	4	0.79	4.07	4	0.98	4.37	5	0.76
Singularidade	4.20	5	1.27	4.69	5	0.86	3.95	5	1.37

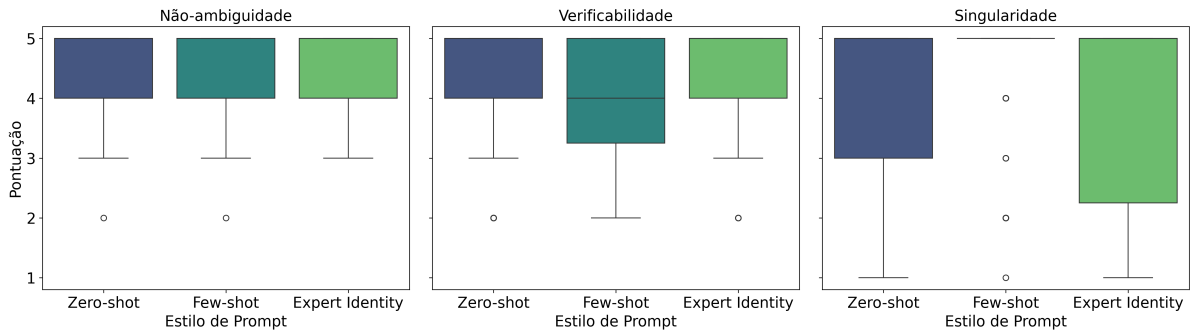

Figura 4.3: Distribuição das Pontuações de Qualidade dos Requisitos do `DeepSeek R1` por Estilo de Prompt

Sumário da QP2

A engenharia de prompts influencia significativamente a qualidade dos requisitos gerados, mas seus efeitos são dependentes do modelo e podem introduzir *trade-offs* críticos entre atributos de qualidade. A estratégia `Few-shot` melhorou de forma consistente e significativa a Singularidade para ambos os LLMs, provando ser eficaz para gerar requisitos atômicos e focados. Em contrapartida, o prompt `Expert Identity` criou uma troca: por vezes, melhorou a Verificabilidade ao adicionar detalhes (notavelmente para o `DeepSeek R1`), mas frequentemente o fez ao custo de degradar a Singularidade para ambos os modelos.

4.4 QP3: Características Qualitativas, Pontos Fortes e Fracos

A análise qualitativa dos requisitos gerados e das explicações do juiz forneceu percepções mais profundas, respondendo à terceira questão de pesquisa: **QP3: Quais são as características qualitativas, os pontos fortes e fracos dos requisitos gerados pelos LLMs?** Os achados qualitativos corroboram amplamente os resultados quantitativos.

Um ponto forte recorrente dos LLMs é a sua capacidade de capturar a intenção central de títulos de *issues* claros e produzir declarações sintaticamente corretas que seguem a estrutura básica de um requisito. No entanto, sua principal fraqueza é uma forte dependência da clareza da entrada, ecoando o princípio "garbage-in, garbage-out" [10] e os desafios da linguagem natural na ER discutidos no Capítulo 2. Títulos vagos ou excessivamente amplos invariavelmente levaram a requisitos com baixa *Verificabilidade*.

A seguir, exemplos extraídos do conjunto de dados ilustram essas observações.

- **Exemplo 1 (Boa Singularidade — `o3-mini`, `Few-shot`):**
  - **Título da issue:** `Add support for nullable static types in GDScript` (godotengine/godot-proposals, 162)
  - **Requisito:** O sistema deve oferecer suporte a tipos estáticos anuláveis no GDScript.
  - **Pontuações (N, V, S):** 5, 5, 5
  - **Justificativa do Juiz:** O requisito especifica claramente o suporte para tipos estáticos anuláveis em GDScript sem ambiguidade. É uma solicitação de funcionalidade singular e pode ser verificada através de testes de implementação.

  - **Observação:** Este caso exemplifica como a combinação de uma entrada clara e uma estratégia de prompt eficaz, como a `Few-shot`, pode produzir um resultado ideal. O prompt guia o modelo para manter o foco, evitando elaborações desnecessárias que comprometeriam a Singularidade.

- **Exemplo 2 (Baixa Singularidade — `DeepSeek R1`, `Expert Identity`):**
  - **Título da issue:** `Implement Terrain` (godotengine/godot-proposals, 6121)
  - **Requisito:** O sistema deve fornecer um sistema de terreno integrado que permita aos usuários criar, modificar e gerenciar paisagens 3D dentro do mecanismo Godot. O sistema de terreno deve oferecer suporte à geração de terreno baseado em mapa de altura, pintura de textura com várias camadas e integração com os sistemas de física, iluminação e material do Godot. Deve incluir recursos para pintura de vegetação, colocação de água e manipulação de nível de detalhe (LOD) para otimização do desempenho. O sistema também deve fornecer uma ferramenta de edição de terreno com uma interface amigável para ajustes de altura, textura e material, e garantir a compatibilidade com os recursos existentes do Godot, como iluminação e sombras dinâmicas. Além disso, o sistema de terreno deve ser projetado para ser eficiente, dimensionável e extensível para acomodar grandes ambientes de mundo aberto e personalizações definidas pelo usuário.
  - **Pontuações (N, V, S):** 3, 3, 2
  - **Justificativa do Juiz:** O requisito especifica um sistema de terreno com recursos claros (gerenciamento de LOD, streaming dinâmico, integração de física, ferramentas de editor, extensibilidade). Entretanto, termos como "eficiente", "dimensionável" e "extensível" introduzem ambiguidade. A verificabilidade é possível para a maioria dos recursos (por exemplo, presença de ferramentas, integração compartilhada), mas termos subjetivos complicam os testes. O requisito se concentra apenas no sistema de terreno, abrangendo seus aspectos técnicos e de ferramentas sem misturar preocupações não relacionadas.
  - **Observação:** Este exemplo é uma ilustração qualitativa clara do *trade-off* identificado na análise quantitativa da Seção 4.3. O prompt `Expert Identity` incentiva o modelo a ser abrangente, resultando em uma declaração detalhada que, embora potencialmente mais informativa, viola diretamente o princípio da Singularidade ao agrupar múltiplas sub-funcionalidades.

- **Exemplo 3 (Requisito vago a partir de título vago — `o3-mini`, `Zero-shot`):**

- **Título da issue:** `Reusing state logic is either too verbose or too difficult` (flutter/flutter, 51752)
- **Requisito:** O sistema deve incluir um mecanismo que simplifica a reutilização da lógica de estado, minimizando a verbosidade e reduzindo a complexidade da implementação.
- **Pontuações (N, V, S):** 3, 2, 5
- **Justificativa do Juiz:** O requisito especifica um mecanismo para simplificar a reutilização da lógica de estado com clareza de propósito, mas usa termos vagos como 'minimizando' e 'reduzindo' sem métricas quantificáveis.
- **Observação:** Este resultado demonstra que o princípio "garbage-in, garbage-out" é amplificado pela estratégia de prompt `Zero-shot`. A falha aqui não está apenas na entrada, mas na incapacidade da estratégia *zero-shot* de guiar o modelo do espaço do problema para o espaço da solução, um passo crucial na especificação de requisitos.

Sumário da QP3

Qualitativamente, um ponto forte chave dos LLMs é sua capacidade de capturar a intenção central de títulos de entrada claros e produzir requisitos sintaticamente corretos. Sua principal fraqueza é uma forte dependência da clareza da entrada; títulos vagos ou excessivamente amplos invariavelmente produzem requisitos que carecem de verificabilidade. Uma característica recorrente é um *trade-off* fundamental entre detalhe e foco; prompts projetados para obter conteúdo abrangente (ex: `Expert Identity`) frequentemente produzem declarações mais descritivas que violam o princípio da singularidade ao agrupar múltiplas capacidades distintas.

4.5 Achados Adicionais e Implicações Metodológicas

Análises suplementares revelaram outros insights relevantes. Foi encontrada uma variação estatisticamente significativa na qualidade dos requisitos para todas as três métricas ($p < 0.001$, via Kruskal-Wallis), o que reforça o papel crítico da qualidade dos dados de entrada. Adicionalmente, a análise de correlação de Spearman (Tabela 4.5) mostrou uma forte relação positiva entre *Não Ambiguidade* e *Verificabilidade* ($\rho = 0.69$). Este resultado é teoricamente esperado, pois, conforme a norma ISO/IEC/IEEE 29148:2018, a clareza de um requisito é um pré-requisito para que sua implementação possa ser verificada objetivamente.

Uma implicação metodológica importante de nossos achados refere-se à validade da abordagem *LLM-as-a-Judge*. Conforme discutido na Seção 3.4, um dos vieses conhecidos

Tabela 4.5: Matriz de Correlação de Spearman para as Métricas de Qualidade ($N = 900$)

Métrica	**Não Ambiguidade**	**Verificabilidade**	**Singularidade**
Não Ambiguidade	1.00	0.69	0.37
Verificabilidade	0.69	1.00	0.30
Singularidade	0.37	0.30	1.00

desta metodologia é o "viés de verbosidade", onde juízes LLM tendem a favorecer respostas mais longas. Curiosamente, nossos resultados indicam que o protocolo de avaliação resistiu a esse viés no que tange à métrica de *Singularidade*. O prompt `Expert Identity` encorajou a geração de requisitos mais verbosos e elaborados, que, por sua vez, receberam pontuações mais baixas de Singularidade. Em vez de recompensar a verbosidade, nosso juiz penalizou corretamente os requisitos menos focados e multifacetados.

Essa robustez provavelmente se deve ao embasamento do prompt de avaliação em critérios explícitos e decompostos, derivados da norma ISO 29148. A instrução clara para avaliar se o requisito declarava uma "única funcionalidade", fornecida no prompt de avaliação, parece ter sido crucial. Isso sugere que fundamentar avaliações *LLM-as-a-Judge* em critérios claros de padrões estabelecidos é uma estratégia vital para mitigar vieses conhecidos e aumentar a validade de construto de avaliações automatizadas em Engenharia de Requisitos.

4.6 Limitações e Ameaças à validade

Para garantir a transparência e o rigor científico, esta seção aborda as limitações inerentes ao estudo e discute as potenciais ameaças à validade dos resultados apresentados. O reconhecimento explícito dessas questões é fundamental para contextualizar as conclusões, orientar a interpretação dos achados e sugerir caminhos para pesquisas futuras que possam superar tais restrições.

4.6.1 Limitações do Estudo

Além das ameaças formais à validade, que serão detalhadas na seção seguinte, é importante reconhecer algumas limitações de escopo que definem as fronteiras desta pesquisa.

Uma primeira limitação reside no escopo dos modelos de linguagem avaliados. O estudo concentrou-se em dois LLMs para geração (`o3-mini` e `DeepSeek R1`) e um para avaliação (`Qwen QwQ-32b`). Embora a seleção tenha sido justificada com base em critérios de desempenho e custo-benefício (conforme detalhado na Seção 3.3.2), o ecossistema de LLMs é vasto e está em constante evolução. Modelos com arquiteturas, dados de treina-

mento ou métodos de ajuste fino distintos poderiam apresentar sensibilidades diferentes às estratégias de prompt e exibir outros padrões de desempenho.

Em segundo lugar, a generalização dos achados é condicionada pela natureza dos dados de entrada. A pesquisa focou-se exclusivamente na geração de requisitos a partir de títulos de *issues* de cinco repositórios de software de código aberto de grande porte. Embora essa escolha represente um cenário prático e desafiador, as conclusões podem não ser diretamente aplicáveis a outros contextos, como projetos comerciais de código fechado, domínios com requisitos de segurança críticos, ou fontes de entrada mais ricas (por exemplo, transcrições de reuniões ou descrições completas de *issues*).

Finalmente, a própria metodologia de avaliação, baseada no paradigma *LLM-as-a-Judge*, constitui uma limitação importante. Embora justificada pela escalabilidade e consistência, e tendo seus viéses conhecidos mitigados por estratégias como a seleção de um juiz arquitetonicamente distinto, a avaliação automatizada permanece uma aproximação da avaliação por especialistas humanos. A subjetividade e a profundidade do julgamento humano não podem ser totalmente replicadas, e os vieses inerentes aos LLMs, mesmo que controlados, ainda representam uma restrição.

4.6.2 Ameaças à Validade

Para uma análise mais sistemática, as ameaças à validade são discutidas a seguir, seguindo as categorias estabelecidas por Wohlin et al. [45].

Ameaças à Validade de Construto

A validade de construto refere-se à correspondência entre os construtos teóricos da pesquisa e suas medições operacionais.

- **Medição da Qualidade dos Requisitos:** A qualidade foi medida utilizando os atributos de *Não Ambiguidade*, *Verificabilidade* e *Singularidade*, derivados da norma ISO/IEC/IEEE 29148:2018. Conforme justificado na Seção 3.4.1, essa seleção se baseou na sua avaliabilidade isolada para dados de OSS. Contudo, esses atributos representam apenas um subconjunto das características definidas pela norma (omitindo, por exemplo, Viabilidade, Completude e Correção). Portanto, nossos achados refletem a qualidade primordialmente sob essa ótica específica.
- **LLM-as-a-Judge como Proxy de Qualidade:** A avaliação dependeu de um LLM (`Qwen QwQ-32b`) como juiz. Embora essa abordagem ofereça escalabilidade e consistência, e trabalhos anteriores sugiram alta concordância com humanos [15, 29], a interpretação dos critérios de qualidade pelo juiz LLM pode diferir sutilmente da de

especialistas humanos ou exibir vieses inerentes. Mitigamos essa ameaça fornecendo definições claras baseadas na norma e capturando a justificativa do juiz, mas a avaliação permanece uma aproximação da "verdadeira" qualidade.

- **Representação de Solicitações de Funcionalidades por Títulos:** Utilizamos apenas os títulos das *issues* do GitHub como entrada para a geração de requisitos. Os títulos podem não capturar todo o contexto ou as nuances presentes no corpo da *issue* ou nos comentários. Essa simplificação, feita em prol da consistência e viabilidade, significa que nossos resultados refletem a capacidade dos LLMs de gerar requisitos a partir de declarações concisas e potencialmente incompletas, e não de solicitações totalmente detalhadas.

Ameaças à Validade Interna

A validade interna aborda fatores que poderiam ter influenciado os resultados observados além das manipulações experimentais.

- **Efeito de Confusão da Qualidade da Entrada:** A clareza, complexidade e especificidade de domínio inerentes aos títulos das *issues* variaram significativamente, influenciando a qualidade dos requisitos gerados independentemente do LLM ou do prompt utilizado. Embora tenhamos analisado o efeito do repositório como um proxy, controlar precisamente a qualidade da entrada em todos os 150 títulos diversos foi inviável.
- **Não Determinismo dos LLMs:** Embora tenhamos utilizado sementes (*seeds*) e configurações de temperatura fixas, o não determinismo inerente às APIs dos LLMs ou otimizações subjacentes poderia, potencialmente, introduzir pequenas variações. Observamos reprodutibilidade prática durante nossos experimentos, mas não podemos garantir saídas idênticas em uma replicação exata.

Ameaças à Validade Externa

A validade externa refere-se à generalização de nossos achados para outros contextos.

- **Seleção Limitada de LLMs:** Avaliamos apenas dois LLMs (`o3-mini` e `DeepSeek R1`). Os achados relativos ao desempenho e à sensibilidade aos estilos de prompt podem não se generalizar diretamente para outros modelos, tais como: modelos maiores da OpenAI, `Claude` e variantes do `Llama` com diferentes arquiteturas, dados de treinamento ou ajustes finos.

- **Técnicas de Prompt Específicas:** Testamos três estilos de prompt específicos. A eficácia relativa pode diferir com outras técnicas como *Chain-of-Thought*, templates complexos ou geração aumentada por recuperação de informação.
- **Especificidade da Tarefa e da Entrada:** Nosso estudo focou na geração de uma única declaração de requisito a partir de títulos de *issues*. Os resultados podem não ser relevantes para a geração de requisitos a partir de diferentes tipos de entradas, como histórias de usuário, descrições detalhadas e atas de reunião, ou para outras etapas da ER, como classificação, rastreamento ou refinamento.
- **Contexto dos Repositórios OSS:** Os dados originaram-se de cinco repositórios de código aberto específicos e ativos. As práticas para escrever solicitações de funcionalidades e a natureza dos requisitos podem diferir em outros projetos OSS, projetos menores, domínios diferentes (por exemplo, sistemas de segurança crítica) ou ambientes de desenvolvimento comercial de código fechado.

Ameaças à Validade de Conclusão e Confiabilidade

Esta categoria relaciona-se à capacidade de tirar conclusões corretas e à reprodutibilidade do estudo.

- **Confiabilidade do Juiz LLM:** Além das preocupações de validade de construto, a consistência do juiz na aplicação dos critérios de pontuação ao longo dos 900 requisitos pode ser questionada. Efeitos de fadiga são minimizados em comparação com humanos [15], mas a consistência interna do modelo não é garantida. Utilizamos saídas JSON estruturadas para garantir que as classificações fossem capturadas sistematicamente.
- **Poder Estatístico e Testes:** Utilizamos testes não paramétricos apropriados para dados ordinais, conforme descrito na Seção 3.5. Embora o tamanho da amostra (N=900) forneça um poder estatístico razoável, a possibilidade de erros do Tipo II (não detectar pequenos efeitos verdadeiros) existe, como em qualquer análise estatística.
- **Extração e Processamento de Dados:** Erros potenciais no uso da API do GitHub ou nos scripts de processamento poderiam afetar o conjunto de dados. O uso padrão da API e a verificação dos scripts foram empregados para minimizar esse risco.
- **Reprodutibilidade:** Conforme observado na Validade Interna, o não determinismo dos LLMs representa um desafio. Além disso, a rápida evolução dos LLMs significa que as versões específicas dos modelos utilizados (`o3-mini`, `DeepSeek R1`, `Qwen`

`QwQ-32b`) podem ser atualizadas ou descontinuadas, afetando potencialmente futuras tentativas de replicação. Documentamos claramente os modelos e parâmetros utilizados para mitigar essa ameaça.

4.7 Síntese do Capítulo

Este capítulo apresentou e discutiu os resultados da avaliação empírica da geração de requisitos de software por LLMs. Foi demonstrado que, embora os modelos sejam geralmente eficazes, sua performance é fortemente modulada pela qualidade da entrada e pela estratégia de engenharia de prompts.

Em resposta à **QP1**, os resultados indicam que LLMs são ferramentas capazes, mas a qualidade da saída varia significativamente. No que tange à **QP2**, foi revelado que a engenharia de prompts é um fator determinante, com a estratégia `Few-shot` se mostrando robusta para melhorar a Singularidade, enquanto a estratégia `Expert Identity` introduziu um trade-off entre Verificabilidade e Singularidade. Finalmente, a análise qualitativa da **QP3** ilustrou esses padrões com exemplos concretos, destacando a dependência da clareza da entrada e o conflito entre detalhe e foco.

Os achados reforçam a visão de que os LLMs funcionam melhor como assistentes poderosos que exigem projeto de prompt cuidadoso e supervisão humana para refinar as saídas, garantindo Verificabilidade e Singularidade adequadas. Esses resultados formam a base para as conclusões gerais da dissertação, as implicações para a prática da Engenharia de Requisitos e as direções para trabalhos futuros, que serão detalhados no capítulo seguinte.

Capítulo 5

Conclusão

Esta dissertação propôs-se a investigar a aplicação de LLMs na automação de uma tarefa fundamental e desafiadora da ER: a geração de requisitos de software a partir de solicitações de funcionalidades informais. Conforme discutido no Capítulo 2, o contexto dos projetos de OSS intensifica esse desafio, devido ao grande volume e à natureza não estruturada das entradas provenientes de sistemas de rastreamento de *issues*. O objetivo central foi avaliar empiricamente não apenas a eficácia dos LLMs nessa tarefa, mas também como sua performance é modulada por diferentes modelos e estratégias de engenharia de prompts.

Para alcançar esse objetivo, foi implementada a metodologia detalhada no Capítulo 3, que envolveu a coleta de dados de repositórios OSS proeminentes, a geração de 900 requisitos e sua subsequente avaliação automatizada através do paradigma *LLM-as-a-Judge*, com base em atributos de qualidade da norma ISO/IEC/IEEE 29148:2018. Este capítulo final sintetiza os resultados obtidos, discute suas implicações teóricas e práticas, reconhece as limitações do estudo e aponta direções para pesquisas futuras.

5.1 Síntese dos Resultados e Respostas às Questões de Pesquisa

A análise dos dados, apresentada no Capítulo 4, forneceu respostas claras às questões de pesquisa que nortearam este trabalho.

Em resposta à **QP1 (Qual a eficácia dos LLMs para gerar requisitos de software a partir de títulos de *issues* de projetos OSS?)**, o estudo concluiu que os LLMs são, de modo geral, eficazes. As pontuações médias para os atributos de *Não Ambiguidade*, *Verificabilidade* e *Singularidade* foram consistentemente altas (superiores a 4.2 em uma escala de 5), indicando que os modelos são capazes de produzir saídas de quali-

dade. Contudo, essa eficácia é marcada por uma inconsistência significativa, evidenciada pelos altos desvios padrão e pela presença de requisitos com baixa pontuação. Portanto, embora promissores, os LLMs não garantem a geração de requisitos de alta qualidade de forma uniforme, sendo sua performance fortemente dependente de outros fatores.

No que tange à **QP2 (Como diferentes estratégias de engenharia de prompts influenciam a qualidade dos requisitos gerados?)**, os resultados revelaram que a engenharia de prompts é um fator determinante, mas seus efeitos são dependentes do modelo e podem introduzir *trade-offs* entre os atributos de qualidade. A estratégia `Few-shot` destacou-se por melhorar de forma robusta e estatisticamente significativa a *Singularidade* em ambos os modelos. Em contrapartida, a estratégia `Expert Identity`, embora concebida para gerar saídas mais completas, frequentemente degradou a *Singularidade* ao mesmo tempo que, em um dos modelos, melhorou a *Verificabilidade*. Este achado, detalhado na Seção 4.3, expõe uma tensão fundamental entre a geração de requisitos detalhados e o princípio da atomicidade.

Finalmente, para a **QP3 (Quais são as características qualitativas, os pontos fortes e fracos dos requisitos gerados pelos LLMs?)**, a análise qualitativa da Seção 4.4 corroborou os achados quantitativos. Um ponto forte recorrente foi a capacidade dos modelos de produzir declarações sintaticamente corretas que capturam a intenção de entradas claras. A principal fraqueza foi a extrema dependência da qualidade da entrada, aderindo ao princípio "garbage-in, garbage-out", onde títulos vagos invariavelmente levaram a requisitos não verificáveis. A análise qualitativa também ilustrou o *trade-off* entre detalhe e foco, mostrando como prompts que incentivam a completude podem resultar em requisitos que, embora mais ricos em informação, violam a *Singularidade*.

5.2 Implicações do Estudo

Os resultados desta dissertação oferecem contribuições tanto para o campo acadêmico da Engenharia de Software quanto para a prática profissional da Engenharia de Requisitos.

5.2.1 Implicações para a Pesquisa

Este trabalho avança o estado da arte da aplicação de LLMs em ER de várias maneiras. Enquanto estudos anteriores, como os de Krishna et al. [13] e Almonte et al. [14], estabeleceram a viabilidade dos LLMs para gerar artefatos de requisitos, nossa pesquisa fornece uma análise mais granular. Demonstramos empiricamente os *trade-offs* que surgem entre diferentes atributos de qualidade (notavelmente, *Verificabilidade* vs. *Singularidade*) como consequência de estratégias de prompt específicas. Essa descoberta adiciona uma

camada de complexidade à compreensão de como otimizar a geração de requisitos, sugerindo que não existe uma "melhor estratégia" universal, mas sim uma escolha dependente dos objetivos de qualidade priorizados.

Adicionalmente, ao comparar dois LLMs distintos, mostramos que a resposta às estratégias de prompt não é uniforme, reforçando a necessidade de estudos que considerem a interação entre modelo e prompt, uma área pouco explorada por trabalhos como o de Ronanki et al. [10].

Metodologicamente, este estudo contribui para a validação do paradigma *LLM-as-a-Judge* no domínio da ER. Conforme discutido na Seção 4.5, nosso protocolo de avaliação, fundamentado em critérios explícitos da norma ISO/IEC/IEEE 29148:2018, demonstrou ser capaz de mitigar o conhecido "viés de verbosidade" [15]. Ao penalizar corretamente os requisitos mais longos, porém menos singulares, gerados pelo prompt `Expert Identity`, o juiz LLM demonstrou uma aderência aos critérios definidos, o que fortalece a validade de construto de avaliações automatizadas e escaláveis quando estas são rigorosamente fundamentadas em padrões estabelecidos.

5.2.2 Implicações para a Prática

Para os profissionais de Engenharia de Software, a principal implicação é que os LLMs devem ser vistos como assistentes poderosos, e não como substitutos autônomos para engenheiros de requisitos. A forte dependência da clareza da entrada significam que a supervisão humana permanece indispensável.

Com base nos achados, podem-se extrair as seguintes recomendações práticas:

- Para gerar uma lista inicial de requisitos atômicos e focados, que podem servir como base para decomposição e refinamento, a estratégia **`Few-shot`** é a mais recomendada, pois demonstrou ser a mais eficaz para garantir a *Singularidade*.
- Ao utilizar estratégias que visam gerar conteúdo mais detalhado, como a **`Expert Identity`**, os profissionais devem estar cientes do risco de produzir requisitos não singulares. Embora possam conter informações úteis para melhorar a *Verificabilidade*, essas saídas devem ser tratadas como especificações que requerem decomposição manual em múltiplos requisitos atômicos antes de serem integradas ao processo de desenvolvimento.
- A qualidade da entrada é primordial. Investir tempo na clarificação de títulos de *issues* ou na elaboração de descrições mais detalhadas antes de alimentar um LLM provavelmente resultará em uma economia de esforço significativa no refinamento posterior dos requisitos gerados.

5.3 Trabalhos Futuros

Conforme detalhado na Seção 4.6.1, este estudo possui limitações que definem as fronteiras de suas conclusões e abrem caminhos para pesquisas futuras. As principais limitações incluem a avaliação de um número restrito de LLMs e estratégias de prompt, o foco exclusivo em títulos de *issues* como fonte de entrada, e a dependência de um juiz LLM para a avaliação, cujos vieses, embora mitigados, não podem ser completamente eliminados.

Com base nessas limitações e nos achados do estudo, as seguintes direções para trabalhos futuros são propostas:

- **Explorar Entradas Mais Ricas:** Investigar a geração de requisitos utilizando não apenas o título, mas o corpo completo da *issue*, incluindo comentários. Isso exigiria técnicas para sumarizar e extrair as informações mais relevantes de longas discussões. A Geração Aumentada por Recuperação (RAG) poderia ser uma abordagem promissora para fornecer contexto relevante ao LLM.
- **Sofisticação das Estratégias de Prompt:** Avaliar técnicas mais avançadas, como *Chain-of-Thought* (CoT), para incentivar o LLM a "raciocinar" sobre o requisito antes de gerá-lo, potencialmente melhorando a qualidade de requisitos derivados de entradas ambíguas.
- **Decomposição Automatizada de Requisitos:** Dado que o prompt `Expert Identity` tende a gerar saídas detalhadas, mas não singulares, uma linha de pesquisa promissora seria desenvolver um segundo estágio no pipeline, onde outro LLM é encarregado de decompor automaticamente esses requisitos complexos em um conjunto de requisitos atômicos e singulares.
- **Validação Humana e Comparativa:** Realizar um estudo comparativo envolvendo especialistas humanos para avaliar os requisitos gerados. Isso permitiria não apenas validar os resultados do *LLM-as-a-Judge* em maior profundidade, mas também quantificar o nível de concordância entre o julgamento humano e o automatizado no contexto específico da ER.
- **Generalização para Outros Contextos:** Replicar o estudo em diferentes domínios, como projetos comerciais de código fechado ou sistemas de segurança crítica, para verificar se os padrões de desempenho e os *trade-offs* observados se mantêm em ambientes com processos de ER mais formais.

A automação da Engenharia de Requisitos, um dos gargalos mais persistentes no desenvolvimento de software, deu um passo significativo com o advento de LLMs. Esta dissertação demonstrou que, embora a promessa de uma automação completa ainda não

tenha sido alcançada, os LLMs são ferramentas de imenso potencial, capazes de transformar entradas informais em artefatos de requisitos estruturados com um nível de qualidade notável.

A principal contribuição deste trabalho é a evidência empírica de que a eficácia dessa transformação não é absoluta, mas sim uma função complexa da clareza da entrada, da escolha do modelo e, crucialmente, da estratégia de interação empregada. A descoberta de *trade-offs* explícitos entre atributos de qualidade fundamentais, como *Verificabilidade* e *Singularidade*, sublinha a necessidade de uma abordagem estratégica e consciente ao aplicar essas tecnologias. Em última análise, o futuro da Engenharia de Requisitos assistida por IA reside não na substituição da perícia humana, mas em uma colaboração sinérgica, onde a capacidade de geração dos LLMs é guiada e refinada pela intuição, conhecimento de domínio e julgamento crítico dos engenheiros de software.

Referências

[1] Sommerville, Ian: *Software engineering*. Pearson, Boston, 10ª edição, 2016, ISBN 9780133943030. 1, 8, 9

[2] Attanayaka, Buddhima, Dasuni Nawinna, Kalpani Manathunga e Pradeep K.W. Abeygunawardhana: *Success factors of requirement elicitation in the field of software engineering*. Em *2022 4th International Conference on Advancements in Computing (ICAC)*, páginas 240–245, 2022. 1, 3, 8, 9

[3] Tasnim, Maliha, Maruf Rayhan, Zheying Zhang e Timo Poranen: *A systematic literature review on requirements engineering practices and challenges in open-source projects*. Em *2023 49th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, páginas 278–285, 2023. 1, 2, 3, 10, 12

[4] Lim, Sachiko, Aron Henriksson e Jelena Zdravkovic: *Data-driven requirements elicitation: A systematic literature review*. SN Comput. Sci., 2(1):16, 2021. `https://doi.org/10.1007/s42979-020-00416-4`. 1, 3, 10

[5] IEEE: *ISO/IEC/IEEE International Standard - systems and software engineering – life cycle processes – requirements engineering*. ISO/IEC/IEEE 29148:2018(E), páginas 1–104, 2018. 1, 4, 8, 11, 17, 18, 19, 20, 27, 28

[6] Zhao, Wayne Xin, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie e Ji-Rong Wen: *A survey of large language models*. CoRR, abs/2303.18223, 2023. `https://doi.org/10.48550/arXiv.2303.18223`. 1, 12, 13, 14

[7] Arora, Chetan, John Grundy e Mohamed Abdelrazek: *Advancing requirements engineering through generative AI: assessing the role of llms*. CoRR, abs/2310.13976, 2023. `https://doi.org/10.48550/arXiv.2310.13976`. 1, 3, 10, 17, 33

[8] Alhoshan, Waad, Alessio Ferrari e Liping Zhao: *Zero-shot learning for requirements classification: An exploratory study*. Inf. Softw. Technol., 159:107202, 2023. `https://api.semanticscholar.org/CorpusID:256697343`. 1, 4, 17, 19, 21, 25

[9] White, Jules, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith e Douglas C. Schmidt: *A prompt pattern catalog to enhance prompt engineering with chatgpt*. ArXiv, abs/2302.11382, 2023. `https://api.semanticscholar.org/CorpusID:257079092`. 2, 3, 4, 14, 36

[10] Ronanki, Krishna, Beatriz Cabrero Daniel, Jennifer Horkoff e Christian Berger: *Requirements engineering using generative ai: Prompts and prompting patterns*. ArXiv, abs/2311.03832, 2023. https://api.semanticscholar.org/CorpusID:265043266. 2, 3, 4, 14, 15, 18, 19, 21, 25, 36, 38, 48

[11] Mesquita, Rodrigo, Geovana Ramos Sousa Silva e Edna Dias Canedo: *On the experiences of practitioners with requirements elicitation techniques*. Proceedings of the XXXVII Brazilian Symposium on Software Engineering, 2023. https://api.semanticscholar.org/CorpusID:262467959. 2

[12] Canedo, Edna Dias, Angélica Toffano Seidel Calazans, Geovana Ramos Sousa Silva, Eloisa Toffano Seidel Masson e Isabel Sofia Brito: *On the challenges to documenting requirements in agile software development: A practitioners' perspective*. Anais do XXVII Congresso Ibero-Americano em Engenharia de Software (CIbSE 2024), 2024. https://api.semanticscholar.org/CorpusID:270177412. 2

[13] Krishna, Madhava, Bhagesh Gaur, Arsh Verma e Pankaj Jalote: *Using llms in software requirements specifications: An empirical evaluation*. Em *2024 IEEE 32nd International Requirements Engineering Conference (RE)*, páginas 475–483, 2024. 3, 4, 17, 19, 21, 27, 35, 47

[14] Almonte, Jomar Thomas, Santhosh Anitha Boominathan e Nathalia Nascimento: *Automated non-functional requirements generation in software engineering with large language models: A comparative study*, 2025. https://arxiv.org/abs/2503.15248. 3, 17, 21, 27, 35, 47

[15] Zheng, Lianmin, Wei Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez e Ion Stoica: *Judging llm-as-a-judge with mt-bench and chatbot arena*, 2023. https://arxiv.org/abs/2306.05685. 4, 16, 17, 27, 28, 42, 44, 48

[16] Ralph, Paul, Nauman bin Ali, Sebastian Baltes, Domenico Bianculli, Jessica Diaz, Yvonne Dittrich, Neil Ernst, Michael Felderer, Robert Feldt, Antonio Filieri, Breno Bernard Nicolau de França, Carlo Alberto Furia, Greg Gay, Nicolas Gold, Daniel Graziotin, Pinjia He, Rashina Hoda, Natalia Juristo, Barbara Kitchenham, Valentina Lenarduzzi, Jorge Martínez, Jorge Melegati, Daniel Mendez, Tim Menzies, Jefferson Molleri, Dietmar Pfahl, Romain Robbes, Daniel Russo, Nyyti Saarimäki, Federica Sarro, Davide Taibi, Janet Siegmund, Diomidis Spinellis, Miroslaw Staron, Klaas Stol, Margaret Anne Storey, Davide Taibi, Damian Tamburri, Marco Torchiano, Christoph Treude, Burak Turhan, Xiaofeng Wang e Sira Vegas: *Empirical standards for software engineering research*, 2021. https://arxiv.org/abs/2010.03525. 6

[17] Pérez-Verdejo, J. Manuel, Á. J. Sánchez-García, J. O. Ocharán-Hernández, E. Mezura-Montes e K. Cortés-Verdín: *Requirements and github issues: An automated approach for quality requirements classification*. Programming and Computer Software, 47(8):704–721, dezembro 2021, ISSN 0361-7688, 1608-3261. https://link.springer.com/10.1134/S0361768821080193. 10

[18] Minaee, Shervin, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain e Jianfeng Gao: *Large language models: A survey*. 2024. https://arxiv.org/abs/2402.06196. 12, 13, 14

[19] Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser e Illia Polosukhin: *Attention is all you need*. Advances in neural information processing systems, 30, 2017. 12, 13

[20] Radford, Alec, Karthik Narasimhan, Tim Salimans, Ilya Sutskever *et al.*: *Improving language understanding by generative pre-training*. 13

[21] Devlin, Jacob, Ming Wei Chang, Kenton Lee e Kristina Toutanova: *Bert: Pre-training of deep bidirectional transformers for language understanding*. arXiv preprint arXiv:1810.04805, 2018. 14

[22] Ouyang, Long, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike e Ryan Lowe: *Training language models to follow instructions with human feedback*, 2022. https://arxiv.org/abs/2203.02155. 14

[23] Schulhoff, Sander, Michael Ilie, Nishant Balepur, Konstantine Kahadze, Amanda Liu, Chenglei Si, Yinheng Li, Aayush Gupta, HyoJung Han, Sevien Schulhoff *et al.*: *The prompt report: A systematic survey of prompting techniques*. arXiv preprint arXiv:2406.06608, 2024. 14, 15

[24] Brown, Tom B., Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeff Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever e Dario Amodei: *Language models are few-shot learners*. ArXiv, abs/2005.14165, 2020. https://api.semanticscholar.org/CorpusID:218971783. 15, 25

[25] Xu, Benfeng, An Yang, Junyang Lin, Quang Wang, Chang Zhou, Yongdong Zhang e Zhendong Mao: *Expertprompting: Instructing large language models to be distinguished experts*. ArXiv, abs/2305.14688, 2023. https://api.semanticscholar.org/CorpusID:258865458. 15, 26

[26] Lin, Chin Yew: *ROUGE: A package for automatic evaluation of summaries*. Em *Text Summarization Branches Out*, páginas 74–81, Barcelona, Spain, julho 2004. Association for Computational Linguistics. https://aclanthology.org/W04-1013/. 16

[27] Papineni, Kishore, Salim Roukos, Todd Ward e Wei Jing Zhu: *Bleu: a method for automatic evaluation of machine translation*. Em *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, página 311–318,

USA, 2002. Association for Computational Linguistics. https://doi.org/10.3115/1073083.1073135. 16

[28] Li, Dawei, Bohan Jiang, Liangjie Huang, Alimohammad Beigi, Chengshuai Zhao, Zhen Tan, Amrita Bhattacharjee, Yuxuan Jiang, Canyu Chen, Tianhao Wu, Kai Shu, Lu Cheng e Huan Liu: *From generation to judgment: Opportunities and challenges of llm-as-a-judge*, 2025. https://arxiv.org/abs/2411.16594. 16

[29] Gu, Jiawei, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Saizhuo Wang, Kun Zhang, Yuanzhuo Wang, Wen Gao, Lionel Ni e Jian Guo: *A survey on llm-as-a-judge*, 2024. https://arxiv.org/abs/2411.15594. 16, 17, 28, 42

[30] Khan, Javed Ali, Shamaila Qayyum e Hafsa Shareef Dar: *Large language model for requirements engineering: A systematic literature review.* março 2025. https://www.researchsquare.com/article/rs-5589929/v1. 17

[31] Hymel, Cory e Hiroe Johnson: *Analysis of llms vs human experts in requirements engineering*, 2025. https://arxiv.org/abs/2501.19297. 18, 19, 21

[32] Ronanki, Krishna, Christian Berger e Jennifer Horkoff: *Investigating chatgpt's potential to assist in requirements elicitation processes*, 2023. https://arxiv.org/abs/2307.07381. 18, 21

[33] Ataei, Mohammadmehdi, Hyunmin Cheong, Daniele Grandi, Ye Wang, Nigel Morris e Alexander Tessier: *Elicitron: An llm agent-based simulation framework for design requirements elicitation*, 2024. https://arxiv.org/abs/2404.16045. 18, 19, 21

[34] Jin, Dongming, Zhi Jin, Xiaohong Chen e Chunhui Wang: *Mare: Multi-agents collaboration framework for requirements engineering.* ArXiv, abs/2405.03256, 2024. https://api.semanticscholar.org/CorpusID:269605506. 18, 21

[35] Lubos, Sebastian, Alexander Felfernig, Thi Ngoc Trang Tran, Damian Garber, Merfat El Mansi, Seda Polat Erdeniz e Viet Man Le: *Leveraging llms for the quality assurance of software requirements.* Em *2024 IEEE 32nd International Requirements Engineering Conference (RE)*, páginas 389–397, 2024. 18, 19, 21

[36] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia,

Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang e Zhen Zhang: *Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning*, 2025. `https://arxiv.org/abs/2501.12948`. 26

[37] White, Colin, Samuel Dooley, Manley Roberts, Arka Pal, Ben Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Siddartha Naidu, Chinmay Hegde, Yann LeCun, Tom Goldstein, Willie Neiswanger e Micah Goldblum: *Livebench: A challenging, contamination-free llm benchmark*, 2024. `https://arxiv.org/abs/2406.19314`. 26

[38] Yang, An, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang e Zihan Qiu: *Qwen2.5 technical report*. arXiv preprint arXiv:2412.15115, 2024. 27

[39] Qwen Team: *QwQ-32B: Embracing the power of reinforcement learning*, March 2025. `https://qwenlm.github.io/blog/qwq-32b/`. 27

[40] Mann, H. B. e D. R. Whitney: *On a test of whether one of two random variables is stochastically larger than the other*. The Annals of Mathematical Statistics, 18(1):50–60, março 1947, ISSN 0003-4851. `http://projecteuclid.org/euclid.aoms/1177730491`. 30

[41] Kruskal, William H. e W. Allen Wallis: *Use of ranks in one-criterion variance analysis*. Journal of the American Statistical Association, 47(260):583–621, dezem-

bro 1952, ISSN 0162-1459, 1537-274X. http://www.tandfonline.com/doi/abs/10.1080/01621459.1952.10483441. 30

[42] Dunn, Olive Jean: *Multiple comparisons using rank sums.* Technometrics, 6(3):241–252, agosto 1964, ISSN 0040-1706, 1537-2723. http://www.tandfonline.com/doi/abs/10.1080/00401706.1964.10490181. 30

[43] Spearman, C.: *The proof and measurement of association between two things.* The American Journal of Psychology, 15(1):72, janeiro 1904, ISSN 00029556. https://www.jstor.org/stable/1412159?origin=crossref. 30

[44] Marques, Nuno, Rodrigo Rocha Silva e Jorge Bernardino: *Using chatgpt in software requirements engineering: A comprehensive review.* Future Internet, 16(6):1–21, 2024, ISSN 1999-5903. 33

[45] Wohlin, Claes, Per Runeson, Martin Host, Magnus C. Ohlsson, Bjorn Regnell e Anders Wesslen: *Experimentation in software engineering*, 2012. https://link.springer.com/book/10.1007/978-3-662-69306-3. 42