Serviço Público Federal

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
SISTEMA INTEGRADO DE PATRIMÔNIO, ADMINISTRAÇÃO E CONTRATOS

PROCESSO 23091.009655/2022-41

ELETRÔNICO

Cadastrado em 24/06/2022

Processo disponível para recebimento com código de barras/QR Code

Nome(s) do Interessado(s): FELIPE TORRES LEITE

E-mail: [redacted]

Identificador: [redacted]

Tipo do Processo:
AFASTAMENTO NO PAÍS (DOCENTE)

Assunto do Processo:
022.121 - APERFEIÇOAMENTO E TREINAMENTO: CURSOS (INCLUSIVE BOLSAS DE ESTUDO) PROMOVIDOS POR OUTRAS INSTITUIÇÕES NO BRASIL

Assunto Detalhado:
SOLICITO AFASTAMENTO PARA DOUTORADO, CONFORME DOCUMENTAÇÃO ANEXA.

Unidade de Origem:
SECRETARIA, ARQUIVO E PROTOCOLO - PAU DOS FERROS (11.01.36.03)

Criado Por:
VANESSA VELEZ DOS SANTOS

Observação:

MOVIMENTAÇÕES ASSOCIADAS

Data	Destino
24/06/2022	DEPARTAMENTO DE ENGENHARIAS E TECNOLOGIA - PAU DOS FERROS (11.01.36.12.08)
13/07/2022	CENTRO MULTIDISCIPLINAR - PAU DOS FERROS (11.01.36.12)
15/07/2022	DEPARTAMENTO DE ENGENHARIAS E TECNOLOGIA - PAU DOS FERROS (11.01.36.12.08)
22/07/2022	CENTRO MULTIDISCIPLINAR - PAU DOS FERROS (11.01.36.12)
25/07/2022	DEPARTAMENTO DE ENGENHARIAS E TECNOLOGIA - PAU DOS FERROS (11.01.36.12.08)
25/07/2022	CENTRO MULTIDISCIPLINAR - PAU DOS FERROS (11.01.36.12)
26/07/2022	PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO (11.01.03)
04/08/2022	DEPARTAMENTO DE ENGENHARIAS E TECNOLOGIA - PAU DOS FERROS (11.01.36.12.08)
04/08/2022	CENTRO MULTIDISCIPLINAR - PAU DOS FERROS (11.01.36.12)
05/08/2022	PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO (11.01.03)
08/08/2022	DIVISÃO DE DESENVOLVIMENTO DE PESSOAS (11.01.04.04)
11/08/2022	COMISSÃO PERMANENTE DE PESSOAL DOCENTE (11.01.26)
19/08/2022	SECRETARIA DE ORGÃOS COLEGIADOS (11.03.01)
01/09/2022	DIVISÃO DE DESENVOLVIMENTO DE PESSOAS (11.01.04.04)
06/10/2022	SETOR DE CADASTRO (11.01.04.05.02)
07/10/2022	PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO (11.01.03)
21/08/2023	DEPARTAMENTO DE ENGENHARIAS E TECNOLOGIA - PAU DOS FERROS (11.01.36.12.08)

Data	Destino
30/11/2023	SETOR DE CAPACITAÇÃO E APERFEIÇOAMENTO (11.01.04.04.02)
19/01/2024	SETOR DE CADASTRO (11.01.04.05.02)
24/01/2024	PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO (11.01.03)
18/07/2024	DEPARTAMENTO DE ENGENHARIAS E TECNOLOGIA - PAU DOS FERROS (11.01.36.12.08)
07/08/2024	CENTRO MULTIDISCIPLINAR - PAU DOS FERROS (11.01.36.12)
20/08/2024	PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO (11.01.03)
23/08/2024	SETOR DE CAPACITAÇÃO E APERFEIÇOAMENTO (11.01.04.04.02)
23/08/2024	SETOR DE CADASTRO (11.01.04.05.02)
28/08/2024	PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO (11.01.03)
30/08/2024	SETOR DE CADASTRO (11.01.04.05.02)
03/09/2024	PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO (11.01.03)
11/09/2024	DIVISÃO DE DESENVOLVIMENTO DE PESSOAS (11.01.04.04)
11/09/2024	SETOR DE CAPACITAÇÃO E APERFEIÇOAMENTO (11.01.04.04.02)
17/09/2024	COMISSÃO PERMANENTE DE PESSOAL DOCENTE (11.01.26)
14/10/2024	SECRETARIA DE ORGÃOS COLEGIADOS (11.03.01)
08/11/2024	SETOR DE CAPACITAÇÃO E APERFEIÇOAMENTO (11.01.04.04.02)
15/01/2025	SETOR DE CADASTRO (11.01.04.05.02)
	SETOR DE CAPACITAÇÃO E APERFEIÇOAMENTO

22/08/2023	CENTRO MULTIDISCIPLINAR - PAU DOS FERROS (11.01.36.12)
23/08/2023	PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO (11.01.03)
23/08/2023	CENTRO MULTIDISCIPLINAR - PAU DOS FERROS (11.01.36.12)
24/08/2023	PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO (11.01.03)
12/09/2023	DEPARTAMENTO DE ENGENHARIAS E TECNOLOGIA - PAU DOS FERROS (11.01.36.12.08)
19/09/2023	PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO (11.01.03)
19/09/2023	DIVISÃO DE DESENVOLVIMENTO DE PESSOAS (11.01.04.04)
25/09/2023	COMISSÃO PERMANENTE DE PESSOAL DOCENTE (11.01.26)
29/09/2023	SECRETARIA DE ORGÃOS COLEGIADOS (11.03.01)
09/11/2023	DIVISÃO DE DESENVOLVIMENTO DE PESSOAS (11.01.04.04)
15/01/2025	(11.01.04.04.02)
28/10/2025	DEPARTAMENTO DE ENGENHARIAS E TECNOLOGIA - PAU DOS FERROS (11.01.36.12.08)
05/11/2025	CENTRO MULTIDISCIPLINAR - PAU DOS FERROS (11.01.36.12)
10/11/2025	SETOR DE CAPACITAÇÃO E APERFEIÇOAMENTO (11.01.04.04.02)
17/11/2025	DEPARTAMENTO DE ENGENHARIAS E TECNOLOGIA - PAU DOS FERROS (11.01.36.12.08)
21/11/2025	SECRETARIA, ARQUIVO E PROTOCOLO - PAU DOS FERROS (11.01.36.03)
21/11/2025	DEPARTAMENTO DE ENGENHARIAS E TECNOLOGIA - PAU DOS FERROS (11.01.36.12.08)
21/11/2025	PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO (11.01.03)
25/11/2025	SETOR DE CAPACITAÇÃO E APERFEIÇOAMENTO (11.01.04.04.02)
15/12/2025	COMISSÃO PERMANENTE DE PESSOAL DOCENTE (11.01.26)
19/01/2026	SECRETARIA DE ORGÃOS COLEGIADOS (11.03.01)

Para visualizar este processo, entre no **Portal Público** em https://sipac.ufersa.edu.br/public e acesse a Consulta de Processos.

Visualizar no Portal Público

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
CONSELHO UNIVERSITÁRIO

RESOLUÇÃO Nº 60, DE 29 DE AGOSTO DE 2022

O VICE-REITOR NA PRESIDÊNCIA DO CONSELHO UNIVERSITÁRIO – CONSUNI DA UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO – UFERSA, no uso de suas atribuições legais, e tendo em vista o que estabelece a Lei nº 8.112/1990, de 11 de dezembro de 1990, e nº 12.772/12, de 28 de dezembro de 2012; o Regimento da Ufersa; a Resolução Consuni/Ufersa nº 003, de 25 de junho de 2018; o Processo n° 23091.009655/2022-41; a deliberação deste Órgão Colegiado em sua 7ª Reunião Ordinária de 2022, realizada no dia 29 de agosto de 2022, resolve:

Art. 1º Aprovar o afastamento do servidor docente Felipe Torres Leite, pertencente ao Departamento de Engenharias e Tecnologia – Detec, vinculado ao Centro Multidisciplinar de Pau dos Ferros, com a finalidade de realizar o doutorado em Ciência da Computação na Universidade Federal de Campina Grande – UFCG, em Campina Grande – PB, no período de 22 de setembro de 2022 a 07 de agosto de 2026.

Parágrafo único. A aprovação de que trata o caput deve ser renovada anualmente, sendo tal renovação submetida à análise do Conselho competente.

Art. 2º Esta Resolução entra em vigor nesta data.

ROBERTO VIEIRA PORDEUS

RESOLUÇÃO Nº 60/2022 - SOC (11.03.01)

(Nº do Protocolo: NÃO PROTOCOLADO)

(Assinado digitalmente em 01/09/2022 15:21)
ERICKA TAYANA LIMA BEZERRA
ASSISTENTE EM ADMINISTRACAO
GAB (11.03)
Matrícula: ###292#5

Visualize o documento original em https://sipac.ufersa.edu.br/documentos/ informando seu número: ***60****, ano:* ***2022****, tipo:* ***RESOLUÇÃO****, data de emissão:* ***01/09/2022*** *e o código de verificação:* ***354dbc6867***

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
GABINETE DA REITORIA

PORTARIA Nº 605, DE 5 DE OUTUBRO DE 2022

A REITORA DA UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO, no uso de suas atribuições conferidas pelo Decreto de 21 de agosto de 2020, publicado na edição extra no Diário Oficial da União de 21 de agosto de 2020, e tendo em vista o que consta no Processo no 23091.009655/2022-41; a Resolução n° 60, de 29 de agosto de 2022 do Consuni, resolve:

Art. 1º Autorizar o afastamento do servidor docente Felipe Torres Leite, matrícula Siape nº [redacted], lotado no Departamento de Engenharias e Tecnologia – Detec, vinculado ao Centro Multidisciplinar de Pau dos Ferros, com a finalidade de realizar doutorado em Ciência da Computação na Universidade Federal de Campina Grande – UFCG, em Campina Grande – PB, no período de 28 de setembro de 2022 a 7 de agosto de 2026.

Art. 2º Esta Portaria entra em vigor nesta data e seus efeitos retroagem a 28 de setembro de 2022.

LUDIMILLA CARVALHO SERAFIM DE OLIVEIRA

PORTARIA Nº 1023/2022 - DDP (11.01.04.04)

(Nº do Protocolo: NÃO PROTOCOLADO)

(Assinado digitalmente em 06/10/2022 14:39)
CAMILA DE SOUZA FILGUEIRA DANTAS
ASSISTENTE EM ADMINISTRACAO
SCA (11.01.04.04.02)
Matrícula: ###420#8

*Visualize o documento original em https://sipac.ufersa.edu.br/documentos/ informando seu número: **1023**, ano: **2022** , tipo: **PORTARIA**, data de emissão: **06/10/2022** e o código de verifica*[redacted]

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
CONSELHO DE ENSINO, PESQUISA E EXTENSÃO

RESOLUÇÃO Nº 39, DE 18 DE OUTUBRO DE 2023

O VICE-REITOR NA PRESIDÊNCIA DO CONSELHO DE ENSINO, PESQUISA E EXTENSÃO – CONSEPE DA UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO – UFERSA, no uso de suas atribuições legais, e tendo em vista o que estabelecem as Leis nº 8.112, de 11 de dezembro de 1990, e nº 12.772, de 28 de dezembro de 2012; o Regimento da Ufersa; a Resolução Consuni/Ufersa nº 003/2018, de 25 de junho de 2018; o Processo nº 23091.009655/2022-41; a Portaria nº 1.879, de 4 de outubro de 2023, do Gabinete da Reitoria da Ufersa; a deliberação deste Órgão Colegiado em sua 9ª Reunião Ordinária de 2023, realizada no dia 18 de outubro de 2023, resolve:

Art. 1º Homologar a designação pela Reitora, *ad referendum* do Consepe, de renovação de afastamento do servidor docente Felipe Torres Leite, pertencente ao Departamento de Engenharias e Tecnologia – Detec, vinculado ao Centro Multidisciplinar de Pau dos Ferros, com a finalidade de dar continuidade ao doutorado em Ciência da Computação, na Universidade Federal de Campina Grande – UFCG, em Campina Grande, na Paraíba, no período de 22 de setembro de 2023 a 21 de setembro de 2024.

Art. 2º Esta Resolução entra em vigor nesta data e seus efeitos retroagem a 22 de setembro de 2023.

ROBERTO VIEIRA PORDEUS

RESOLUÇÃO Nº 112/2023 - SOC (11.03.01)

(Nº do Protocolo: NÃO PROTOCOLADO)

(Assinado digitalmente em 09/11/2023 10:22)
ERICKA TAYANA LIMA BEZERRA
ASSISTENTE EM ADMINISTRACAO
GAB (11.03)
Matrícula: ###292#5

Visualize o documento original em https://sipac.ufersa.edu.br/documentos/ informando seu número: ***112****, ano:* ***2023****, tipo:* ***RESOLUÇÃO****, data de emissão:* ***09/11/2023*** *e o código de verificação:* ***88a5f67f31***

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
GABINETE DA REITORIA

PORTARIA Nº 1.879, DE 4 DE OUTUBRO DE 2023

A REITORA DA UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO, no uso de suas atribuições conferidas pelo Decreto de 21 de agosto de 2020, publicado na edição extra no Diário Oficial da União de 21 de agosto de 2020, e tendo em vista o que estabelece a Lei nº 8.112, de 11 de dezembro de 1990, e suas alterações; a Lei nº 12.772, de 28 de dezembro de 2012, e suas alterações; o Decreto nº 9.991, de 28 de agosto de 2019, e suas alterações; o inciso VI do artigo 44 do Estatuto da universidade; a Resolução Consuni/Ufersa nº 003, de 25 de junho de 2018; o Processo n° 23091.009655/2022-41; a Resolução n° 60, de 29 de agosto de 2022 do Consuni da Ufersa; a Portaria nº 605, de 5 de outubro de 2022, resolve:

Art. 1º Autorizar, ad referendum do Conselho de Ensino, Pesquisa e Extensão – Consepe, a renovação de afastamento do servidor docente Felipe Torres Leite, matrícula Siape nº [redacted], pertencente ao Departamento de Engenharias e Tecnologia – Detec, vinculado ao Centro Multidisciplinar de Pau dos Ferros, com a finalidade de dar continuidade ao doutorado em Ciência da Computação, na Universidade Federal de Campina Grande – UFCG, em Campina Grande, na Paraíba, no período de 22 de setembro de 2023 a 21 de setembro de 2024.

Art. 2º Esta Portaria entra em vigor nesta data e seus efeitos retroagem a 22 de setembro de 2023.

LUDIMILLA CARVALHO SERAFIM DE OLIVEIRA

PORTARIA Nº 3/2023 - SCA (11.01.04.04.02)

(Nº do Protocolo: NÃO PROTOCOLADO)

(Assinado digitalmente em 19/01/2024 11:10)
MONALIZA FERREIRA RODRIGUES DE PAULA
ASSISTENTE EM ADMINISTRACAO
SCA (11.01.04.04.02)
Matrícula: ###840#8

*Visualize o documento original em https://sipac.ufersa.edu.br/documentos/ informando [redacted], tipo: **PORTARIA**, data de emissão: **19/01/2024** e o código de verificação [redacted]*

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
CONSELHO DE ENSINO, PESQUISA E EXTENSÃO

RESOLUÇÃO Nº 57, DE 29 DE OUTUBRO DE 2024

O VICE-REITOR NA PRESIDÊNCIA DO CONSELHO DE ENSINO, PESQUISA E EXTENSÃO – CONSEPE DA UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO – UFERSA, no uso de suas atribuições legais, e tendo em vista o que estabelecem as Leis nº 8.112, de 11 de dezembro de 1990, e nº 12.772, de 28 de dezembro de 2012, e suas respectivas alterações; o Decreto nº 9.991, de 28 de agosto de 2019, alterado pelo Decreto nº 10.506, de 2 de outubro de 2020; o Regimento da Ufersa; a Resolução Consuni/Ufersa nº 003/2018, de 25 de junho de 2018; o Processo nº 23091.009655/2022-41; a deliberação deste Órgão Colegiado em sua 9ª Reunião Ordinária de 2024, realizada no dia 29 de outubro de 2024, resolve:

Art. 1º Aprovar a renovação de afastamento do servidor docente Felipe Torres Leite, pertencente ao Departamento de Engenharias e Tecnologia — DETEC, vinculado ao Centro Multidisciplinar de Pau dos Ferros – CMPF, com a finalidade de dar continuidade ao Doutorado em Ciências da Computação, na Universidade Federal de Campina Grande — UFCG, em Campina Grande – PB, no período de 22 de setembro de 2024 a 21 de setembro de 2025.

Art. 2º Esta Resolução entra em vigor nesta data e seus efeitos retroagem a 22 de setembro de 2024.

NILDO DA SILVA DIAS

RESOLUÇÃO Nº 79/2024 - SOC (11.03.01)

(Nº do Protocolo: NÃO PROTOCOLADO)

(Assinado digitalmente em 08/11/2024 09:59)
LUIZ DJALMA DIAS FILHO
ASSISTENTE EM ADMINISTRACAO
GAB (11.03)
Matrícula: ###038#8

*Visualize o documento original em https://sipac.ufersa.edu.br/documentos/ informando seu número: **79**, ano: **2024**, tipo: **RESOLUÇÃO**, data de emissão: **08/11/2024** e o código de verificação: **aee820ba1e***

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
GABINETE DA REITORIA

PORTARIA Nº 2.179, DE 26 DE DEZEMBRO DE 2024

O VICE-REITOR NO EXERCÍCIO DA REITORIA DA UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO, no uso de suas atribuições conferidas pela Portaria nº 1.437, de 9 de setembro de 2024, publicada no Diário Oficial da União nº 175, seção 2, pág. 37, e tendo em vista o que estabelecem os incisos VI e XIX do art. 44 do Estatuto da universidade; o artigo 58 da Lei nº 8.112, de 11 de dezembro de 1990, e suas alterações; a Resolução nº 57, de 29 de outubro de 2024, do Consepe da Ufersa; a Portaria nº 605, de 5 de outubro de 2022; o Processo n° 23091.009655/2022-41, resolve:

Art. 1º Autorizar a renovação de afastamento do servidor docente Felipe Torres Leite, pertencente ao Departamento de Engenharias e Tecnologia — DETEC, vinculado ao Centro Multidisciplinar de Pau dos Ferros – CMPF, com a finalidade de dar continuidade ao Doutorado em Ciências da Computação, na Universidade Federal de Campina Grande — UFCG, em Campina Grande – PB, no período de 22 de setembro de 2024 a 21 de setembro de 2025.

Art. 2º Esta Portaria entra em vigor nesta data e seus efeitos retroagem a 22 de setembro de 2024.

NILDO DA SILVA DIAS

PORTARIA Nº 154/2024 - SCA (11.01.04.04.02)

(Nº do Protocolo: NÃO PROTOCOLADO)

(Assinado digitalmente em 15/01/2025 16:11)
MONALIZA FERREIRA RODRIGUES DE PAULA
CHEFE DE SETOR
SCA (11.01.04.04.02)
Matrícula: ###840#8

Visualize o documento original em https://sipac.ufersa.edu.br/documentos/ informando seu número: ***154****, ano:* ***2024****, tipo:* ***PORTARIA****, data de emissão:* ***15/01/2025*** *e o código de verificação:* ***e1c90fefd1***

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO - UFERSA
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO - PROPPG
Av. Francisco Mota, 572 – C. Postal 137 – Bairro Pres. Costa e Silva – Mossoró – RN – CEP: 59.625-900 - Tel.: (84)3317-8296/8295 – E.mail: proppg@ufersa.edu.br

REQUERIMENTO E ANEXOS PARA RENOVAÇÃO DE AFASTAMENTOS DE SERVIDORES DOCENTES DA UFERSA PARA QUALIFICAÇÃO EM INSTITUIÇÕES NACIONAIS OU ESTRANGEIRAS EM NÍVEL DE PÓS-GRADUAÇÃO *STRICTO SENSU*

1. PREENCHIDO PELO REQUERENTE

Nome (completo sem abreviaturas): Felipe Torres Leite
Identidade: [redacted]
CPF: [redacted]
E-mail: [redacted]
Tipo de Afastamento: Integral: (x) **Parcial:** ( )
Tempo de Serviço Averbado para Aposentadoria: (7) **Anos**
Início de Exercício no Cargo: 04/04/2018 **Total:** 7 **anos** 6 **meses** (Anexar Declaração do PRORH).

2. PREENCHIDO PELO REQUERENTE
CURSO: Ciência da Computação
Nível: Mestrado ( ) **Doutorado** (x)
Área de concentração: Engenharia de Software
Liberação inicial: Início: 22/09/2024 **Término:** 21/09/2025
Período solicitado para (renovação): Início: 22/09/2025 **Término:** 02/08/2026
Previsão para término do curso: 02/08/2026

ANEXAR (Obrigatório)

I. Lista de verificação própria disponibilizada pela PROPPG (**Check-List**); ***(Anexo I)***

II – Justificativa de seu requerimento; ***(Anexo II)***

III- Relatório de atividades acadêmicas (Anexo III) (quando se tratar do relatório referente ao 3º semestre (mestrado) e 5º semestre (doutorado), deverá ser acompanhado do **projeto de dissertação/Tese)**

IV- Relatório de avaliação de desempenho, feito pelo/a orientador/a (Anexo IV)

V - Declaração de matrícula (Local da pós-graduação) (Anexo V)

VI- Histórico Escolar (Anexo VII) (Disponível na Página da PROPPG**)**

VII- Termo de Compromisso dos docentes que assumirão os componentes curriculares do docente afastado, durante o período de renovação do afastamento, restrito aos casos de indisponibilidade de vaga para contratação de professor substituto; ***(Anexo VII)***

VIII – Termo de Compromisso, devidamente preenchido e assinado com testemunhas; ***(Anexo VIII)***

IX - Parecer da chefia imediata (Departamento acadêmico de lotação do requerente); ***(Anexo IX)***

X - Parecer do Conselho do Centro ao qual o requerente faz parte. ***(Anexo X).***

XI-Declaração que não responde a PAD ou Sindicância **(https://progepe.ufersa.edu.br/formularios/);**

XII - Declaração de Licenças e Afastamentos **(https://progepe.ufersa.edu.br/solicitacao-de-declaracao-3/);**

XIII - Cópia do trecho do Plano de Desenvolvimento de Pessoas (PDP) da Ufersa, onde está indicada a necessidade de desenvolvimento correlacionando o afastamento com as competências aprovadas no PDP vigente da UFERSA (https://progepe.ufersa.edu.br/planos-de-desenvolvimento-de-pessoas-anuais/).

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO - UFERSA
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO - PROPPG
Av. Francisco Mota, 572 – C. Postal 137 – Bairro Pres. Costa e Silva – Mossoró – RN – CEP: 59.625-900 - Tel.: (84)3317-8296/8295 – E.mail: proppg@ufersa.edu.br

***Obs**. A renovação de afastamento para qualificação em nível de pós-graduação stricto sensu dar-se-á nos termos da legislação em vigor, devendo a manifestação de intenção de renovação do afastamento ser protocolada em **até 60 (sessenta) dias antes do término do afastamento**. Conforme Art. 19. da RESOLUÇÃO CONSUNI/UFERSA Nº 003/2018, de 25/06/2018*

Data: 23/10/2025

_____

Assinatura do requerente

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO - UFERSA
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO - PROPPG
Av. Francisco Mota, 572 – C. Postal 137 – Bairro Pres. Costa e Silva – Mossoró – RN – CEP: 59.625-900 - Tel.: (84)3317-8296/8295 – E.mail: proppg@ufersa.edu.br

(Anexo I)

Check-List – Renovação de Afastamento para qualificação

Nome do solicitante: Felipe Torres Leite	
Local da Qualificação: Universidade Federal de Campina Grande - UFCG	
(x) No País ( ) No exterior	
Período solicitado para renovação do afastamento: 22/09/2025 a 02/08/2026	
Documentos Anexados – Processo de Renovação:	**Número da página (Preenchido pela PROPPG):**
I. Lista de verificação própria disponibilizada pela PROPPG (Check-List); ***(Anexo I)***	
II. Justificativa de seu requerimento; (***Anexo II***)	
III. Relatório de atividades acadêmicas (***Anexo III***)	
IV. Relatório de avaliação de desempenho, feito pelo orientador (***Anexo IV***)	
V. Declaração de Matrícula ***(Anexo V)***	
VI. Histórico Escolar – Atualizado ***(Anexo VI)***	
VII – Termo de Compromisso, devidamente preenchido e assinado com testemunhas; (Anexo VIII)	
VIII. Documentação que formalize a substituição do(a) interessado: ***(Anexo VIII)*** • Utilização de vaga ou disponibilidade de professor substituto a ser contratado(a) • Termo de Compromisso dos docentes que assumirão as disciplinas	
IX. Parecer da chefia imediata (Departamento acadêmico de lotação do requerente); ***(Anexo IX)***	
X. Parecer do Conselho do Centro ao qual o requerente faz parte. ***(Anexo X).***	
XI-Declaração que não responde a PAD ou Sindicância (https://progepe.ufersa.edu.br/formularios/);	
XII - Declaração de Licenças e Afastamentos (https://progepe.ufersa.edu.br/solicitacao-de- declaracao-3/);	
XIII - Cópia do trecho do Plano de Desenvolvimento de Pessoas (PDP) da Ufersa, onde está indicada a necessidade de desenvolvimento correlacionando o afastamento com as competências aprovadas no PDP vigente da UFERSA (https://progepe.ufersa.edu.br/planos-de-desenvolvimento-de-pessoas-anuais/).	

D E C L A R A Ç Ã O

DECLARAMOS, para os devidos fins, que o(a) servidor(a) FELIPE TORRES LEITE, matrícula SIAPE [redacted], ocupante do cargo de PROFESSOR 3 GRAU, classe B, nível 001, do quadro de pessoal do(a) UFERSA, foi admitido(a) a partir de 04/04/2018, sendo lotado(a) no(a) DEPARTAMENTO DE ENGENHARIAS E TECNOLOGIA - PAU DOS FERROS, em regime de Dedicação exclusiva.

Mossoró/RN, 17 de Outubro de 2025.

Código de verificação: [redacted]

Para verificar a autenticidade deste documento acesse **http://sigrh.ufersa.edu.br/sigrh/public/autenticidade/tipo_documento.jsf**, informando a matrícula siape, data de emissão do documento e o código de verificação.

Av. Francisco Mota, 572 – C. Postal 137 – Bairro Pres. Costa e Silva – Mossoró – RN – CEP: 59.625-900 - Tel.: (84)3317-8296/8295 – E.mail: proppg@ufersa.edu.br

(Anexo II)

JUSTIFICATIVA PARA O AFASTAMENTO

Eu, Felipe Torres Leite, matrícula SIAPE nº [redacted], venho respeitosamente solicitar a **renovação do afastamento integral** referente ao período de **22 de setembro de 2025 a 02 de agosto de 2026**, com o objetivo de prosseguir na realização das atividades competentes ao processo de **Doutoramento em Ciência da Computação** da **Universidade Federal de Campina Grande (UFCG)**, no âmbito do **Programa de Pós-Graduação em Ciência da Computação (PPGCC)**.

---

1. Contextualização

Atualmente, professor efetivo da **Universidade Federal Rural do Semi-Árido (UFERSA)**, lotado no **Departamento de Engenharias e Tecnologia (DETEC)** do **Centro Multidisciplinar de Pau dos Ferros (CMPF)**, atuando nos cursos de **Bacharelado Interdisciplinar em Tecnologia da Informação (BITI)** e **Bacharelado em Engenharia de Software (BES)**.

O afastamento integral foi concedido para a realização do Doutorado em Ciência da Computação no Programa de Pós-Graduação em Ciência da Computação (PPGCC) da Universidade Federal de Campina Grande (UFCG), com início das atividades acadêmicas em agosto de 2022 e **liberação efetiva do afastamento** a partir de **22 de setembro de 2022** até 07 de agosto de 2026. Desde então, o afastamento vem sendo renovado anualmente, conforme pareceres favoráveis aprovados pelos órgãos competentes (em anexo, as *Resoluções CONSUNI/UFERSA nº 60, de 29 de agosto de 2022; CONSEPE/UFERSA nº 39, de 18 de outubro de 2023;* e *CONSEPE/UFERSA nº 57, de 29 de outubro de 2024*).

Em face do exposto, **solicito a renovação do afastamento integral** correspondente ao período de **22 de setembro de 2025 a 02 de agosto de 2026**, referente ao **último ciclo de afastamento previsto**, em observância ao **limite máximo de quatro anos** estabelecido pela legislação vigente e em conformidade com as diretrizes da *Resolução CONSUNI/UFERSA nº 003/2018, de 25 de junho de 2018*.

Este pedido fundamenta-se na necessidade de conclusão das etapas finais da pesquisa de doutorado, incluindo a defesa da qualificação, a execução e validação dos experimentos e a elaboração da tese, atividades que exigem **dedicação exclusiva** e acompanhamento presencial junto ao grupo de pesquisa e ao orientador na instituição de destino.

Considerando que o programa de doutorado está sediado em Campina Grande/PB, enquanto a unidade de origem do docente está localizada em Pau dos Ferros/RN, em estados distintos e separados por significativa distância geográfica, o afastamento integral permanece indispensável para garantir a viabilidade logística e acadêmica do desenvolvimento da pesquisa.

---

2. Desenvolvimento das Atividades de Doutorado e Reestruturação de Orientação

Desde o ingresso no **Programa de Pós-Graduação em Ciência da Computação (PPGCC)** da **Universidade Federal de Campina Grande (UFCG)**, em **agosto de 2022**, as atividades acadêmicas e de pesquisa vêm sendo desenvolvidas de forma contínua, conforme o plano de qualificação aprovado.

No primeiro biênio, o requerente esteve vinculado ao **Laboratório de Sistemas Embarcados e Computação Pervasiva (Embedded)**, sob orientação do **Prof. Dr. Hyggo Almeida**, participando de projetos do grupo de pesquisa **ISE – Intelligent Software Engineering**. Nesse período, foram **concluídas todas as disciplinas obrigatórias e eletivas**, bem como **atividades de pesquisa e orientação vinculadas ao projeto de tese**, com ênfase em **Engenharia de Software Inteligente** e **aplicações de aprendizado de máquina à detecção de padrões de código e refatoração automática**.

Entretanto, no decorrer do segundo ano, tornou-se necessária a alteração de orientação em virtude do afastamento institucional do orientador originalmente designado, decisão de caráter pessoal formalmente aprovada pelo colegiado do programa, que resultou em sua desvinculação temporária de todas as atividades acadêmicas, incluindo a orientação de doutorandos. A orientação passou então a ser conduzida pelo **Prof. Dr. Wilkerson Andrade**, docente permanente do PPGCC/UFCG e membro do **Software Practices Laboratory (SPLab)**, vinculado ao Departamento de Sistemas e Computação da UFCG,

assegurando a continuidade da linha de pesquisa originalmente proposta. Em decorrência dessa transição, o projeto foi realocado para o SPLab, o que resultou na atualização do plano de pesquisa e da metodologia adotada, mantendo-se a linha temática central, mas com uma abordagem aprimorada e mais alinhada às práticas contemporâneas da Engenharia de Software Experimental.

O espaço físico do SPLab é fundamental para o andamento da pesquisa, pois proporciona o acesso a recursos computacionais e infraestrutura especializada indispensáveis à execução dos experimentos planejados. A utilização desses recursos é condição necessária para que a investigação alcance os resultados esperados dentro do cronograma aprovado, garantindo a qualidade científica e a viabilidade técnica do estudo.

A transição entre orientadores e grupos de pesquisa, embora necessária, gerou impactos no cronograma original, especialmente na etapa de coleta e análise de dados experimentais. Tal reestruturação demandou a solicitação formal de prorrogação de prazo ao PPGCC/UFCG, a qual foi deferida após tramitação administrativa, garantindo a continuidade do trabalho com novo cronograma aprovado (em anexo documentos referentes à aprovação do orientador e do colegiado do programa no tocante à prorrogação de prazo).

Atualmente, o projeto encontra-se em fase de ajuste e execução dos experimentos revisados, com previsão de finalização da coleta de dados e submissão de artigos científicos ainda no período correspondente à renovação solicitada. As atividades de qualificação já estão autorizadas e em preparação, conforme o plano atualizado.

A mudança de laboratório e orientação representa uma evolução acadêmica e técnica significativa no desenvolvimento da tese, ampliando a rede de pesquisa e fortalecendo o vínculo entre os grupos Embedded e SPLab, ambos reconhecidos pela excelência em Engenharia de Software Inteligente e Computação Experimental.

---

3. Situação Atual e Andamento da Qualificação

A pesquisa encontra-se em fase de consolidação experimental e etapa preparatória para a qualificação de doutorado, conforme o novo cronograma aprovado pelo colegiado do PPGCC/UFCG após a readequação decorrente da

mudança de orientação e do laboratório de pesquisa. Atualmente, o trabalho está na fase de análise dos resultados parciais e preparação de artigos científicos para divulgação dos resultados preliminares.

A **qualificação de doutorado** encontra-se formalmente **autorizada e em fase de planejamento**, com previsão de realização **no período de vigência desta renovação** e sob supervisão direta do orientador. Dessa forma, ainda não é possível anexar o **texto de qualificação** a este processo administrativo, uma vez que o documento está em elaboração e será submetido à banca examinadora do programa nas próximas etapas.

O colegiado do **Programa de Pós-Graduação em Ciência da Computação (PPGCC/UFCG)** reconheceu oficialmente a necessidade de **ajuste de cronograma e prorrogação de prazo** (em anexo, certidão referente ao processo nº *23096.051522/2025-10*), tendo **deferido o pedido apresentado pelo orientador e pelo doutorando**. Ressalta-se que, **devido à tramitação prolongada desse processo**, o parecer final foi disponibilizado ao requerente apenas em **17 de outubro de 2025**, o que impactou o cronograma de submissão do pedido de renovação de afastamento.

Essa aprovação assegura o **alinhamento institucional entre a UFERSA e a UFCG** quanto à condução regular e responsável do projeto de pesquisa, **e reafirma que o andamento acadêmico segue plenamente dentro dos marcos legais e regimentais**.

---

4. Substituição Docente e Continuidade Institucional

A continuidade das atividades de **ensino, pesquisa e extensão** vinculadas aos cursos do **Departamento de Engenharias e Tecnologia (DETEC)** do **CMPF/UFERSA** tem sido plenamente assegurada durante o período de afastamento, em conformidade com as normas institucionais de qualificação docente.

Atualmente, as disciplinas e atividades sob responsabilidade do requerente vêm sendo conduzidas por professor substituto devidamente contratado por meio de edital público, com experiência compatível com a área de Computação e desempenho satisfatório nas funções docentes. O profissional

manifestou interesse na renovação de seu contrato, o que permitirá a manutenção das atividades acadêmicas sem prejuízo à comunidade universitária e ao andamento regular do semestre letivo.

Essa estrutura de substituição docente assegura a continuidade dos processos de ensino, pesquisa e extensão, preservando o suporte a projetos institucionalizados no DETEC/CMPF. Assim, garante-se que o afastamento do docente em processo de qualificação não compromete o funcionamento acadêmico da unidade, reforçando o caráter planejado e responsável da solicitação.

---

5. Aspectos Pessoais e Recuperação de Saúde

Durante o ano de 2025, o requerente enfrentou **intercorrências de saúde pessoal e familiar**, diagnosticadas clinicamente como **Transtorno de Ansiedade Generalizada (CID-10: F41.1)**, condição que exigiu **tratamento psiquiátrico e psicológico contínuo**. Esse quadro impactou temporariamente o ritmo de execução das atividades acadêmicas e administrativas, refletindo-se em **atrasos pontuais** no cumprimento de prazos, inclusive na tramitação do pedido de renovação de afastamento.

Com o tratamento em curso e acompanhamento médico regular, houve restabelecimento pleno da capacidade de desempenho, permitindo a retomada integral das atividades de pesquisa e produção científica. O requerente tem seguido todas as recomendações clínicas, o que assegura estabilidade e desempenho adequados para o desenvolvimento das tarefas previstas no cronograma atualizado do doutorado.

Em função dessas circunstâncias e do tempo necessário para que o colegiado do PPGCC emitisse o parecer sobre o pedido de prorrogação de prazo, o presente pedido de renovação foi protocolado após o prazo originalmente estabelecido pela UFERSA. Tal decisão ocorreu com o apoio do orientador, considerando que, nesta etapa, o envio do texto de qualificação é exigido juntamente com a documentação de renovação, o que ainda não foi possível devido à tramitação do parecer e ao próprio andamento da pesquisa.

Dessa forma, solicita-se que as instâncias competentes da UFERSA **considerem a justificativa do envio do processo de renovação de afastamento** fora do prazo, uma vez que o atraso não decorre de negligência,

mas de um contexto documentado, devidamente respaldado pelo programa de pós-graduação, pelo orientador e pelo professor requerente, e que não gera prejuízo à instituição, ao programa ou às atividades acadêmicas em curso.

---

6. Relevância Institucional e Compromisso com o Retorno

A qualificação em nível de doutorado constitui **investimento estratégico da UFERSA** na formação de seu corpo docente, fortalecendo as políticas de valorização profissional e a capacidade de produção científica e tecnológica da instituição.

A pesquisa desenvolvida no **Software Practices Laboratory (SPLab)** da **UFCG** está diretamente alinhada às **demandas contemporâneas da Engenharia de Software**, contribuindo para a integração entre **ensino, pesquisa e inovação tecnológica**, áreas centrais ao desenvolvimento institucional do **CMPF/UFERSA**.

O retorno do docente qualificado proporcionará ganhos concretos à universidade, incluindo:

- Ampliação das linhas de pesquisa em **Engenharia de Software Inteligente** e **Análise de Padrões de Código**;
- Potencialização de parcerias e projetos interinstitucionais com grupos consolidados da **UFCG**;
- Fortalecimento da formação discente nos cursos de **BITI** e **BES**;
- Aumento da produção científica e tecnológica da unidade, com ênfase na aplicação de técnicas de **Inteligência Artificial** e **Aprendizado de Máquina** em contextos de **Engenharia de Software Experimental**.

Além disso, a atuação em grupos de pesquisa reconhecidos nacionalmente tem possibilitado a integração da UFERSA em redes colaborativas de pesquisa, ampliando sua visibilidade acadêmica e o alcance de seus resultados científicos.

O docente reafirma, portanto, o compromisso institucional com o retorno às atividades na UFERSA, colocando-se à disposição para aplicar, difundir e multiplicar os conhecimentos adquiridos durante o doutorado, contribuindo para

o desenvolvimento de novos projetos, disciplinas e grupos de estudo vinculados à universidade.

---

7. Conclusão e Pedido

Considerando o histórico de dedicação integral às atividades do doutorado, o cumprimento das obrigações acadêmicas e regimentais, a relevância científica do projeto e a manutenção da normalidade das atividades docentes no campus de origem, solicita-se a **renovação do afastamento integral** pelo período de **22 de setembro de 2025 a 02 de agosto de 2026**, correspondente ao **último ciclo de afastamento previsto**.

Este período é essencial para viabilizar a **realização da qualificação**, a **conclusão dos experimentos**, a **redação final da tese** e a **defesa pública**, assegurando o encerramento regular do curso de doutorado e a devolução de resultados acadêmicos de impacto à instituição.

Reitera-se o **comprometimento do requerente com o retorno às atividades docentes e de pesquisa** na **UFERSA**, em consonância com os princípios de responsabilidade, produtividade e contribuição institucional que orientam este processo.

Aproveito para agradecer ao **DETEC**, à **Chefia Imediata** e às **instâncias superiores da UFERSA** pela atenção e pelo contínuo apoio à qualificação docente, reafirmando o compromisso de honrar o investimento institucional por meio de ações concretas de **ensino, pesquisa e inovação**.

Data: 23 de Outubro de 2025

gov.br
Documento assinado digitalmente
FELIPE TORRES LEITE
Data: 23/10/2025 10:37:44-0300
Verifique em https://validar.iti.gov.br

Assinatura do requerente

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
CONSELHO UNIVERSITÁRIO

RESOLUÇÃO Nº 60, DE 29 DE AGOSTO DE 2022

O VICE-REITOR NA PRESIDÊNCIA DO CONSELHO UNIVERSITÁRIO – CONSUNI DA UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO – UFERSA, no uso de suas atribuições legais, e tendo em vista o que estabelece a Lei nº 8.112/1990, de 11 de dezembro de 1990, e nº 12.772/12, de 28 de dezembro de 2012; o Regimento da Ufersa; a Resolução Consuni/Ufersa nº 003, de 25 de junho de 2018; o Processo n° 23091.009655/2022-41; a deliberação deste Órgão Colegiado em sua 7ª Reunião Ordinária de 2022, realizada no dia 29 de agosto de 2022, resolve:

Art. 1º Aprovar o afastamento do servidor docente Felipe Torres Leite, pertencente ao Departamento de Engenharias e Tecnologia – Detec, vinculado ao Centro Multidisciplinar de Pau dos Ferros, com a finalidade de realizar o doutorado em Ciência da Computação na Universidade Federal de Campina Grande – UFCG, em Campina Grande – PB, no período de 22 de setembro de 2022 a 07 de agosto de 2026.

Parágrafo único. A aprovação de que trata o caput deve ser renovada anualmente, sendo tal renovação submetida à análise do Conselho competente.

Art. 2º Esta Resolução entra em vigor nesta data.

ROBERTO VIEIRA PORDEUS:06759688449
Assinado de forma digital por ROBERTO VIEIRA PORDEUS:06759688449
Dados: 2022.08.31 09:35:00 -03'00'

ROBERTO VIEIRA PORDEUS

Emitido em 29/08/2022

RESOLUÇÃO Nº 60/2022 - SOC (11.03.01)

(Nº do Protocolo: NÃO PROTOCOLADO)

(Assinado digitalmente em 01/09/2022 15:21)
ERICKA TAYANA LIMA BEZERRA
ASSISTENTE EM ADMINISTRACAO
GAB (11.03)
Matrícula: 1929215

Para verificar a autenticidade deste documento entre em https://sipac.ufersa.edu.br/documentos/ informando seu número: **60**, ano: **2022**, tipo: **RESOLUÇÃO**, data de emissão: **01/09/2022** e o código de verificação: **354dbc6867**

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
CONSELHO DE ENSINO, PESQUISA E EXTENSÃO

RESOLUÇÃO Nº 39, DE 18 DE OUTUBRO DE 2023

O VICE-REITOR NA PRESIDÊNCIA DO CONSELHO DE ENSINO, PESQUISA E EXTENSÃO – CONSEPE DA UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO – UFERSA, no uso de suas atribuições legais, e tendo em vista o que estabelecem as Leis nº 8.112, de 11 de dezembro de 1990, e nº 12.772, de 28 de dezembro de 2012; o Regimento da Ufersa; a Resolução Consuni/Ufersa nº 003/2018, de 25 de junho de 2018; o Processo nº 23091.009655/2022-41; a Portaria nº 1.879, de 4 de outubro de 2023, do Gabinete da Reitoria da Ufersa; a deliberação deste Órgão Colegiado em sua 9ª Reunião Ordinária de 2023, realizada no dia 18 de outubro de 2023, resolve:

Art. 1º Homologar a designação pela Reitora, *ad referendum* do Consepe, de renovação de afastamento do servidor docente Felipe Torres Leite, pertencente ao Departamento de Engenharias e Tecnologia – Detec, vinculado ao Centro Multidisciplinar de Pau dos Ferros, com a finalidade de dar continuidade ao doutorado em Ciência da Computação, na Universidade Federal de Campina Grande – UFCG, em Campina Grande, na Paraíba, no período de 22 de setembro de 2023 a 21 de setembro de 2024.

Art. 2º Esta Resolução entra em vigor nesta data e seus efeitos retroagem a 22 de setembro de 2023.

gov.br
Documento assinado digitalmente
ROBERTO VIEIRA PORDEUS
Data: 08/11/2023 08:51:47-0300
Verifique em https://validar.iti.gov.br

ROBERTO VIEIRA PORDEUS

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
CONSELHO DE ENSINO, PESQUISA E EXTENSÃO

RESOLUÇÃO Nº 57, DE 29 DE OUTUBRO DE 2024

O VICE-REITOR NA PRESIDÊNCIA DO CONSELHO DE ENSINO, PESQUISA E EXTENSÃO – CONSEPE DA UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO – UFERSA, no uso de suas atribuições legais, e tendo em vista o que estabelecem as Leis nº 8.112, de 11 de dezembro de 1990, e nº 12.772, de 28 de dezembro de 2012, e suas respectivas alterações; o Decreto nº 9.991, de 28 de agosto de 2019, alterado pelo Decreto nº 10.506, de 2 de outubro de 2020; o Regimento da Ufersa; a Resolução Consuni/Ufersa nº 003/2018, de 25 de junho de 2018; o Processo nº 23091.009655/2022-41; a deliberação deste Órgão Colegiado em sua 9ª Reunião Ordinária de 2024, realizada no dia 29 de outubro de 2024, resolve:

Art. 1º Aprovar a renovação de afastamento do servidor docente Felipe Torres Leite, pertencente ao Departamento de Engenharias e Tecnologia — DETEC, vinculado ao Centro Multidisciplinar de Pau dos Ferros – CMPF, com a finalidade de dar continuidade ao Doutorado em Ciências da Computação, na Universidade Federal de Campina Grande — UFCG, em Campina Grande – PB, no período de 22 de setembro de 2024 a 21 de setembro de 2025.

Art. 2º Esta Resolução entra em vigor nesta data e seus efeitos retroagem a 22 de setembro de 2024.

gov.br
Documento assinado digitalmente
NILDO DA SILVA DIAS
Data: 08/11/2024 06:45:57-0300
Verifique em https://validar.iti.gov.br

NILDO DA SILVA DIAS

Universidade Federal de Campina Grande– UFCG

Centro de Engenharia Elétrica e Informática - CEEI

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO - PPGCC

Formulário para Solicitação de Prorrogação de Prazo

Nome Completo:	Felipe Torres Leite
Orientador(es):	Wilkerson de Lucena Andrade
Nível:	( ) Mestrado (x) Doutorado
Tipo de Prorrogação:	(x) Prorrogação de Prazo para Defesa de Qualificação de Tese de Doutorado ( ) Prorrogação de Prazo para Defesa de Tese de Doutorado ( ) Prorrogação de Prazo para Defesa de Qualificação de Dissertação de Mestrado ( ) Prorrogação de Prazo para Defesa de Dissertação de Mestrado
Duração da Prorrogação que está solicitando (meses):	6 meses
Data da 1ª Matrícula	02/08/2022
Já obteve prorrogação para o mesmo tipo de solicitação?	( ) Não (x) Sim. Quantos meses? 6 meses
Já obteve interrupção de estudos?	(x) Não ( ) Sim. Quantos períodos? _____

Apresente abaixo justificativas para a solicitação (caso necessário, anexar documento ao processo impresso)

Venho, respeitosamente, solicitar a prorrogação do prazo para a realização da qualificação de doutorado, a fim de viabilizar minha matrícula no componente curricular de Qualificação no semestre letivo de 2025.2. No meu caso específico, esta solicitação está diretamente relacionada a questões de saúde, já informadas ao meu orientador, Prof. Dr. Wilkerson Andrade. Caso necessário, coloco-me à disposição para apresentar documentos comprobatórios adicionais, embora os já pertinentes tenham sido encaminhados ao meu orientador. Assim, solicito alguns meses adicionais para a finalização da proposta de qualificação e a realização de sua defesa, de modo a assegurar que o curso de doutorado transcorra regularmente, com a execução de todas as tarefas pendentes até sua conclusão.

Declaramos que as informações prestadas neste formulário são completas e verdadeiras.

Campina Grande, 08 de Agosto de 2025.

Assinatura do(a) aluno(a) (solicitante)	Assinatura Orientador(a) Principal
gov.br Documento assinado digitalmente FELIPE TORRES LEITE Data: 08/08/2025 09:19:50-0300 Verifique em https://validar.iti.gov.br	gov.br Documento assinado digitalmente WILKERSON DE LUCENA ANDRADE Data: 11/08/2025 15:52:21-0300 Verifique em https://validar.iti.gov.br

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CNPJ nº 05.055.128/0001-76
POS-GRADUACAO EM CIENCIA DA COMPUTACAO
Rua Aprígio Veloso, 882, Edifício Telmo Silva de Araújo, Bloco CG1, - Bairro Universitário, Campina Grande/PB, CEP 58429-900
Telefone: 2101-1122 - (83) 2101-1123 - (83) 2101-1124
Site: http://computacao.ufcg.edu.br - E-mail: secpg@computacao.ufcg.edu.br

CERTIDÃO

Processo nº 23096.051522/2025-10

Interessado: Felipe Torres Leite

Assunto: Solicitação de prorrogação de prazo para a defesa do exame de qualificação da tese de doutorado

Certificamos, para os devidos fins, que na **7ª Reunião Ordinária de 2025 do Colegiado do Curso de Pós-Graduação em Ciência da Computação**, realizada em 08 de outubro de 2025, o colegiado **aprovou**, à unanimidade, o parecer da relatora, profa. Melina Mongiovi Brito Lira, **favorável** a prorrogação de prazo por 06 (seis) meses, **contados a partir de 02 de agosto de 2025**, para a defesa do exame de qualificação da tese de doutorado, conforme consta na respectiva Ata.

Prof. Leandro Balby Marinho

Coordenador do PPGCC/UFCG

Documento assinado eletronicamente por **LEANDRO BALBY MARINHO**, **COORDENADOR(A)**, em 17/10/2025, às 09:23, conforme horário oficial de Brasília, com fundamento no art. 8º, caput, da Portaria SEI nº 002, de 25 de outubro de 2018.

[redacted]

Referência: Processo nº 23096.051522/2025-10 SEI nº 5957973

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO - UFERSA
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO - PROPPG
Av. Francisco Mota, 572 – C. Postal 137 – Bairro Pres. Costa e Silva – Mossoró – RN – CEP: 59.625-900 - Tel.: (84)3317-8296/8295 – E.mail: proppg@ufersa.edu.br

(Anexo III)

RELATÓRIO DE ATIVIDADES ACADÊMICAS

(Realizadas nos últimos 2 semestres de afastamento)

Quando se tratar do relatório referente ao 3º semestre (mestrado) e 5º semestre (doutorado), deverá ser acompanhado do **projeto de dissertação/Tese)**

No segundo ano de doutoramento, houve o cumprimento das seguintes atividades/disciplinas:

Semestre 2024.2:

- CC007: QUALIFICAÇÃO DE DOUTORADO

Semestre 2025.1:

- CC007: QUALIFICAÇÃO DE DOUTORADO

Data: 08/08/2025

gov.br Documento assinado digitalmente
FELIPE TORRES LEITE
Data: 08/08/2025 09:54:50-0300
Verifique em https://validar.iti.gov.br

Assinatura do requerente

gov.br Documento assinado digitalmente
WILKERSON DE LUCENA ANDRADE
Data: 11/08/2025 15:52:21-0300
Verifique em https://validar.iti.gov.br

Assinatura do Orientador

(Anexo IV)

RELATÓRIO DE AVALIAÇÃO DE DESEMPENHO
(Feito pelo/a orientador/a)

O doutorando FELIPE TORRES LEITE vem trabalhando no documento para a defesa da qualificação desde 2024. No entanto, ao longo desse período, a evolução do trabalho tem sido aquém do esperado para o estágio atual do programa.

É importante ressaltar que o doutorando ainda tem tempo para reverter esse quadro, desde que haja um comprometimento efetivo e mudanças concretas na postura e na organização do trabalho. A qualificação é uma etapa central no doutorado e exige um esforço contínuo, crítico e disciplinado.

Data: 11/08/2025

gov.br Documento assinado digitalmente
WILKERSON DE LUCENA ANDRADE
Data: 11/08/2025 15:52:21-0300
Verifique em https://validar.iti.gov.br

Assinatura do(a) orientador(a)

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CNPJ nº 05.055.128/0001-76
POS-GRADUACAO EM CIENCIA DA COMPUTACAO
Rua Aprígio Veloso, 882, Edifício Telmo Silva de Araújo, Bloco CG1, - Bairro Universitário, Campina Grande/PB, CEP 58429-900
Telefone: 2101-1122 - (83) 2101-1123 - (83) 2101-1124
Site: http://computacao.ufcg.edu.br - E-mail: secpg@computacao.ufcg.edu.br

DECLARAÇÃO

Processo nº 23096.073012/2025-01

Declaramos, para fins de comprovação, que **FELIPE TORRES LEITE** , matrícula [redacted], é aluno(a) regularmente matriculado(a) no curso de DOUTORADO do Programa de Pós-Graduação Stricto Sensu em Ciência da Computação da Universidade Federal de Campina Grande - UFCG, na área de CIÊNCIA DA COMPUTAÇÃO, sob a orientação de WILKERSON LUCENA DE ANDRADE. Abaixo, seguem discriminadas as informações solicitadas por essa empresa.

Data de início do curso: 02/08/2022.

Data de término do curso: Prazo máximo de 48 meses para a conclusão. Portanto, a data limite para conclusão é 02/08/2026.

Carga horária: 40 horas semanais.

Obs.: O aluno de DOUTORADO deve realizar matrícula a cada período semestral, cursar, no mínimo, 35 créditos em disciplinas e desenvolver seu projeto de pesquisa com acompanhamento direto de seu Orientador(a). Assim, o aluno deve frequentar regularmente a instituição até a conclusão do curso.

Modalidade (Presencial ou À Distância): Presencial.

Prof. Leandro Balby Marinho

Coordenador do PPGCC/UFCG

Documento assinado eletronicamente por **LEANDRO BALBY MARINHO**, **COORDENADOR(A)**, em 17/10/2025, às 15:57, conforme horário oficial de Brasília, com fundamento no art. 8º, caput, da Portaria SEI nº 002, de 25 de outubro de 2018.

[redacted]

Universidade Federal de Campina Grande - UFCG

Centro de Engenharia Elétrica e Informática - CEEI

Programa de Pós-Graduação em Ciência da Computação - PPGCC

HISTÓRICO ESCOLAR - DOUTORADO

Nome: FELIPE TORRES LEITE

Matrícula: [redacted] **Identidade:** [redacted] **Órgão Expedidor:** ITEP

[redacted]ulo - SP

Curso Anterior: Graduação em Ciência da Computação **Conclusão:** 25/04/2013

Código	Nome	Créd.	CH	Período	Nota	Conceito	Situação
	EQUIVALÊNCIA AO TÍTULO DE MESTRE	22	330				APROVADO
CC003	ENGENHARIA DE SOFTWARE	4	60	2022.2	9,4		APROVADO
CC004	ESTÁGIO DOCÊNCIA I (APROVEITAMENTO DE ESTUDOS)	2	60	2022.2	APROVADO		APROVADO
CC005	ESTÁGIO DOCÊNCIA II (APROVEITAMENTO DE ESTUDOS)	2	60	2022.2	APROVADO		APROVADO
CC112	TÓPICOS ESPECIAIS EM CIÊNCIA DA COMPUTAÇÃO (APRENDIZAGEM DE MÁQUINA)	4	60	2022.2	8.5		APROVADO
INF 084.2	TÓPICOS ESPECIAIS EM CIÊNCIA DA COMPUTAÇÃO (ARQUITETURA DE SOFTWARE)	4	60	2022.2	9,7		APROVADO
69883	PROFICIÊNCIA EM LÍNGUA ESTRANGEIRA (INGLÊS)*	0	0	2022.2			APROVADO
INF106	FUNDAMENTOS DE PESQUISA EM CIÊNCIA DA COMPUTAÇÃO I	4	60	2023.1	8,5		APROVADO
INF107	FUNDAMENTOS DE PESQUISA EM CIÊNCIA DA COMPUTAÇÃO II	4	60	2023.1	7,7		APROVADO
INF108	FUNDAMENTOS DE PESQUISA EM CIÊNCIA DA COMPUTAÇÃO III	4	60	2023.1	9.4		APROVADO
CC001	PROJETO DE PESQUISA (ENGENHARIA DE SOFTWARE INTELIGENTE I)	2	30	2023.2	9,0		APROVADO
CC001	PROJETO DE PESQUISA (ENGENHARIA DE SOFTWARE INTELIGENTE II)	2	30	2024.1	9,0		APROVADO
INF066	PROFICIÊNCIA EM LÍNGUA ESTRANGEIRA (ESPANHOL)**	0	0	2024.1			APROVADO
CC001	PROJETO DE PESQUISA (ESTUDO EXPERIMENTAL EM ENGENHARIA DE SOFTWARE I)	2	30	2024.2	9,0		APROVADO
CC007	QUALIFICAÇÃO DE DOUTORADO	0	0	2024.2			MATRICULADO
CC007	QUALIFICAÇÃO DE DOUTORADO	0	0	2025.1			MATRICULADO
CC007	QUALIFICAÇÃO DE DOUTORADO	0	0	2025.2			MATRICULADO

Av. Aprígio Veloso, 882 - 58429-140, Campina Grande, PB
Fone: (83) 2101-1124 - Fax: (83) 2101-1123 - Email: ppgcc@copin.ufcg.edu.br

Observações:

* Aproveitamento decorrente do título de mestre.

** Homologado pelo colegiado em 04.06.2024.

Créditos Integralizados para o DOUTORADO: 56

Data da admissão: 02/08/2022

Título da Tese: ""

Linha de Pesquisa:

Área de Concentração: CIÊNCIA DA COMPUTAÇÃO

Orientador(es): WILKERSON DE LUCENA ANDRADE

Bolsa de Estudo:
Órgão Financiador: CAPES **Início da Bolsa:** 01/11/2023 **Término da Bolsa:** 31/08/2026

Banca Examinadora:

Situação: DOUTORANDO EM CIÊNCIA DA COMPUTAÇÃO
CRA: 8.89
Carga Horário Obtida: 900 horas.
Obs.: Cada crédito corresponde a 15 horas/aula.

Programa de Pós-Graduação em Ciência da Computação - PPGCC
Campina Grande, 17 de Outubro de 2025.

Av. Aprígio Veloso, 882 - 58429-140, Campina Grande, PB
Fone: (83) 2101-1124 - Fax: (83) 2101-1123 - Email: ppgcc@copin.ufcg.edu.br

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO - UFERSA
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO - PROPPG
Av. Francisco Mota, 572 – C. Postal 137 – Bairro Pres. Costa e Silva – Mossoró – RN – CEP: 59.625-900 - Tel.: (84)3317-8296/8295 – E.mail: proppg@ufersa.edu.br

(Anexo VIII)

TERMO DE DECLARAÇÃO E COMPROMISSO

EU, Felipe Torres Leite, portador do CPF nº [redacted] RG nº [redacted], matrícula siape nº [redacted], devidamente autorizado(a) pela Universidade Federal Rural do Semi-Árido – UFERSA para realizar o curso de o doutorado em Ciência da Computação na Universidade Federal de Campina Grande (UFCG), pelo Programa de Pós-Graduação em Ciência da Computação (PPGCC), pelo presente e na melhor forma de direito, conforme a Lei nº 8.112/90, em seu Artigo 96-A, o Regimento Geral da UFERSA, em seu Artigo 338, e a RESOLUÇÃO CONSUNI/UFERSA Nº 003/2018, de 25 de junho de 2018, assumo o compromisso formal de permanecer, obrigatoriamente a serviço da UFERSA, por tempo integral e com dedicação exclusiva por um prazo igual ao do afastamento, a contar da conclusão do referido curso, sob pena de ressarcimento de todas as despesas, diretas ou indiretas em que a mesma tenha incorrido financiando aquele curso, tais como: salários, gratificações, passagens, diárias, ajudas de custo, bolsa de complementação salarial, bolsa de estudos, custos de matrícula, mensalidades e anuidades, enfim, qualquer dispêndio feito pela União, através da sua administração direta ou indireta, centralizada ou descentralizada, com o fim de custeio do curso em epígrafe.
Declaro estar ciente das Normas e Regulamentos do Curso.
Fica eleito o foro da Justiça Federal, Seção Judiciária do Rio Grande do Norte para dirimir todas as questões porventura decorrentes deste instrumento.

Pau dos Ferros (RN), 17 de Outubro de 2025.

gov.br Documento assinado digitalmente
FELIPE TORRES LEITE
Data: 17/10/2025 09:05:37-0300
Verifique em https://validar.iti.gov.br

Assinatura

gov.br Documento assinado digitalmente
FRANCISCO CARLOS GURGEL DA SILVA SEGUNI
Data: 18/10/2025 09:58:38-0300
Verifique em https://validar.iti.gov.br

Francisco Carlos Gurgel da Silva Segundo
[redacted]

gov.br Documento assinado digitalmente
WAGNA MAQUIS CARDOSO DE MELO GONCALV
Data: 17/10/2025 09:39:29-0300
Verifique em https://validar.iti.gov.br

Wagna Maquis Cardoso de Melo Gonçalves
[redacted]

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO - UFERSA
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO - PROPPG
Av. Francisco Mota, 572 – C. Postal 137 – Bairro Pres. Costa e Silva – Mossoró – RN – CEP: 59.625-900 - Tel.: (84)3317-8296/8295 – E.mail: proppg@ufersa.edu.br

(Anexo IX)

PARECER DA CHEFIA IMEDIATA

(Departamento Acadêmico de lotação do requerente)

(Obrigatório)

Pode utilizar documento oficial do setor (Departamento) em que o solicitante esteja vinculado dispensando este formulário.

Data: ___/___/____

_____

Assinatura do Chefe imediato

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO - UFERSA
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO - PROPPG
Av. Francisco Mota, 572 – C. Postal 137 – Bairro Pres. Costa e Silva – Mossoró – RN – CEP: 59.625-900 - Tel.: (84)3317-8296/8295 – E.mail: proppg@ufersa.edu.br

(Anexo X)

PARECER DO CONSELHO DO CENTRO AO QUAL O REQUERENTE FAZ PARTE

(Obrigatório)

Pode utilizar documento oficial do CONSELHO DO CENTRO em que o solicitante esteja vinculado dispensando este formulário.

Data: ___/___/____

_____

Assinatura do presidente do Conselho de Centro

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
ASSESSORIA ESPECIAL

DECLARAÇÃO Nº 196 / 2025 - ASESP (11.01.14)

Nº do Protocolo: 23091.015202/2025-31

Mossoró-RN, 17 de outubro de 2025.

DECLARAÇÃO

Declaramos, para os fins que se fizerem necessários, que o(a) servidor(a) **Felipe Torres Leite**, matrícula Siape Nº [redacted], ocupante do cargo de **Professor de Magistério Superior**, não sofreu penalidades administrativas nos últimos 05 (cinco) anos, e não possui, até a presente data, registro de ter respondido à Processo Administrativo Disciplinar no Sistema de Gestão de Processos Disciplinares (CGU-PAD), nos termos da Lei nº 8.112/90, que dispõe sobre o Regime Jurídico Único dos servidores públicos civis da União.

(Assinado digitalmente em 17/10/2025 08:22)
MARIA DA GLORIA DA SILVA
ASSESSOR ESPECIAL
ASESP (11.01.14)
Matrícula: ###609#0

Visualize o documento original em https://sipac.ufersa.edu.br/public/documentos/index.jsp informando seu número: **196**, ano: **2025**, tipo: **DECLARAÇÃO** [redacted]

NECESSIDADES DE DESENVOLVIMENTO QUALIFICAÇÃO

68	Cursos de qualificação vinculados à Grande Área do Conhecimento CIÊNCIAS HUMANAS;
69	Cursos de qualificação vinculados à Grande Área do Conhecimento MULTIDISCIPLINAR;
70	Ampliar conhecimentos relacionados à grande área LINGUÍSTICA, LETRAS E ARTES;
71	Ampliar conhecimentos relacionados à grande área CIÊNCIAS SOCIAIS APLICADAS;
72	Ampliar conhecimentos relacionados à grande área CIÊNCIAS AGRÁRIAS;
73	Ampliar conhecimentos relacionados à grande área CIÊNCIAS DA SAÚDE;
74	Ampliar conhecimentos relacionados à grande área ENGENHARIAS;
75	Ampliar conhecimentos relacionados à grande área CIÊNCIAS BIOLÓGICAS;
76	Ampliar conhecimentos relacionados à grande área CIÊNCIAS EXATAS E DA TERRA;
77	Aprimorar a capacidade de realização de pesquisas científicas aplicadas as CIÊNCIAS VETERINÁRIAS, com a utilização de novas técnicas e metodologias;
78	Aprimorar a capacidade de realização de pesquisas científicas aplicadas as CIÊNCIAS AGRÁRIAS, com a utilização de novas técnicas e metodologias;

REQUERIMENTO Nº 2/2025 - DETEC (11.01.36.12.08)

(Nº do Protocolo: NÃO PROTOCOLADO)

(Assinado digitalmente em 21/11/2025 12:13)
JONAS FIRMINO FILHO
SECRETARIO EXECUTIVO
PAUDOSFERROS (11.01.36)
Matrícula: ###390#5

*Visualize o documento original em https://sipac.ufersa.edu.br/documentos/ informando seu número: **2**, ano: **2025**, tipo: **REQUERIMENTO**, data de emissão: **21/11/2025** e o código de verificação: **ad56f5c840***

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO

DESPACHO Nº 52/2025 - PROPPG (11.01.03)

Nº do Protocolo: NÃO PROTOCOLADO

Mossoró-RN, 24 de novembro de 2025.

Tendo em vista o art. 3º, o art. 15 e o art. 21 da Resolução Consuni/Ufersa nº 003/2018, de 25 de junho de 2018, e considerando os pareceres favoráveis do Centro e do Departamento ao qual o(a) servidor(a) **Felipe Torres Leite** faz parte, a Pró-Reitoria de Pesquisa e Pós-Graduação emite **parecer favorável** após a análise do mérito. Encaminhe-se o processo à Pró-Reitoria de Gestão de Pessoas – PROGEPE para apreciação e deliberação.

(Assinado digitalmente em 24/11/2025 16:48)
LIANA HOLANDA NEPOMUCENO NOBRE
PRO-REITOR(A)
PROPPG (11.01.03)
Matrícula: ###689#4

Processo Associado: 23091.009655/2022-41

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
SETOR DE CAPACITAÇÃO E APERFEIÇOAMENTO

DESPACHO Nº 381/2025 - SCA (11.01.04.04.02)

Nº do Protocolo: NÃO PROTOCOLADO

Mossoró-RN, 15 de dezembro de 2025.

1 - Trata-se de requerimento de renovação de afastamento integral formulado pelo servidor docente **Felipe Torres Leite**, SIAPE nº [redacted], ocupante do cargo de Professor do Magistério Superior, lotado no Departamento de Engenharias e Tecnologia - DETEC, vinculado ao Centro Multidisciplinar de Pau dos Ferros - CMPF, com a finalidade de dar continuidade ao Doutorado em Ciências da Computação, na Universidade Federal de Campina Grande - UFCG, em Campina Grande - PB, no período de 22 de setembro de 2025 a 02 de agosto de 2026.

2 - Considerando o disposto no Art. 19 da Resolução 03/2018/CONSUNI, de 25 de junho de 2018, o qual declara que:

> Art. 19. A renovação de afastamento para qualificação em nível de pós-graduação stricto sensu ou estágio pós-doutoral dar-se-á nos termos da legislação em vigor, devendo a manifestação de intenção de renovação do afastamento ser protocolada em até 60 (sessenta) dias antes do término do afastamento.

4 - Considerando que o último período de afastamento total do servidor interessado encerrou-se em 21 setembro de 2022, conforme Portaria nº 2.179, de 26 de dezembro de 2024;

5 - Considerando que a data de protocolo da solicitação de renovação do afastamento se deu no dia 23 de outubro de 2025, conforme documento retirado do processo devido constar anexos que continham informações sensíveis protegidas pelo sigilo médico e pela Lei Geral de Proteção de Dados (LGPD), de acordo com o encaminhamento nº 32 / 2025 - SCA (doc. 38);

6 - Considerando as manifestações favoráveis do Departamento de Engenharias e Tecnologia - DETEC, do Centro Multidisciplinar de Pau dos Ferros - CMPF e da Pró-Reitoria de Pesquisa e Pós-Graduação - PROPPG, constantes dos documentos nº 36, 37 e 41 dos autos;

7 - Verifica-se, entretanto, que o pedido de renovação foi apresentado em desconformidade com o prazo estabelecido no art. 19 da Resolução nº 03/2018/CONSUNI, configurando o não atendimento de requisito formal indispensável à concessão da renovação do afastamento, razão pela qual, em observância ao princípio da legalidade, manifesta-se pelo **INDERERIMENTO** do pleito.

8 - Diante do exposto, encaminhem-se os autos à Comissão Permanente de Pessoal Docente - CPPD, para apreciação e deliberação.

(Assinado digitalmente em 15/12/2025 17:11)
JOSIMAR CARDOSO DE QUEIROZ
DIRETOR
DDP (11.01.04.04)
Matrícula: ###359#8

(Assinado digitalmente em 16/12/2025 15:19)
PRISCCILA SOUZA DE MENEZES
ASSISTENTE EM ADMINISTRACAO
SCA (11.01.04.04.02)
Matrícula: ###182#6

Processo Associado: 23091.009655/2022-41

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
COMISSÃO PERMANENTE DE PESSOAL DOCENTE

DESPACHO Nº 4/2026 - CPPD (11.01.26)

Nº do Protocolo: NÃO PROTOCOLADO

Mossoró-RN, 19 de janeiro de 2026.

Analisando a solicitação constante neste processo administrativo feita pelo servidor docente Felipe Torres Leite, matrícula Siape nº [redacted], de renovação do afastamento integral para continuidade do Doutorado em Ciências da Computação na Universidade Federal de Campina Grande – UFCG, no período de 22 de setembro de 2025 a 02 de agosto de 2026, e considerando:

- A documentação anexa, que demonstra o atendimento aos requisitos institucionais e a justificativa para o pedido fora do prazo;
- O Despacho nº 11/2025 - DETEC, que aprovou o afastamento na 7ª Assembleia Extraordinária do Departamento de Engenharias e Tecnologia, realizada em 04 de novembro de 2025, destacando a disponibilidade de professor substituto e as justificativas apresentadas para o atraso na solicitação;
- O Parecer favorável do Centro Multidisciplinar de Pau dos Ferros (CMPF), emitido em 07 de novembro de 2025, após apreciação na 5ª Reunião Extraordinária do Conselho de Centro, aprovando por unanimidade a renovação;
- O Despacho nº 52/2025 - PROPPG, que emitiu parecer favorável após análise do mérito acadêmico;
- O Despacho nº 381/2025 - SCA, que apontou o descumprimento do prazo estabelecido no art. 19 da Resolução CONSUNI/UFERSA nº 003/2018 e opinou pelo indeferimento, encaminhando em seguida o processo para apreciação desta Comissão;

Considerando ainda:

- As justificativas apresentadas pelo docente, correspondentes às circunstâncias excepcionais que justificam o atraso na solicitação, como a mudança de orientador, questões de saúde do servidor e a tramitação administrativa no programa de pós-graduação;
- A garantia de substituição docente, mediante contratação de professor substituto, sem prejuízo às atividades acadêmicas;
- A relevância da qualificação para a instituição e para a formação discente, bem como o compromisso do servidor com o retorno e a aplicação dos conhecimentos adquiridos;

esta Comissão Permanente de Pessoal Docente (CPPD), após análise das justificativas apresentadas e considerando o caráter excepcional do caso, posiciona-se a favor da referida solicitação.

Encaminhe-se este processo à Secretaria dos Órgãos Colegiados para apreciação e deliberação pelo Conselho Superior competente.

(Assinado digitalmente em 19/01/2026 10:56)
LUCIANA VIEIRA DE PAIVA
PROFESSOR 3 GRAU
CCBS (11.01.00.07)
Matrícula: ###692#5

Processo Associado: 23091.009655/2022-41

4, ano: **2026**, tipo: **DI** [redacted]

Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da Computação

Uma Abordagem Experimental Híbrida para a Detecção de Code Smells em TypeScript

Felipe Torres Leite

Proposta de Tese submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande como parte dos requisitos necessários para obtenção do grau de Doutor em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Engenharia de Software

Wilkerson L. Andrade
(Orientador)

Campina Grande, Paraíba, Brasil

Resumo

A detecção automática de *Code Smells* (*CS*) constitui um elemento fundamental para mitigar a dívida técnica e aprimorar a manutenibilidade de sistemas de software. Embora ferramentas clássicas de análise estática, como SonarQube, ESLint e *typescript-eslint*, sejam amplamente utilizadas na indústria, persistem limitações relevantes, incluindo sensibilidade limitada ao contexto semântico, dependência de heurísticas rígidas, cobertura limitada, especialmente para *CS* de natureza estrutural em ecossistemas baseados em JavaScript/TypeScript, e predominância de estudos focados no ecossistema Java. Nesse cenário, *Large Language Models* (LLMs) emergem como uma alternativa promissora devido à sua capacidade de interpretar relações estruturais e semânticas no código-fonte. Entretanto, a literatura ainda carece de avaliações sistemáticas, padronizadas e reprodutíveis que comparem, de forma rigorosa, o desempenho desses modelos às ferramentas tradicionais, especialmente no contexto da linguagem TypeScript.

Esta proposta de tese conduz um estudo empírico abrangente para investigar o potencial dos LLMs na detecção de seis *CS* clássicos definidos por Fowler — *Parallel Inheritance Hierarchies*, *Lazy Class*, *Middle Man*, *Alternative Classes with Different Interfaces*, *Incomplete Library Class* e *Refused Bequest* — em projetos reais de código aberto desenvolvidos em TypeScript, totalizando mais de uma centena de repositórios públicos e aproximadamente 700 mil linhas de código. A metodologia foi estruturada em três fases, envolvendo um mapeamento sistemático da literatura, a análise das ferramentas existentes, a construção de um *dataset* rotulado, integrado a um protocolo experimental reprodutível, por meio de dupla revisão independente. Esse conjunto foi empregado em experimentos comparativos com múltiplos LLMs (GPT-4o, Claude 3.5 e Gemini 1.0), apoiados em análises exploratórias, estatísticas descritivas e avaliação qualitativa das ocorrências detectadas. O estudo também incorpora análises de consistência, de variabilidade entre *prompts* e de alinhamento às refatorações propostas por Fowler.

Os resultados preliminares evidenciam em quais condições os LLMs ampliam a cobertura de detecção em relação às ferramentas tradicionais, em quais cenários permanecem limitações relevantes e como modelos de linguagem podem complementar a análise estática no contexto de uma abordagem híbrida de detecção em *pipelines* reais de desenvolvimento.

A consolidação desses achados, por meio da ampliação dos experimentos, da validação empírica e do refinamento do *benchmark*, deverá subsidiar a formulação de diretrizes práticas e de evidências sistematizadas para pesquisadores e engenheiros de software ao longo das etapas subsequentes da pesquisa.

Abstract

The automatic detection of Code Smells (CS) is fundamental to mitigating technical debt and improving the maintainability of software systems. Although classical static analysis tools, such as SonarQube, ESLint, and *typescript-eslint*, are widely adopted in industrial practice, relevant limitations persist, including limited sensitivity to semantic context, reliance on rigid heuristics, restricted coverage, especially for structurally-oriented CS in JavaScript/TypeScript ecosystems, and a predominance of studies focused on the Java ecosystem. In this context, Large Language Models (LLMs) have emerged as a promising alternative due to their ability to interpret structural and semantic relationships in source code. Nevertheless, the literature still lacks systematic, standardized, and reproducible evaluations that rigorously compare the performance of these models with traditional tools, particularly in the context of TypeScript.

This thesis proposal conducts a comprehensive empirical study to investigate the potential of LLMs in detecting six classical CS defined by Fowler — Parallel Inheritance Hierarchies, Lazy Class, Middle Man, Alternative Classes with Different Interfaces, Incomplete Library Class, and Refused Bequest — in real-world open-source projects developed in TypeScript, comprising more than one hundred public repositories and approximately 700,000 lines of code. The methodology is structured into three phases, involving a systematic mapping of the literature, an analysis of existing tools, and the construction of a labeled dataset integrated into a reproducible experimental protocol through independent double review. This dataset is employed in comparative experiments with multiple LLMs (GPT-4o, Claude 3.5, and Gemini 1.0), supported by exploratory analyses, descriptive statistics, and qualitative assessment of detected occurrences. The study also incorporates analyses of consistency, prompt variability, and alignment with Fowler's refactoring principles.

Preliminary results indicate the conditions under which LLMs extend detection coverage relative to traditional tools, the scenarios in which relevant limitations persist, and how language models can complement traditional static analysis within a hybrid detection framework in real-world development pipelines. The consolidation of these findings, through the expansion of experiments, empirical validation, and refinement of the benchmark, is

expected to support the formulation of practical guidelines and systematized evidence for researchers and software engineers in subsequent stages of the research.

Conteúdo

Lista de Siglas e Abreviaturas

API – Application Programming Interface

AST – Abstract Syntax Tree

CNN – Convolutional Neural Network

CS – Code Smell(s)

ES – Engenharia de Software

EMS – Estudo de Mapeamento Sistemático

EVM – Ethereum Virtual Machine

GRU – Gated Recurrent Unit

GWCS – Gas-Wasting Code Smell(s)

IA – Inteligência Artificial

IAG – Inteligência Artificial Generativa

IQR – Intervalo Interquartil

LOC – Lines Of Code

LLM – Large Language Model

LSTM – Long Short-Term Memory

MCP – Model Context Protocol

PLN – Processamento de Linguagem Natural

QP – Questão de Pesquisa

RSL – Revisão Sistemática da Literatura

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Os *code smells* constituem indícios de problemas de projeto que, embora não comprometam diretamente a funcionalidade do software, elevam a dívida técnica e dificultam as atividades de manutenção ao longo do ciclo de vida [32]. A literatura [7, 61] os caracteriza como "sintomas" de questões estruturais mais profundas, cuja detecção orienta ações de refatoração voltadas ao aprimoramento da legibilidade, à redução da complexidade e ao fortalecimento do *design*.

Nesse âmbito, o catálogo clássico proposto por Fowler *et al.* [33] reúne definições consolidadas, princípios práticos e exemplos recorrentes (*e.g.* *God Class*, *Shotgun Surgery*, *Lazy Class*, *Middle Man* e *Divergent Change*) que se tornaram referência para estudos posteriores [32, 20, 66, 7, 61, 68]. Termos correlatos, como *code anomaly*, *bad smell*, *anti-pattern* e *design smell*, também aparecem na literatura para descrever fenômenos semelhantes [20, 52, 53, 76, 50]. Nesta proposta, adota-se preferencialmente a terminologia *Code Smell* (*CS*), em consonância com a definição original de Fowler *et al.* [33], posteriormente atualizada por Fowler [32].

Diversos fatores contribuem para a introdução desses problemas no código, abrangendo dimensões humanas, organizacionais, de requisitos e tecnológicas. Entre eles destacam-se: (i) falta de habilidade ou de consciência por parte dos desenvolvedores; (ii) mudanças frequentes de requisitos; (iii) restrições impostas pela linguagem; e (iv) pressão por prazos [66]. Esses fatores reforçam a necessidade de mecanismos de apoio que permitam a detecção precoce de indícios de degradação da arquitetura interna do código.

Nesse cenário, ferramentas automatizadas de análise estática baseadas em regras e heu-

rísticas, tais como SonarQube, ESLint, typescript-eslint e Embold, tornaram-se amplamente utilizadas em projetos contemporâneos [7, 13, 20, 51, 65, 69, 77]. O SonarQube provê verificação multilinguagem, incluindo JavaScript e TypeScript, e contempla dimensões de manutenibilidade, confiabilidade e segurança [73, 13, 20].

Já o ESLint consolidou-se como o linter mais difundido no ecossistema JavaScript, especialmente devido a sua extensibilidade por meio de plugins e ao suporte expressivo da comunidade [28, 77]. Como o ESLint não interpreta tipos de forma nativa, o *typescript-eslint* integra um parser e acesso ao sistema de tipos para possibilitar regras específicas à linguagem [78].

A plataforma Embold, por sua vez, fornece recursos para detecção de antipadrões [24], incluindo *God Class*, *Shotgun Surgery* e *Feature Envy*, além de métricas e limiares configuráveis para monitoramento contínuo da qualidade em múltiplas linguagens, entre elas TypeScript [24, 25, 69, 51]. É relevante destacar que, não obstante a documentação oficial da plataforma [24, 25] adote a nomenclatura "antipadrão", suas definições permanecem substancialmente ancoradas nos conceitos consolidados por Fowler *et al.* [33, 32].

Apesar de sua relevância, a literatura evidencia limitações persistentes nessas ferramentas de detecção de *CS*. Em geral, tais abordagens baseiam-se em heurísticas rígidas e apresentam restrições recorrentes quanto à cobertura contextual e à interpretação semântica de estruturas complexas [7, 20, 65]. *Smells* de nível de *design* — derivados de relações interclasse ou intermódulo, responsabilidades difusas ou acoplamentos estruturais — tendem a ser menos sinalizados do que problemas sintáticos ou idiomáticos [7, 61].

Essa lacuna resulta, na prática, em falsos negativos para padrões arquiteturais, como o *CS Middle Man* (delegação ociosa), ou para hierarquias acopladas, como o *CS Parallel Inheritance Hierarchies*, cuja identificação requer compreensão da intenção, dos papéis e das colaborações entre componentes [20, 7]. Considerando que *CS* impactam a compreensibilidade, a manutenibilidade e a produtividade [61, 7, 66], tais limitações comprometem a priorização de refatorações e favorecem o acúmulo de dívida técnica ao longo da evolução do software.

1.1 Problemática

As limitações existentes nas abordagens automatizadas de detecção de *CS* compõem um *gap* persistente que afeta tanto a prática industrial quanto a pesquisa acadêmica [66]. De modo geral, destacam-se: (i) cobertura parcial de tipos; (ii) foco excessivo em casos clássicos já consolidados na literatura; (iii) baixa sensibilidade a relações semânticas complexas; e (iv) viés de linguagem, com maior suporte para Java e menor cobertura para linguagens como C, C++, JavaScript e TypeScript. Tais restrições resultam em subdetecção sistemática de sinais estruturais relevantes, especialmente os associados a responsabilidades distribuídas ou a acoplamentos arquiteturais.

Nesse âmbito, *Large Language Models* (LLMs) surgem como alternativa promissora para a detecção de *CS*, uma vez que conseguem modelar contexto semântico, relações interclasse e intenções subjacentes ao comportamento do código por meio de arquiteturas de transformadores e de mecanismos de atenção [31, 75]. Esses modelos, amplamente empregados em tarefas de linguagem e de Engenharia de Software (ES), como, por exemplo, geração e explicação de código, preenchimento automático e correção de *bugs*, têm sido explorados em estudos recentes [21, 12].

Ademais, podem ser especializados por meio de estratégias de engenharia de *prompt*, seja em formato genérico ou detalhado, conforme evidenciado por investigações empíricas que analisam a sensibilidade de LLMs ao tipo e ao conteúdo do *prompt* [63, 17, 84]. Essa flexibilidade reforça o potencial desses modelos para identificar indícios estruturais alinhados aos princípios de refatoração de Fowler [32].

Estudos preliminares [31, 75, 21, 12, 63, 17, 84] indicam que o desempenho de LLMs em tarefas de análise de código é fortemente influenciado pelo tipo, pelo conteúdo e pelo nível de detalhamento do *prompt*, bem como pela complexidade estrutural do módulo analisado. Resultados iniciais mostram melhorias quando *prompts* mais específicos são empregados, sobretudo em modelos proprietários como o ChatGPT, evidenciando a influência do domínio fornecido na identificação de padrões arquiteturais complexos.

Apesar desses avanços, revisões recentes [68, 81, 37] evidenciam lacunas relevantes, como a ausência de *benchmarks* padronizados, a predominância de estudos centrados em Java e a escassez de avaliações reprodutíveis em diferentes ecossistemas de desenvolvi-

mento. Tais limitações dificultam comparações robustas e comprometem a generalização dos resultados.

No escopo específico da detecção de *CS* e do apoio à refatoração, estudos recentes [81, 12, 63, 41] exploram estratégias variadas de *prompting* e relatam divergências expressivas entre categorias de *smells*, métricas adotadas e linguagens avaliadas. Persistem, contudo, lacunas significativas: (i) a ausência de *benchmarks* comparáveis; (ii) a baixa atenção dedicada ao ecossistema TypeScript; e (iii) a insuficiência de análises empíricas controladas em comparação com plataformas de análise estática tradicionais [68, 31, 37]. Esses fatores configuram um panorama em que a efetividade de LLMs na detecção de *CS* permanece em aberto, o que justifica a condução de estudos sistemáticos e comparativos.

1.2 Proposta de Solução

Diante das limitações persistentes das abordagens tradicionais de análise estática, notadamente a dependência de heurísticas rígidas, a cobertura parcial de tipos, a baixa sensibilidade a relações semânticas e os vieses de linguagem, e considerando a ainda incipiente validação sistemática da eficácia de LLMs na detecção de *CS*, esta proposta de tese tem como objetivo avançar a identificação de *CS* no domínio da linguagem TypeScript por meio da investigação e sistematização de novas abordagens de detecção, combinando, de forma integrada, técnicas clássicas de análise estática e modelos baseados em LLMs.

Esse avanço é perseguido por meio da investigação sistemática de novas abordagens para a detecção de *CS*, indo além da aplicação isolada de técnicas existentes e explorando mecanismos capazes de ampliar a sensibilidade semântica e estrutural da análise em TypeScript.

Como etapa inicial dessa investigação, foi conduzido um estudo exploratório envolvendo mais de 100 projetos públicos em TypeScript, com o objetivo de validar o protocolo experimental, avaliar a viabilidade operacional da abordagem proposta e gerar evidências empíricas preliminares sobre o comportamento das técnicas analisadas. Os resultados dessa fase inicial subsidiam os refinamentos metodológicos e fundamentam a ampliação sistemática dos experimentos apresentados ao longo desta proposta de tese.

Como contribuição metodológica central desta proposta, prevê-se a construção de um *benchmark* reprodutível para a detecção de *CS* em TypeScript, concebido como infraestru-

tura experimental para a investigação, validação e comparação sistemática de abordagens tradicionais e baseadas em LLMs. Esse *benchmark* será composto por um conjunto de dados rotulado, protocolos e instrumentos experimentais, permitindo avaliar, de forma controlada, o desempenho relativo e o potencial de complementaridade entre as abordagens, tomando como referência a taxonomia de Fowler [32].

Para tanto, será definida previamente uma lista de *CS* de interesse, acompanhada de critérios operacionais ajustados às particularidades da linguagem TypeScript. A unidade de análise adotada corresponderá ao módulo de código (arquivo `.ts` ou submódulo coeso), possibilitando a identificação de ocorrências em linhas específicas e a documentação das características pertinentes a cada instância observada.

O estudo empregará um conjunto de dados curado, composto por trechos de código extraídos de repositórios públicos e organizados de modo a contemplar tanto exemplos com ocorrência dos *CS* selecionados quanto casos isentos dessas manifestações. A rotulagem do conjunto de dados obedecerá a um protocolo de dupla revisão independente, incluindo o cálculo da concordância entre avaliadores e a arbitragem em casos de divergência. Esse arranjo metodológico visa garantir consistência, rastreabilidade e confiabilidade ao longo da elaboração das anotações. Esse conjunto anotado constituirá um *benchmark* em TypeScript para *CS*, concebido para apoiar experimentos reprodutíveis e análises comparáveis no domínio.

Como parte da investigação empírica apoiada por esse *benchmark*, serão aplicadas duas famílias de detectores: (i) ferramentas tradicionais (SonarQube, ESLint e *typescript-eslint*) operando sob configurações previamente fixadas; e (ii) LLMs modernos (GPT-4, Claude 3.5 e Gemini 1.0) executados em ambiente controlado, com parâmetros definidos e *prompts* padronizados, em consonância com evidências que destacam a sensibilidade desses modelos ao tipo, ao conteúdo e ao nível de detalhamento do *prompt* [68, 63].

A avaliação será operacionalizada por meio de métricas de Precisão, *Recall* e F1 por tipo de *CS*, acompanhadas de intervalos de confiança obtidos por *bootstrap*, de modo a assegurar robustez na interpretação dos resultados. Adicionalmente, as ameaças à validade (interna, externa, de construção e de conclusão) serão explicitadas e mitigadas mediante a documentação integral dos *prompts*; da manutenção estável dos parâmetros de execução de cada modelo (incluindo versão, temperatura e demais configurações); e da disponibilização pública dos artefatos produzidos, reforçando a rastreabilidade, a auditabilidade e a replicabilidade do

benchmark e dos experimentos.

Com essa investigação, pretende-se aprimorar a identificação de *CS* em TypeScript, produzindo evidências empíricas consistentes sobre as limitações e as potencialidades de uma abordagem híbrida integrada entre técnicas tradicionais de análise estática e modelos baseados em LLMs. Em paralelo, a proposta visa entregar um *benchmark* reprodutível e diretrizes práticas para a incorporação dessas abordagens em *pipelines* de qualidade, esclarecendo *se*, *quando* e *como* tais modelos podem ampliar as capacidades de detecção e apoiar refatorações [68, 32].

Dessa forma, a proposta não se limita a uma comparação de desempenho entre técnicas existentes, mas busca produzir conhecimento novo sobre como *CS* podem ser identificados de forma mais eficaz em TypeScript, a partir da combinação de critérios operacionais, evidências empíricas e novas formas de exploração semântica do código.

À luz dessa proposta, apresentam-se, a seguir, o objetivo geral (1.2.1) e os objetivos específicos (1.2.2) que estruturam a investigação, orientam a execução das etapas metodológicas e delimitam o escopo analítico adotado neste trabalho.

1.2.1 Objetivo Geral

Investigar, sistematizar e avançar a identificação de *CS* em projetos desenvolvidos em TypeScript por meio da proposição e validação de uma abordagem experimental baseada na sistematização de critérios operacionais, na construção de um *benchmark* e na integração de técnicas tradicionais e LLMs tomando como referência a taxonomia de Fowler [32].

A proposta visa caracterizar as limitações das técnicas de detecção atualmente empregadas; analisar o potencial dos LLMs na identificação de *CS* que dependem de contexto semântico e estrutural; e derivar diretrizes para a integração entre abordagens clássicas e baseadas em modelos de linguagem, com foco no aprimoramento da detecção e no apoio à refatoração no ecossistema TypeScript.

1.2.2 Objetivos Específicos

Para viabilizar o alcance do objetivo geral, os objetivos específicos organizam o conjunto de etapas que orientaram o desenvolvimento do estudo, assegurando clareza metodológica e

coerência entre as fases da pesquisa.

1. Caracterizar as capacidades e limitações das ferramentas tradicionais de detecção de *CS* na linguagem TypeScript.
2. Delimitar o escopo dos *CS* investigados e formalizar critérios operacionais com base na taxonomia de Fowler.
3. Revisar o estado da arte sobre a aplicação de LLMs em tarefas de análise de código, com ênfase na detecção de *CS*.
4. Construir um conjunto de dados rotulado, composto por trechos de código em TypeScript, conforme o protocolo estabelecido, constituindo um *benchmark* para experimentos de detecção de *CS*.
5. Analisar e comparar, com métricas padronizadas, os resultados de detecção gerados por LLMs e por ferramentas tradicionais de análise estática.
6. Propor e analisar uma abordagem baseada em LLMs para apoiar a detecção e a refatoração de *CS* em TypeScript, fundamentada em evidências empíricas e na integração com técnicas tradicionais.
7. Disponibilizar publicamente os artefatos derivados do *benchmark* proposto (incluindo *datasets*, *prompts*, *scripts* e documentação), de modo a assegurar a reprodutibilidade, a transparência e o uso por investigações futuras.
8. Conduzir um processo sistemático de validação dos achados, considerando as ameaças à validade e as respectivas estratégias de mitigação.

Tais objetivos fundamentam a formulação das Questões de Pesquisa (QPs) que orientam o percurso investigativo desta proposta.

1.2.3 Questões de Pesquisa (QPs)

A presente proposta foi guiada pelo seguinte conjunto de três Questões de Pesquisa (QPs), formuladas para investigar, por meio de um *benchmark*, as limitações, capacidades e condições de complementaridade entre abordagens tradicionais e baseadas em LLMs na detecção de *CS* em TypeScript:

QP1. *Em que medida ferramentas tradicionais de análise estática são capazes de detectar CS em projetos em TypeScript?*

Essa questão estabelece a linha de base necessária para compreender o desempenho e as limitações dos detectores clássicos, especialmente no que se refere à sensibilidade a padrões arquiteturais, às dependências implícitas e às relações interclasse/intermódulo.

QP2. *Qual é a eficácia de LLMs na detecção e no apoio à refatoração de CS em TypeScript, considerando diferentes tipos de smells e prompts de entrada?*

Essa questão examina o desempenho dos LLMs em relação a dimensões críticas já mapeadas pela literatura, como sensibilidade ao *prompt*, cobertura por categoria de *smell* e capacidade de interpretar intenções e relações semânticas que desafiam heurísticas tradicionais.

QP3. *Sob quais condições LLMs podem superar ou complementar ferramentas tradicionais na detecção de CS em TypeScript?*

Essa questão investiga a interação entre abordagens clássicas e baseadas em LLMs, identificando cenários em que os modelos podem apresentar vantagem, seja por ampliar a cobertura, reduzir falsos negativos ou oferecer suporte mais refinado à refatoração.

1.3 Metodologia do Estudo

A metodologia adotada nesta proposta foi estruturada em três fases analíticas complementares, concebidas para conferir rigor conceitual, sistematicidade empírica e reprodutibilidade ao processo investigativo. Essas fases foram definidas para sustentar a construção de um *benchmark* para detecção de *CS* em TypeScript, bem como a investigação de abordagens tradicionais e baseadas em LLMs no contexto dos artefatos experimentais desenvolvidos nesta pesquisa.

A Fase 1 concentrou-se na consolidação do escopo conceitual da pesquisa. Nessa fase, foi conduzido um exame aprofundado das ferramentas tradicionais de detecção de *CS* e uma revisão crítica da literatura sobre o uso de LLMs em tarefas de análise de código, com o objetivo de identificar limitações metodológicas, lacunas empíricas e desafios específicos do

ecossistema do TypeScript. Como resultado, procedeu-se à delimitação e à operacionalização de um subconjunto de *CS* clássicos, selecionados com base na taxonomia de Fowler [32], bem como à definição de critérios operacionais explícitos para sua identificação em código em TypeScript. Esse processo fundamentou a especificação conceitual do *benchmark* a ser construído ao longo da pesquisa.

A Fase 2 correspondeu à operacionalização empírica desse escopo, envolvendo a construção do conjunto de dados rotulado, a definição dos protocolos experimentais e a elaboração de *scripts* e *prompts* padronizados. Nessa etapa, foi conduzido um estudo exploratório inicial com projetos reais em TypeScript, por meio da aplicação controlada de ferramentas tradicionais e de LLMs, com o objetivo de validar o protocolo experimental, analisar padrões iniciais de detecção e identificar limitações práticas. Os resultados obtidos nessa fase constituem a base empírica preliminar apresentada no Capítulo 5.

Por fim, a Fase 3, a ser desenvolvida ao longo da etapa final do processo de doutoramento, prevê a expansão dos experimentos com novos modelos e ferramentas, o refinamento do *benchmark* e a realização de um *survey* com desenvolvedores para validação empírica dos achados. Essa fase tem como objetivo consolidar a abordagem proposta, derivar diretrizes práticas para a integração de LLMs a processos de análise estática e sintetizar os resultados da pesquisa, articulando implicações, limitações e oportunidades para trabalhos futuros.

A descrição apresentada nesta seção fornece uma visão geral da abordagem metodológica adotada. A exposição completa das fases, etapas, instrumentos e procedimentos experimentais encontra-se detalhada no Capítulo 4.

1.4 Relevância e Contribuições

As contribuições desta pesquisa organizam-se em quatro eixos complementares que refletem tanto o caráter empírico do estudo quanto sua inserção nas discussões contemporâneas sobre qualidade de software. Em primeiro lugar, o trabalho apresenta evidências sistemáticas, obtidas a partir de um *benchmark* reprodutível, sobre o *gap* de detecção observado em ferramentas tradicionais quando aplicadas a *CS* de natureza estrutural e dependentes de contexto semântico.

Em segundo lugar, apresenta-se uma avaliação rigorosa orientada à investigação das ca-

pacidades e limitações dos LLMs na detecção e no apoio à refatoração de *CS*, sustentada por critérios que delimitam condições de superioridade, equivalência ou complementaridade em relação às abordagens convencionais de análise estática. Em terceiro lugar, propõe-se um protocolo reprodutível, materializado em um *benchmark* público, composto por um conjunto de dados rotulado, *prompts* documentados, *scripts* de execução e diretrizes de padronização dos experimentos, de modo a permitir comparações futuras e promover a consolidação de uma base metodológica para investigações subsequentes.

Por fim, derivam-se diretrizes práticas para a integração de LLMs a ferramentas de mercado e a *pipelines* de qualidade, oferecendo recomendações sobre os tipos de *CS* que podem se beneficiar de abordagens híbridas e sobre estratégias de refatoração consistentes com a taxonomia de Fowler [32].

A relevância deste estudo manifesta-se em duas dimensões. Do ponto de vista científico, a pesquisa contribui para reduzir lacunas persistentes relacionadas à baixa cobertura de linguagens modernas, à ausência de *benchmarks* comparáveis e à escassez de análises empíricas robustas sobre o uso de LLMs na detecção de *CS*, sobretudo no domínio da linguagem TypeScript.

Adicionalmente, os resultados preliminares já obtidos evidenciam diferenças expressivas de cobertura e de comportamento entre abordagens tradicionais e baseadas em LLMs, reforçando a pertinência da investigação e a necessidade de análises comparativas mais aprofundadas nas fases subsequentes do estudo.

Já no plano prático, os resultados fornecem subsídios para apoiar equipes de desenvolvimento na tomada de decisão sobre quando e como empregar LLMs, indicando cenários em que tais modelos podem aprimorar a detecção de problemas de *design*, reduzir a dívida técnica e orientar refatorações com melhor relação custo-benefício.

1.5 Uso de LLMs no apoio à Escrita Acadêmica

Durante a preparação deste trabalho, foram utilizadas ferramentas baseadas em LLMs para apoio às atividades de escrita acadêmica, em conformidade com as diretrizes estabelecidas no documento *Guidelines for the use of AI in academic activities* apresentado no Apêndice B. O uso dessas ferramentas teve caráter estritamente auxiliar, voltado ao aprimoramento lin-

guístico, à organização argumentativa e à clareza expositiva do texto, sem substituir a autoria intelectual, a análise crítica ou a tomada de decisões científicas, que permaneceram integralmente sob responsabilidade do autor.

Especificamente, os LLMs foram empregados apenas em atividades de apoio, incluindo: (i) refinamento estilístico de trechos previamente redigidos; (ii) revisão gramatical e ortográfica; (iii) reorganização de estruturas sintáticas com vistas à maior fluidez textual; e (iv) verificação de consistência terminológica ao longo dos capítulos.

Ressalta-se que nenhuma tradução direta do português para o inglês foi realizada por meio de ferramentas de IA, tampouco houve geração automática de conteúdo conceitual ou científico. Em todos os casos, as sugestões fornecidas pelos modelos foram cuidadosamente avaliadas, revisadas e, quando necessário, reescritas, de modo a assegurar a precisão conceitual, a coerência metodológica e a aderência ao escopo científico da pesquisa.

Foram utilizadas, de forma complementar, diferentes plataformas e modelos, sempre em suas versões profissionais, pagas ou de código aberto, evitando-se o uso de versões gratuitas com capacidades limitadas. O ChatGPT (versões 4 e 5) foi empregado principalmente para a reorganização textual, a uniformização do estilo e a verificação da coerência interna entre as seções. O modelo Claude, da Anthropic, foi utilizado em atividades que demandavam maior sensibilidade ao contexto discursivo e maior apoio à clareza argumentativa. O Gemini, da Google DeepMind, contribuiu para revisões mais amplas de trechos extensos. Ferramentas especializadas, como o Grammarly, foram empregadas para a inspeção de aspectos gramaticais finos da língua portuguesa. Já o ChatPDF foi utilizado como apoio à leitura, extração e organização de informações provenientes de artigos científicos, auxiliando no processamento de grandes volumes textuais durante a revisão da literatura.

Ressalta-se que os LLMs não foram utilizados para a geração de sugestões de pesquisa, a definição de hipóteses, a elaboração de análises ou a produção de referências bibliográficas. Todas as decisões metodológicas, interpretações teóricas e formulações conceituais foram conduzidas exclusivamente pelo autor.

Dessa forma, o emprego de ferramentas de IA ocorreu de maneira ética, transparente e controlada, contribuindo para a melhoria da clareza, da coesão e da padronização formal do texto, sem comprometer a autoria intelectual ou a originalidade científica do trabalho. Essa abordagem reforça o compromisso desta pesquisa com práticas responsáveis e alinhadas às

recomendações contemporâneas sobre o uso de IA em atividades acadêmicas.

1.6 Estrutura do Documento

Os capítulos subsequentes deste documento estão organizados da seguinte forma:

- **Capítulo 2: Fundamentação Teórica**. Apresenta os conceitos centrais que sustentam esta proposta, abrangendo a caracterização dos *CS*, os princípios de refatoração, os fundamentos dos LLMs e as plataformas que viabilizam sua execução, bem como suas aplicações, benefícios e desafios na Engenharia de Software.

- **Capítulo 3: Trabalhos Relacionados**. Examina estudos recentes sobre a detecção e refatoração de *CS* com LLMs e arquiteturas híbridas, discutindo contribuições, limitações e convergências com os objetivos desta proposta, especialmente no contexto de linguagens modernas, como TypeScript.

- **Capítulo 4: Metodologia**. Detalha o desenho metodológico da pesquisa, estruturado em três fases que compreendem a definição do escopo e dos critérios operacionais, a construção e aplicação de um *benchmark* para detecção de *CS* em TypeScript e a validação empírica dos resultados por meio de *survey*

- **Capítulo 5: Resultados Preliminares**. Apresenta os resultados do Estudo Exploratório 1, conduzido com 106 projetos em TypeScript, que analisa a cobertura, a distribuição e os padrões iniciais de detecção de *CS* por ferramentas tradicionais e LLMs, bem como suas limitações e complementaridades.

- **Apêndice A: Estudo de Mapeamento Sistemático**. Descreve o mapeamento sistemático realizado para levantar evidências sobre o uso de LLMs na detecção de *CS*, identificando tendências, lacunas e oportunidades que justificam esta investigação.

- **Apêndice B: Diretrizes para o Uso de IA em Atividades Acadêmicas**. Apresenta as diretrizes adotadas para o uso ético e responsável de ferramentas de IA ao longo da pesquisa.

- **Apêndice C: Artefatos de Pesquisa e Reprodutibilidade**. Disponibiliza, em ambiente web, o acesso estruturado aos principais artefatos produzidos ao longo do estudo, organizados em uma pasta específica.

Capítulo 2

Fundamentação Teórica

Este capítulo apresenta os fundamentos conceituais que sustentam as análises, as escolhas metodológicas e as discussões desenvolvidas ao longo desta proposta. O objetivo é estabelecer o arcabouço teórico necessário para compreender os fenômenos investigados, situando-os no contexto mais amplo da ES e das recentes transformações impulsionadas pelos LLMs.

Inicialmente, examina-se o conceito de *(CS)*, incluindo suas definições, características e impactos na manutenibilidade e na evolução de sistemas de software. São também discutidos os mecanismos tradicionais de detecção desses indícios estruturais, bem como o conjunto específico de *CS* considerado neste estudo, conforme apresentado na Seção 2.1. Em seguida, a Seção 2.2 aborda os princípios fundamentais de refatoração, destacando sua relevância para a melhoria contínua do design interno do software e exemplificando operações amplamente aceitas na literatura.

Na segunda parte do capítulo, aprofunda-se a discussão sobre LLMs, abordando sua definição, evolução tecnológica e princípios de funcionamento. A Seção 2.3 apresenta os modelos proprietários, as arquiteturas *open source* e as plataformas de execução local, descrevendo suas características, potenciais aplicações e limitações no contexto da ES. Especial atenção é dada às capacidades dos LLMs para compreender, analisar e gerar código, aspecto objeto desta investigação.

Por fim, a Seção 2.4 sintetiza os principais conceitos apresentados e estabelece as bases conceituais que orientam o desenvolvimento da pesquisa nos capítulos posteriores.

2.1 Code Smells (CS)

Os *CS* representam indícios de problemas estruturais que, embora não comprometam o funcionamento imediato da aplicação, aumentam a dívida técnica e dificultam a manutenção [32]. Funcionam como sinais de que o código pode se beneficiar de reorganização para melhorar a legibilidade, reduzir a complexidade e aperfeiçoar o *design* interno [7, 61].

Na obra clássica de Fowler *et al.* [33], a refatoração é apresentada sob uma perspectiva pragmática, voltada sobretudo para programadores. Os autores introduzem princípios fundamentais, destacam a importância da tomada de decisão baseada na experiência e descrevem um conjunto de 22 tipos de *bad smells*, concebidos como padrões ou indícios de potenciais problemas de *design*.

Entre os exemplos mais recorrentes estão: (i) *God Class*; (ii) *Shotgun Surgery*; (iii) *Lazy Class*; (iv) *Middle Man*; e (v) *Divergent Change*. Além de categorizar tais problemas, os autores enfatizam quando e por que a refatoração deve ser empregada, reforçando que o reconhecimento de *CS* é, em grande medida, um processo apoiado tanto na análise técnica quanto na experiência do desenvolvedor.

A literatura evidencia que diferentes denominações têm sido empregadas para se referir a esses problemas estruturais. Segundo a revisão sistemática de Sobrinho *et al.* [20], termos como *code anomaly* [52], *bad smell* [53], *anti-pattern* [76] e *design smell* [50] frequentemente aparecem na literatura especializada. No entanto, nesta proposta, adota-se prioritariamente o termo *Code Smell (CS)*, alinhando-se à terminologia empregada nas obras originais de Fowler [33, 32] e às definições amplamente consolidadas na comunidade de ES [39].

Sharma e Spinellis [66] identificam diversos fatores que contribuem para a adoção de *CS* em sistemas de software. Tais fatores abrangem dimensões humanas, organizacionais, tecnológicas e de requisitos, incluindo: (i) falta de habilidade ou consciência; (ii) mudanças frequentes de requisitos; (iii) restrições impostas por linguagem ou plataforma; (iv) lacunas de conhecimento; (v) processos inadequados; (vi) pressão por prazos; (vii) priorização de funcionalidades em detrimento da qualidade; (viii) políticas organizacionais; (ix) cultura da equipe; e (x) má gestão de recursos. Dessa forma, *CS* geralmente emergem da combinação de práticas de desenvolvimento, de limitações tecnológicas e de fatores humanos.

Diversos estudos [7, 20, 61, 66] mostram que *CS* exercem impacto direto e significa-

tivo na qualidade interna do software. Eles dificultam a compreensão do sistema, elevam o esforço necessário para as modificações, reduzem a confiabilidade, prejudicam o desempenho e afetam negativamente a produtividade da equipe. Estruturas confusas, redundantes ou excessivamente complexas ampliam o risco de defeitos, comprometem a evolução do sistema e podem afetar a motivação dos desenvolvedores. Em última instância, esses fatores repercutem na qualidade do produto final e no custo de sua manutenção.

Ferramentas de detecção automática de *CS* têm sido amplamente empregadas para apoiar desenvolvedores na identificação precoce de problemas estruturais [7, 13, 20, 51, 65, 69, 77]. Soluções como SonarQube [73], ESLint [28], typescript-eslint [78] e Embold [25] utilizam análise estática baseada em regras e heurísticas para detectar padrões potencialmente problemáticos.

Contudo, mesmo com sua ampla adoção, tais ferramentas apresentam limitações significativas, como o uso de heurísticas rígidas, baixa sensibilidade a aspectos semânticos e dificuldade em capturar nuances mais profundas do comportamento do código, o que mantém a detecção de *CS* como um desafio técnico relevante. Tais limitações ajudam a explicar o crescente interesse em investigar soluções alternativas, como, por exemplo, LLMs, tema aprofundado na Subseção 2.3.5.

Diante da relevância dos *CS* para a qualidade do software, o passo seguinte consiste em examinar como esses problemas são tradicionalmente detectados e quais tipos específicos são considerados no escopo desta pesquisa. A Subseção 2.1.1 detalha os seis *CS* selecionados para análise ao longo desta proposta, enquanto a Subseção 2.1.2 discute as abordagens clássicas de detecção automática.

2.1.1 Tipos de CS

Inicialmente, o estudo de Fowler *et al.* (1999) [33] identificou 22 tipos de *CS*. Posteriormente, Fowler [32] revisou e ampliou essa classificação, totalizando 24 tipos de *CS*. Nesta atualização, alguns tipos foram preservados ou reformulados, como *Alternative Classes with Different Interfaces*, *Middle Man* e *Refused Bequest*, e outros foram introduzidos, tais como *Mysterious Name*, *Repeated Switches* e *Loops*.

A seguir, apresenta-se, na Tabela 2.1, o escopo de *CS* selecionado para análise neste estudo, com base nas categorizações de Fowler [33, 32].

Tabela 2.1: Escopo de CS do estudo

ID	**Nome**	**Descrição**
S29	*Parallel Inheritance Hierarchies*	Ocorre quando a criação de uma subclasse em uma hierarquia exige a criação de uma subclasse correspondente em outra hierarquia.
S30	*Lazy Class*	É uma classe que não possui funcionalidade suficiente para justificar a sua existência.
S34	*Middle Man*	É uma classe que faz pouco mais do que delegar chamadas a outro objeto. Serve apenas como intermediário, repassando solicitações a outros objetos.
S36	*Alternative Classes with Different Interfaces*	Ocorre quando duas ou mais classes realizam funções semelhantes, mas possuem interfaces diferentes.
S37	*Incomplete Library Class*	Ocorre quando uma classe de biblioteca ou utilitária não fornece todos os métodos ou funcionalidades necessários, forçando desenvolvimento de métodos para suprir lacunas.
S39	*Refused Bequest*	Ocorre quando uma subclasse herda métodos ou propriedades de uma classe pai, mas não os utiliza ou os rejeita explicitamente.

Conforme indicado na Tabela 2.1, foram selecionados 6 (seis) tipos de *CS* que compõem o escopo deste estudo, conforme definidos por Fowler *et al.* [33, 32]. Para cada tipo, são apresentados trechos de código-fonte ilustrativos, acompanhados de uma descrição conceitual e de uma análise sobre suas principais implicações para o design e a manutenibilidade do software.

O *CS Parallel Inheritance Hierarchies* caracteriza-se pela existência de duas ou mais hierarquias de herança que evoluem de forma paralela e interdependente. Nesse tipo de estrutura, a cada criação de uma nova subclasse em uma hierarquia, é necessário criar uma classe correspondente em outra para preservar o comportamento esperado do sistema [33,

32].

Esse acoplamento excessivo entre hierarquias reduz a modularidade, dificulta a manutenção e gera dependências estruturais entre domínios que deveriam ser independentes (por exemplo, forma e cor), tornando o código rígido para evoluir e frágil diante de mudanças – o que viola o princípio da separação de responsabilidades e compromete a evolução autônoma dos componentes.

Um exemplo deste *CS* é mostrado na Figura 2.1.

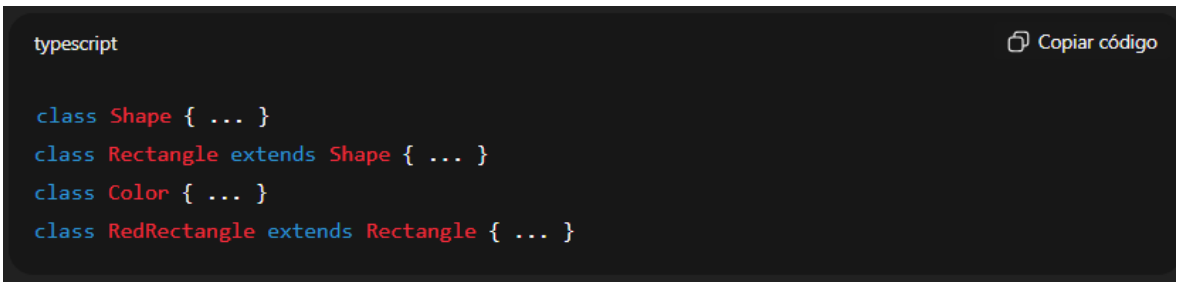

Figura 2.1: Exemplo de Parallel Inheritance Hierarchies.

A Figura 2.1 ilustra duas dimensões conceituais que passam a evoluir de forma acoplada: a hierarquia de formas (`Shape` e sua subclasse `Rectangle`) e a noção de cor (`Color`). Em vez de tratar cor como uma característica independente que pode ser combinada com qualquer forma, o sistema introduz classes específicas como `RedRectangle`, que herda de `Rectangle` e representa simultaneamente "ser um retângulo" e "ser vermelho".

Esse padrão leva a uma explosão combinatória de subclasses: para cada nova forma adicionada à hierarquia (por exemplo, `Circle`, `Triangle`), é comum que o desenvolvedor precise criar variantes específicas para cada cor (`RedCircle`, `BlueTriangle`, etc.). Esse espelhamento forçado entre eixos conceituais distintos caracteriza o *Parallel Inheritance Hierarchies*, pois a introdução de novas especializações em um eixo (forma) exige especializações correspondentes no outro (cor), em vez de permitir a composição flexível entre eles.

Já o *CS Lazy Class* ocorre quando classes são criadas com a intenção de serem expandidas no futuro, mas acabam contendo pouca ou nenhuma lógica significativa. Nesse tipo de situação, a classe não realiza trabalho suficiente para justificar sua existência, tornando-se redundante na estrutura do sistema. Manter essas classes introduz complexidade desnecessária, pois aumenta o número de elementos a serem compreendidos e mantidos sem oferecer

benefícios reais de modularização [33, 32].

Além disso, esse tipo de classe pode confundir desenvolvedores, dificultar a navegação do código e evidenciar uma distribuição inadequada de responsabilidades no design. Um exemplo deste *CS* é apresentado na Figura 2.2.

Figura 2.2: Exemplo de Lazy Class

A Figura 2.2 apresenta uma classe denominada `Logger`, cuja única função é encapsular uma chamada simples a `console.log`. Esse comportamento não adiciona valor significativo ao sistema, visto que o método apenas delega a execução a uma função já existente da linguagem. Caso essa classe seja utilizada apenas uma vez, sua presença não se justifica, podendo ser substituída por uma função utilitária direta.

Esse exemplo evidencia o *Lazy Class*, pois ilustra uma abstração supérflua que aumenta a complexidade estrutural sem contribuir efetivamente para a clareza, a reutilização ou a manutenibilidade do código.

Por sua vez, o *CS Middle Man* surge quando uma classe é criada apenas para intermediar chamadas entre outras classes, delegando a maior parte de suas responsabilidades sem acrescentar valor real ao processamento. Esse tipo de estrutura surge, em geral, a partir de um excesso de abstração ou de uma tentativa de isolamento que não se justifica. Como resultado, o código torna-se mais complexo do que o necessário, com camadas intermediárias que aumentam o acoplamento e reduzem a clareza [33, 32].

A presença de *Middle Man* dificulta a manutenção e a depuração, uma vez que o fluxo de execução passa por classes que não desempenham um papel funcional relevante. Um exemplo deste *CS* é mostrado na Figura 2.3.

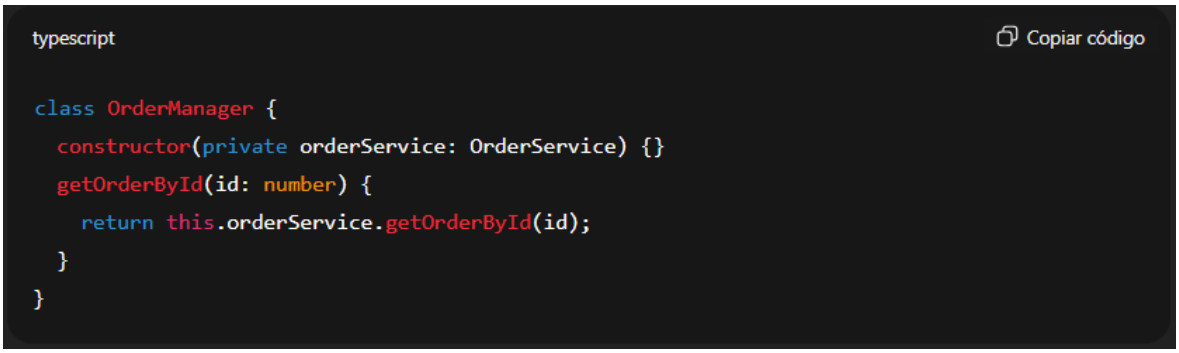

Figura 2.3: Exemplo de Middle Man

A Figura 2.3 apresenta a classe *OrderManager*, cuja única responsabilidade é encaminhar as chamadas ao objeto `OrderService`. Nesse caso, o método `getOrderById()` não contém lógica adicional, apenas repassa a solicitação para outro componente. Essa estrutura cria uma camada intermediária desnecessária, que não agrega comportamento nem encapsula regras de negócio. A classe `OrderManager`, portanto, exemplifica o *Middle Man*, pois atua apenas como um mediador redundante, tornando o sistema mais difícil de compreender e aumentando o custo de manutenção sem trazer benefícios reais de abstração.

Na sequência, o *CS Alternative Classes with Different Interfaces* acontece quando duas ou mais classes oferecem funcionalidades semelhantes, mas expõem interfaces distintas. Esse problema geralmente resulta de decisões de projeto não padronizadas ou da evolução independente de partes do sistema, o que leva à criação de múltiplas implementações que realizam tarefas equivalentes de modos distintos [33, 32].

A ausência de uma interface comum impede o uso de polimorfismo e força o desenvolvedor a lidar com métodos com nomes ou assinaturas divergentes para executar operações semelhantes. Como consequência, o código torna-se redundante e mais difícil de manter, exigindo o uso de adaptadores ou a duplicação de lógica para garantir a integração entre os componentes. Um exemplo deste *CS* é apresentado na Figura 2.4.

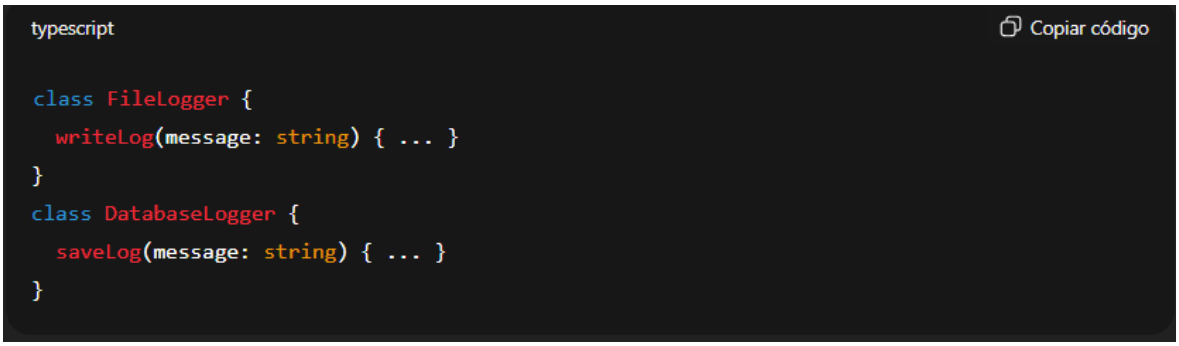

Figura 2.4: Exemplo de Alternative Classes with Different Interfaces

A Figura 2.4 apresenta duas classes, `FileLogger` e `DatabaseLogger`, que executam a mesma função – registrar mensagens de *log* – porém, utilizam métodos diferentes: `writeLog()` e `saveLog()`. Não obstante ambas implementem o mesmo conceito, a ausência de uma interface comum impede que sejam utilizadas de forma polimórfica.

Essa discrepância obriga o desenvolvedor a conhecer e tratar cada classe de forma específica, comprometendo a consistência do código e dificultando a substituição ou extensão das implementações. Esse cenário exemplifica o *Alternative Classes with Different Interfaces*, pois demonstra a falta de padronização entre classes equivalentes e evidencia a necessidade de abstração por meio de uma interface ou de uma classe base compartilhada.

Destaca-se, ainda, o *CS Incomplete Library Class*, que é identificado quando uma classe pertencente a uma biblioteca não fornece todas as funcionalidades necessárias para atender às demandas do sistema que a utiliza. Isso normalmente é verificado quando o *design* inicial da biblioteca possui escopo restrito ou quando o sistema evolui, exigindo comportamentos não previstos pelos desenvolvedores originais da *API* [33, 32].

Nesses casos, os programadores acabam sendo induzidos a criarem extensões, subclasses ou adaptadores para suprir as lacunas existentes, o que resulta em duplicação de código e em soluções locais que não seguem um padrão comum. Tal prática eleva o esforço de manutenção e compromete a confiabilidade, já que cada adaptação pode introduzir inconsistências ou comportamentos inesperados em relação à biblioteca original. Um exemplo deste *CS* é apresentado na Figura 2.5.

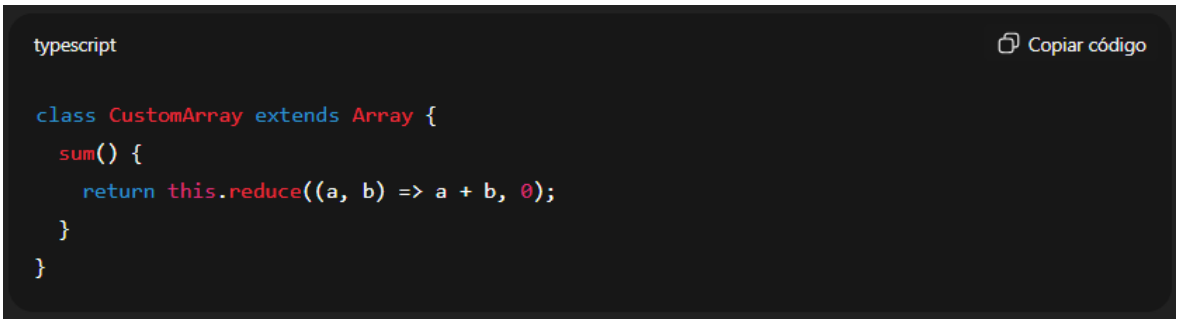

Figura 2.5: Exemplo de Incomplete Library Class

A Figura 2.5 ilustra uma classe denominada `CustomArray`, que estende a classe nativa `Array` e inclui o método `sum()`. Essa adição é necessária porque a biblioteca padrão não oferece essa funcionalidade. Embora pareça uma solução simples, esse tipo de extensão evidencia claramente o *Incomplete Library Class*, pois mostra que a classe base não atende integralmente às demandas da aplicação. Como consequência, os desenvolvedores passam a manter versões personalizadas ou duplicadas da mesma lógica em diferentes partes do sistema, comprometendo a padronização, a reutilização e a manutenibilidade do código.

Por fim, o *CS Refused Bequest* ocorre quando uma subclasse herda métodos ou atributos da superclasse, entretanto, não os utiliza ou até mesmo os sobrescreve para anular o comportamento herdado. Esse padrão indica uma falha na hierarquia de herança, pois a subclasse não se ajusta conceitualmente à abstração definida pela classe base [33, 32].

Como resultado disso, o design se torna inconsistente, e o princípio da substituição de Liskov [44] é violado, uma vez que a subclasse não pode substituir com segurança sua classe base sem alterar o comportamento esperado. Esse tipo de problema aumenta a probabilidade de erros em tempo de execução e reduz a clareza do código, dificultando sua manutenção e evolução. Um exemplo deste *CS* é mostrado na Figura 2.6.

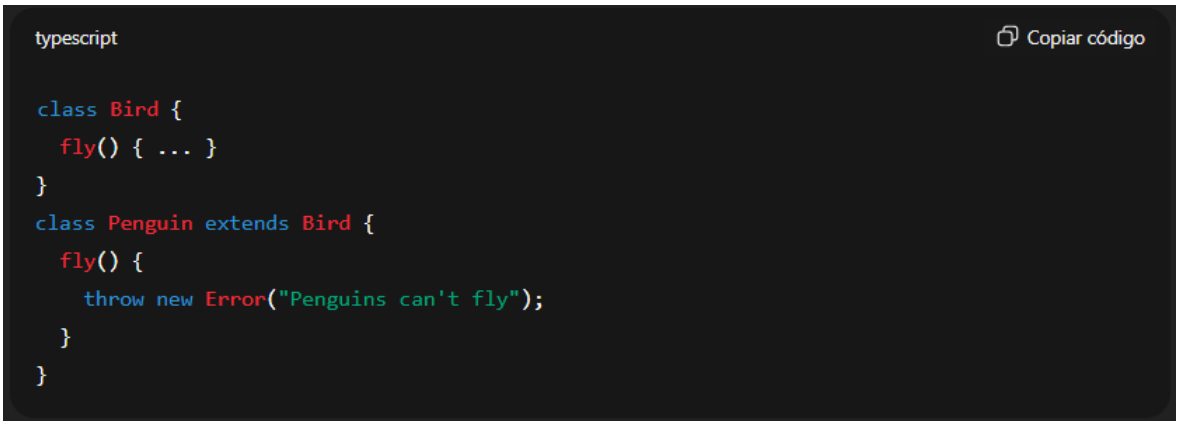

Figura 2.6: Exemplo de Refused Bequest

A Figura 2.6 apresenta uma hierarquia composta pelas classes `Bird` (representando pássaros) e `Penguin` (representando pinguins). A superclasse `Bird` define o método `fly()`, que representa a capacidade de voo característica da maioria das aves. No entanto, a subclasse `Penguin` sobrescreve esse método para lançar uma exceção, pois pinguins não podem voar. Essa situação evidencia claramente o *Refused Bequest*, pois a subclasse herda comportamentos que não se aplicam à sua natureza.

O uso inadequado da herança nesse contexto viola o princípio de substituição de Liskov [44] e demonstra que a relação hierárquica foi mal definida – nesse caso, seria mais apropriado introduzir uma abstração diferente, como uma interface ou uma classe base mais genérica, que representasse comportamentos comuns a todas as aves, sem impor funcionalidades que algumas subclasses não podem desempenhar.

2.1.2 Detecção de *CS*

A identificação e a correção de *CS* constituem etapas fundamentais para prevenir defeitos e reduzir a dívida técnica em sistemas de software. Para apoiar tais atividades, ferramentas de análise estática têm sido amplamente empregadas, permitindo detectar padrões estruturais potencialmente problemáticos e fornecer indicadores objetivos sobre a qualidade interna do código. Essas soluções viabilizam análises sistemáticas em diferentes linguagens e domínios, oferecendo suporte às equipes de desenvolvimento na avaliação contínua de atributos como legibilidade, manutenibilidade e confiabilidade.

O SonarQube [73] é uma das ferramentas mais consolidadas para análise estática, capaz

de identificar problemas de codificação, incluindo *CS* [7], em mais de 30 linguagens (*e.g.*, JavaScript, TypeScript [13]). Seu funcionamento baseia-se em um conjunto extenso de regras que avaliam aspectos de manutenibilidade, confiabilidade e segurança.

Além disso, disponibiliza um painel detalhado de monitoramento contínuo da qualidade, auxiliando na priorização de problemas e no gerenciamento da dívida técnica, como ilustrado no painel de análise estática do SonarQube Figura 2.7.

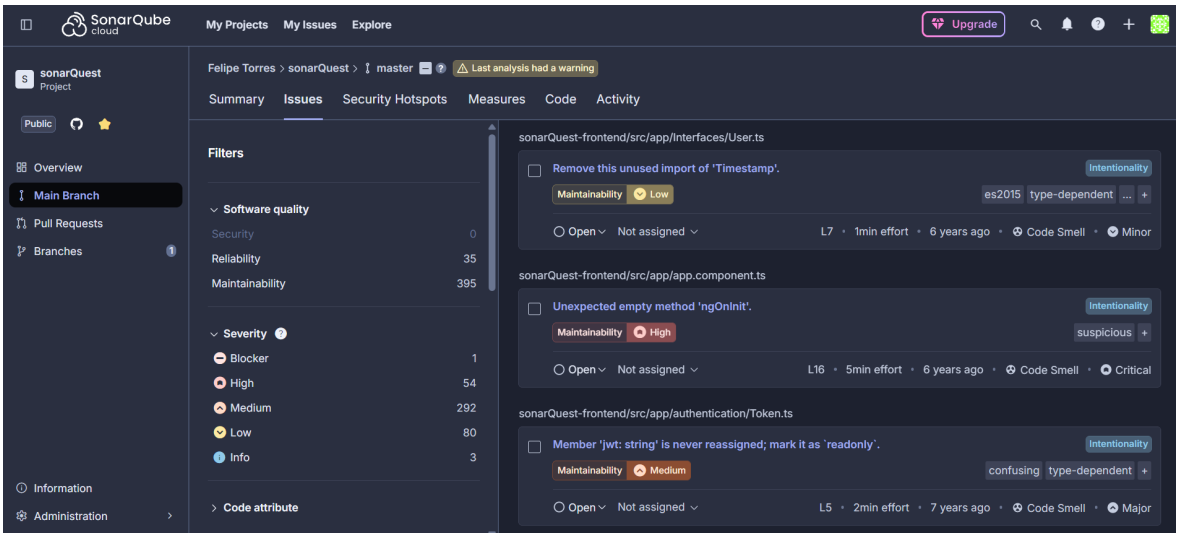

Figura 2.7: Exemplo de painel do SonarQube. Fonte: SonarQube Cloud (2025) [72].

Outra ferramenta amplamente utilizada é o ESLint [28], voltado à identificação e ao relato de padrões problemáticos em código JavaScript (ECMAScript). Extensível e personalizável, o ESLint permite configurar regras e analisadores por meio de *plugins*, adaptando o processo de verificação aos padrões de cada projeto.

Além disso, oferece mecanismos de correção automática baseados em representações sintáticas do código, evitando erros comuns de soluções baseadas exclusivamente na busca textual. Por sua popularidade, é empregado em larga escala por empresas como Microsoft, Airbnb, Netflix e Facebook [28], e amplamente difundido na comunidade JavaScript [77].

A Figura 2.8 ilustra um ambiente de experimentação da ferramenta, enquanto a Figura 2.9 mostra seu uso em um fluxo de desenvolvimento.

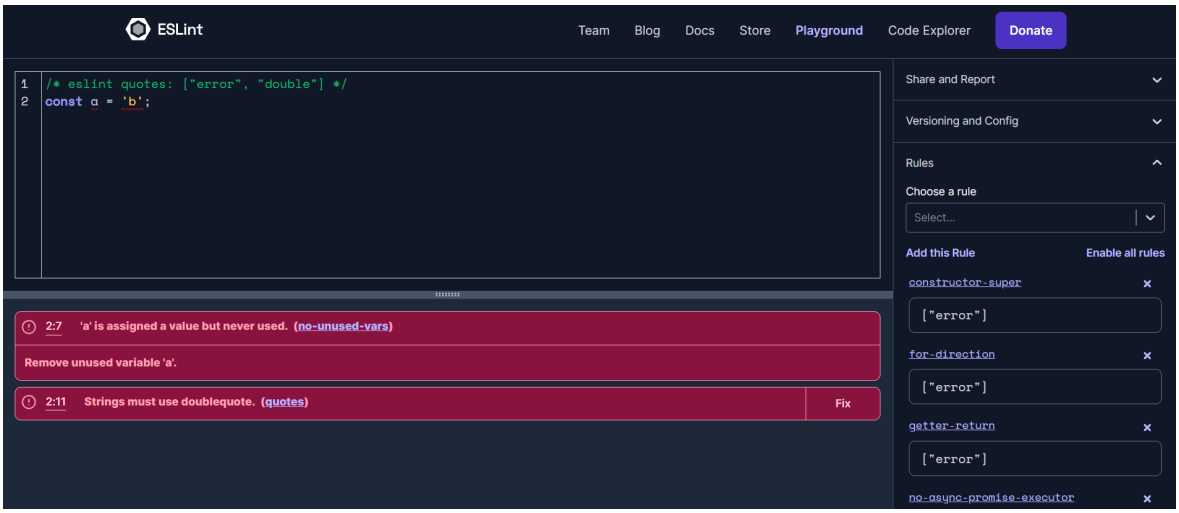

Figura 2.8: Exemplo de ambiente de experimentação do ESLint. Fonte: ESLint [27].

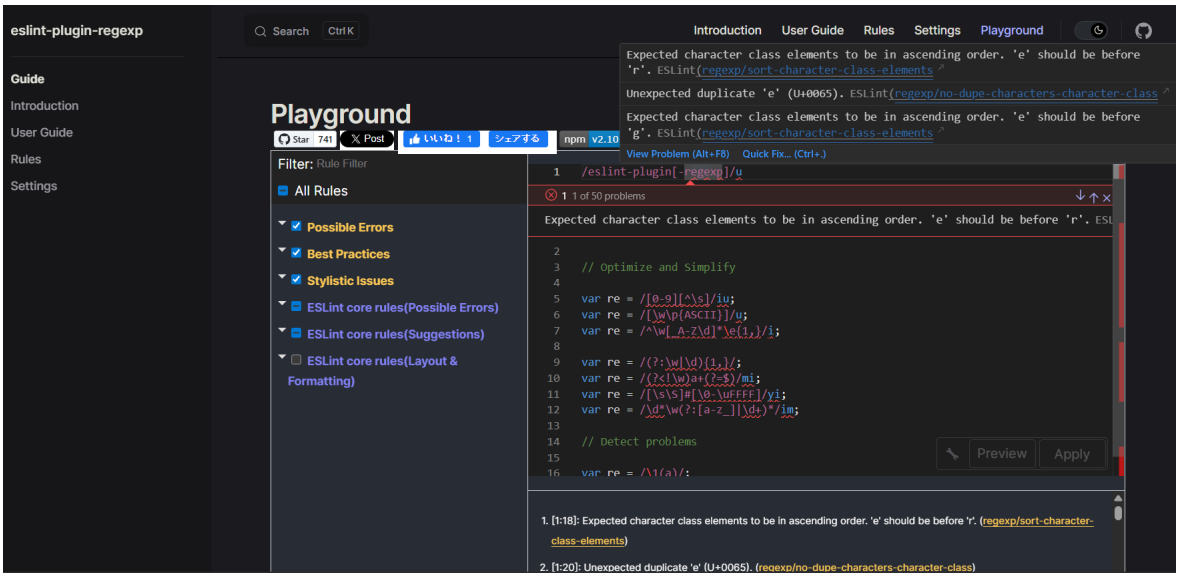

Figura 2.9: Execução do ESLint em contexto de desenvolvimento. Fonte: Ota (2025) [60].

Apesar de amplamente adotado, o ESLint não possui suporte nativo ao sistema de tipos do TypeScript, o que limita sua capacidade de analisar regras que dependem de informações de tipagem. Para superar essa restrição, surgiu o *typescript-eslint* [78], que integra o mecanismo do ESLint ao compilador do TypeScript, permitindo acessar a árvore sintática e os tipos inferidos. Isso permite aplicar regras específicas à linguagem, ampliando a precisão da análise em projetos em TypeScript.

A Figura 2.10 apresenta um exemplo de uso dessa integração para análise estática de

código.

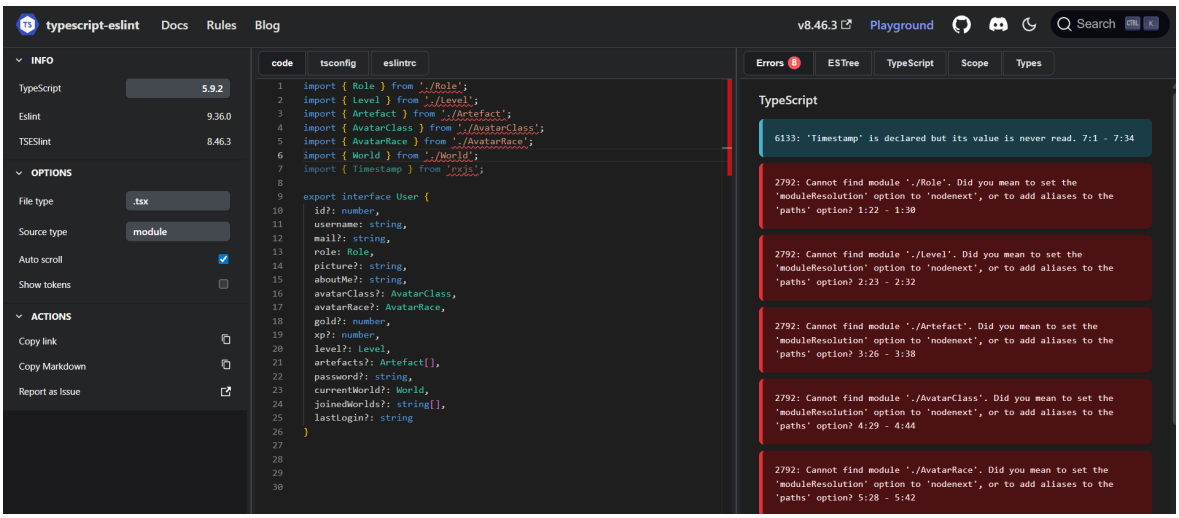

Figura 2.10: Exemplo de uso do *typescript-eslint*. Fonte: typescript-eslint (2025) [78]

Antes de sua descontinuação em 2019, o TSLint era a ferramenta principal de análise estática em TypeScript. Contudo, em razão de esforços da própria equipe do TypeScript e da comunidade, optou-se por consolidar o suporte em torno do ESLint e do *plugin* `@typescript-eslint`. O repositório do TSLint foi então arquivado e deixou de receber atualizações, reforçando a recomendação de migrar para o ecossistema ESLint para evitar incompatibilidades e garantir suporte contínuo às versões recentes da linguagem.

Além das ferramentas baseadas em regras, existem soluções voltadas à análise estrutural e detecção de antipadrões de projeto. O Embold [24, 25] é um exemplo de plataforma que combina análise estática, métricas especializadas e detecção de até 30 tipos de antipadrões, incluindo *God Class*, *Shotgun Surgery* e *Feature Envy*.

Sua interface fornece indicadores específicos com limiares predefinidos e suporte a linguagens como JavaScript e TypeScript [69, 51, 26]. A Figura 2.11 ilustra um painel de análise de código gerado pela ferramenta Embold no ambiente VS Code.

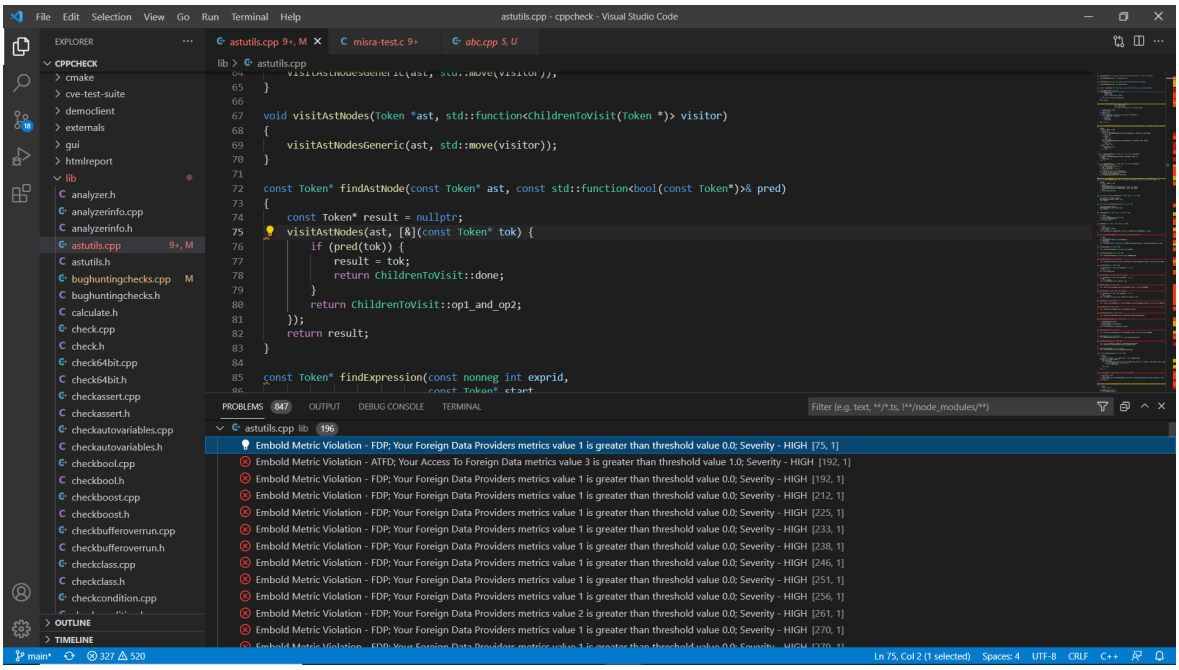

Figura 2.11: Exemplo de análise de código via Embold. Fonte: Embold (2025) [26]

De modo geral, ferramentas como SonarQube, ESLint, *typescript-eslint* e Embold se consolidaram como parte essencial de *pipelines* modernos de desenvolvimento, oferecendo suporte à identificação precoce de *CS* e de outros problemas estruturais [7, 13, 20, 51, 65, 69, 77]. Todavia, essas abordagens permanecem limitadas por sua natureza predominantemente baseada em regras e por sua capacidade reduzida de interpretar aspectos semânticos mais complexos do código. Dificuldades em captar nuances contextuais e as dependências entre elementos estruturais reforçam a necessidade de investigar abordagens complementares.

2.2 Refatoração

A refatoração constitui um processo central no aprimoramento contínuo do design interno de sistemas de software. Segundo Fowler [32], refatorar significa modificar a estrutura interna de um programa para torná-lo mais compreensível e menos custoso de modificar, sem alterar o seu comportamento observável. Essa prática busca preservar a funcionalidade existente enquanto promove melhorias estruturais que elevem a legibilidade, a coesão e a manutenibilidade do código.

Apesar de o termo seja, por vezes, empregado de forma ampla para designar qualquer

tipo de reorganização de código, Fowler [32] propõe uma definição mais precisa: *refatorar* implica aplicar pequenas transformações sucessivas e seguras, capazes de produzir melhorias cumulativas na arquitetura do software. Essa abordagem incremental permite ao desenvolvedor reestruturar o código de forma controlada e contínua, sem introduzir falhas nem interromper o funcionamento do sistema.

Neste trabalho, são destacadas duas operações representativas de refatoração, cada uma associada a um grupo de operações distinto definido por Fowler [32]:

- *Move Function*, pertencente ao grupo que engloba deslocamento de recursos, cuja finalidade é transferir funções ou métodos de um contexto para outro em que possuem maior pertinência semântica e coesão, reduzindo acoplamentos indevidos; e
- *Extract Class*, vinculada ao grupo de encapsulamento, responsável por isolar subconjuntos de responsabilidades em novas classes, agrupando dados e comportamentos correlatos para aprimorar a clareza e a modularidade do *design*.

Essas operações exemplificam como a refatoração atua na melhoria progressiva da estrutura interna do software, promovendo um design mais claro, compreensível e preparado para evoluções futuras.

A refatoração *Move Function* – anteriormente denominada *Move Method* [33] – é uma técnica essencial para a manutenção da modularidade, princípio segundo o qual é desejável modificar partes do sistema sem necessidade de compreender seu funcionamento completo [32]. Em sistemas orientados a objetos, é aplicada quando um método passa a depender mais de atributos ou comportamentos de outra classe do que daqueles presentes em seu próprio contexto. Nesses casos, o método deve ser movido para a classe com a qual mantém maior afinidade lógica. O resultado é um design mais coeso, com menor acoplamento e maior clareza semântica.

A Figura 2.12 apresenta um exemplo de operação *Move Function* retirado de Fowler [32].

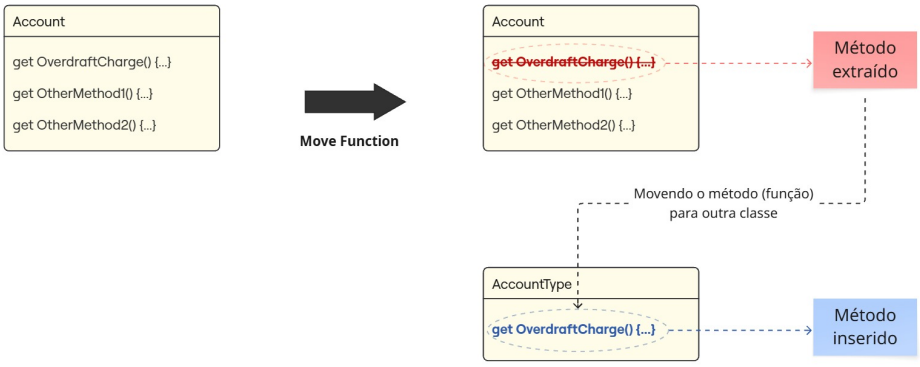

Figura 2.12: Exemplo de *Move Function*. Fonte: Adaptado de Fowler [32]

A Figura 2.12 ilustra o processo de transferência do método `getOverdraftCharge()` da classe `Account` para a classe `AccountType`, representando a migração de uma lógica fortemente dependente de dados mantidos por outra classe. Esse movimento evidencia a importância de posicionar o comportamento junto aos dados que o sustentam, promovendo um *design* mais coeso, intuitivo e de fácil manutenção.

A refatoração *Move Function* (ou *Move Method*) reflete a aplicação prática dos princípios de alta coesão e baixo acoplamento. Em síntese, essa refatoração traduz a ideia de que o bom design de software é construído por meio de ajustes contínuos, acompanhando a evolução da compreensão do domínio e o refinamento das abstrações no código-fonte.

São exemplos de *CS* que podem ser mitigados por meio da operação de *Move Function* (*Move Method*) [33, 32, 64]:

I *Divergent Change*;

II *Shotgun Surgery*;

III *Feature Envy*;

IV *Switch Statements*;

V *Message Chains*;

VI *Data Class*;

VII *Temporary Field*;

VIII *Alternative Classes with Different Interfaces*;

IX *Inappropriate Intimacy*, quando a redistribuição das responsabilidades reduz dependências excessivas entre classes;

X *Parallel Inheritance Hierarchies*, quando o reposicionamento de métodos contribui para reduzir a propagação de alterações entre hierarquias paralelas.

Nos dois últimos casos, *Inappropriate Intimacy* e *Parallel Inheritance Hierarchies* são definições usadas por Fowler *et al.* [33], porém, foram extintas ou incorporadas a outras definições de *smells* na atualização de taxonomia realizada por Fowler [32]. Para eles, a refatoração *Move Function (Move Method)* atua principalmente como um mecanismo de mitigação estrutural, e não como uma solução definitiva.

Embora o reposicionamento adequado de métodos possa reduzir o acoplamento entre classes e diminuir a necessidade de alterações redundantes em hierarquias paralelas, a eliminação completa desses dois *smells* geralmente exige refatorações adicionais mais específicas (como *Extract Class*, *Extract Superclass* ou reorganização das relações de herança).

Já a refatoração *Extract Class* tem como objetivo dividir uma classe que acumulou responsabilidades em excesso em duas ou mais classes menores e mais coesas [33, 32]. Segundo Fowler [32], à medida que o sistema evolui, as classes tendem a crescer naturalmente, incorporando novos campos e métodos que tornam seu entendimento e manutenção mais complexos.

Quando um subconjunto de atributos e comportamentos passa a constituir um agrupamento lógico próprio, ou quando certos dados mudam de forma conjunta, surge o indício de que é necessário extrair uma nova classe. Esse processo permite realocar dados e funções correlacionadas, reduzindo o acoplamento interno e promovendo um *design* mais claro, sustentável e alinhado ao princípio da responsabilidade única [33, 32].

Um exemplo de *Extract Class* é exibido na Figura 2.13 retirado de Fowler [32].

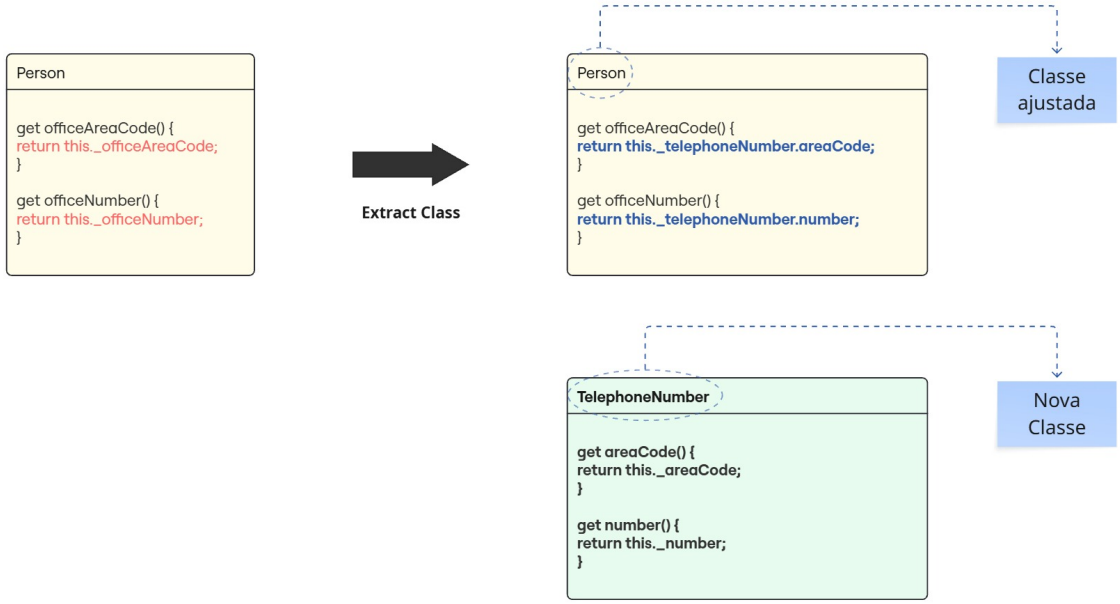

Figura 2.13: Exemplo de *Extract Class*. Fonte: Adaptado de Fowler [32]

A Figura 2.13 ilustra a refatoração *Extract Class*, na qual a classe `Person` delega parte de suas responsabilidades relacionadas a informações de telefone a uma nova classe, `TelephoneNumber`. Inicialmente, `Person` continha os campos `_officeAreaCode` e `_officeNumber`, além dos métodos de acesso correspondentes. Após a refatoração, esses elementos passam a ser encapsulados em uma nova classe denominada `TelephoneNumber`, enquanto `Person` mantém apenas a delegação.

Esse processo aumenta a coesão ao agrupar dados e operações relacionadas e reduz o acoplamento, tornando o código mais modular e compreensível. Segundo Fowler [32], o procedimento deve acontecer de forma incremental e testada: (i) primeiramente, define-se o recorte de responsabilidades; (ii) em seguida, cria-se a nova classe e a referência a ela; (iii) os campos e métodos são movidos gradualmente, validando o comportamento a cada etapa; e, por fim, (iv) revisam-se nomes e interfaces para refletir a nova estrutura.

O resultado é um *design* mais limpo e flexível, preparado para evoluções futuras sem inflar a classe original. Assim, a refatoração *Extract Class* expressa a busca contínua por simplicidade e organização, princípios centrais do bom *design* de software.

São exemplos de *CS* que podem ser mitigados por meio da operação de *Extract Class* [33, 32, 64]:

I *Large Class*;

II *Divergent Change*;

III *Data Clumps*;

IV *Primitive Obsession*;

V *Temporary Field*;

VI *Duplicated Code*, quando uma reorganização estrutural contribui para reduzir ocorrências redundantes já existentes;

VII *Inappropriate Intimacy*, quando a redistribuição de responsabilidades reduz o acoplamento excessivo entre classes.

Nesses dois últimos casos, *Duplicated Code* e *Inappropriate Intimacy* são definições usadas por Fowler *et al.* [33], porém, foram extintas ou incorporadas a outras definições de *smells* na atualização de taxonomia realizada por Fowler [32]. Para eles, a aplicação de *Extract Class* pode não eliminar o problema por completo, pois atua como um mecanismo de mitigação estrutural, reduzindo sintomas associados e preparando o terreno para refatorações adicionais mais específicas (como *Move Function (Move Method)*, *Extract Function (Extract Method)*).

Portanto, uma operação de refatoração, de forma geral, conforme delineada por Fowler [32], deve ser compreendida como um mecanismo disciplinado de evolução arquitetural. Ao preservar o comportamento observável e atuar em pequenas etapas verificáveis, ela evita a degradação estrutural típica de sistemas que passam por múltiplas modificações (ou evoluções) sem reavaliação de *design*.

Além de melhorar a organização do código, a refatoração contribui diretamente para a qualidade interna do software — fator determinante para sua longevidade e sustentabilidade. Por meio de práticas como *Move Function* e *Extract Class*, o código torna-se mais expressivo, flexível e coerente com princípios fundamentais, como coesão, baixo acoplamento e responsabilidade única. Assim, embora uma refatoração seja, em essência, uma pequena alteração estrutural, ela frequentemente envolve operações complementares. Por exemplo, *Extract Class* geralmente implica a aplicação subsequente de *Move Function* e *Move Field*.

Nesse contexto, é importante destacar que a literatura sobre *CS* e refatorações emprega, ao longo do tempo, diferentes taxonomias e denominações. As classificações propostas por Fowler *et al.* em 1999 [33] e a atualização de 2018 [32] coexistem e são amplamente utilizadas na comunidade científica. Considerar apenas a versão mais recente como definitiva pode limitar a abrangência de revisões bibliográficas e análises comparativas, uma vez que muitos estudos empregam a taxonomia original, bem como outras denominações de *code smells* presentes na literatura, como *bad smells*, *design smells* e *code anomalies*. Reconhecer essa diversidade terminológica é, portanto, essencial para garantir rigor e completude na investigação do tema.

Desse modo, a refatoração não apenas mantém o sistema operacionalmente estável, mas também restabelece continuamente a integridade conceitual do *design*, atuando como um elo fundamental entre as práticas de desenvolvimento incremental e os princípios da Engenharia de Software orientada à manutenção e à evolução sustentável.

2.3 Large Language Models (LLMs)

Os LLMs representam um dos avanços mais significativos da Inteligência Artificial (IA) contemporânea, especialmente no âmbito do Processamento de Linguagem Natural (PLN). Construídos sobre arquiteturas de aprendizado profundo e treinados em corpora de grande escala, são capazes de reconhecer padrões linguísticos complexos e de compreender, gerar e manipular linguagem humana mesmo em uma ampla variedade de contextos, com níveis de coerência antes considerados inviáveis [31, 75, 21]. Essa capacidade de produzir respostas contextualizadas, estruturar informações e executar tarefas cognitivas intensivas consolida os LLMs como componentes centrais na evolução recente dos sistemas inteligentes [31].

O desenvolvimento desses modelos é fortemente fundamentado na arquitetura *Transformer*, cuja mecânica de autoatenção permite modelar dependências entre *tokens* independentemente de sua posição na sequência [31, 79]. O processo de treinamento ocorre, em geral, em duas etapas complementares: (i) pré-treinamento, no qual o modelo aprende padrões gerais de linguagem a partir de corpora heterogêneos; e (ii) ajuste fino (*fine-tuning*), dedicado à adaptação a tarefas específicas, como, por exemplo, sumarização, tradução, classificação textual e geração de código [37].

Pesquisas recentes [43, 37, 31, 68, 5] destacam que o aumento da escala dos modelos favorece o surgimento de capacidades emergentes, mas também introduz novos mecanismos de falha, ampliando desafios relacionados à confiabilidade e ao comportamento inesperado, incluindo a possibilidade de manifestar padrões de comportamento inesperado considerados como *CS*.

Os LLMs podem ser organizados em três categorias arquiteturais: (i) modelos *encoder-only*, focados majoritariamente na compreensão; (ii) modelos *encoder-decoder*, adequados a tarefas de transformação de sequências; e (iii) modelos *decoder-only*, orientados à geração de conteúdo [37]. Essa taxonomia influencia diretamente o desempenho, o custo computacional e a adequação de cada modelo a diferentes atividades.

No contexto do desenvolvimento de software, esses modelos vêm sendo utilizados em tarefas como: (i) geração de trechos de código; (ii) preenchimento automático contextual; (iii) sumarização e explicação de funcionalidades; (iv) suporte à depuração; (v) detecção preliminar de defeitos; e (vi) recomendação de refatorações [21, 12, 62, 17, 84]. Evidências indicam que podem reduzir o esforço cognitivo para compreender sistemas complexos e acelerar processos de manutenção, sobretudo quando combinados a técnicas clássicas de análise estática e a boas práticas de engenharia.

Apesar desse potencial, há desafios relevantes associados ao emprego de LLMs em atividades que exigem precisão e confiabilidade. Entre os principais riscos, destaca-se a geração de informações incorretas ou inconsistentes denominadas *alucinação* [31, 62, 4]. São exemplos de alucinações a reprodução de vieses dos dados de treinamento, a opacidade dos processos de inferência e a dificuldade de avaliação sistemática do desempenho em tarefas complexas, especialmente aquelas que envolvem código-fonte [37].

Além disso, ferramentas amplamente adotadas na prática, como o GitHub Copilot [16], podem sugerir trechos com vulnerabilidades, dependências obsoletas ou práticas inseguras, o que evidencia a necessidade de revisão humana e de mecanismos robustos de validação.

Logo, os LLMs devem ser compreendidos como instrumentos poderosos, porém complementares ao julgamento técnico especializado. Sua incorporação ao ciclo de desenvolvimento exige protocolos de governança, diretrizes de uso criterioso e estratégias de mitigação de riscos que assegurem a conformidade com os requisitos de qualidade, segurança e manutenibilidade.

Com o intuito de oferecer uma visão inicial e organizada do ecossistema de modelos de linguagem utilizados nesta proposta, a Figura 2.14 apresenta um mapa mental que sintetiza a divisão fundamental entre LLMs proprietários e *open-source*.

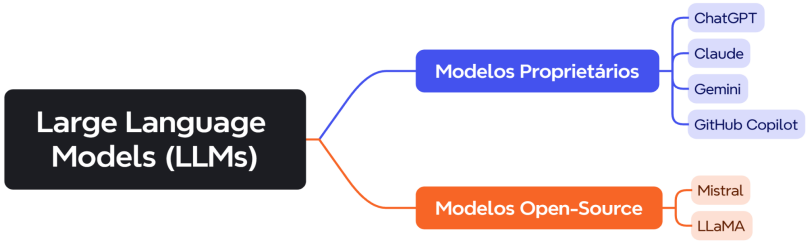

Figura 2.14: Mapa Mental Comparativo de LLMs

A Figura 2.14 destaca, de forma objetiva, alguns dos principais modelos representativos de cada categoria, permitindo uma compreensão rápida da classificação adotada na presente análise. Essa visualização introdutória situa o leitor diante das duas vertentes tecnológicas que estruturam o panorama contemporâneo dos modelos de linguagem.

As próximas subseções aprofundam os elementos essenciais que compõem esse panorama: a Subseção 2.3.1 descreve os principais modelos proprietários; já a Subseção 2.3.2, os modelos *open-source*; a Subseção 2.3.3 apresenta as plataformas que viabilizam a execução, distribuição e experimentação com esses modelos; a Subseção 2.3.4 discute suas capacidades e limitações; e, por fim, a Subseção 2.3.5 examina seu emprego na detecção e interpretação de *CS*. Essa organização fornece a base conceitual necessária para sustentar as análises desenvolvidas nos capítulos subsequentes.

2.3.1 Modelos Proprietários

Os modelos proprietários de LLMs constituem um eixo central no ecossistema contemporâneo de IA aplicada à Linguagem Natural, sendo desenvolvidos e mantidos por empresas que controlam tanto os pesos internos quanto os detalhes de seu treinamento.

No que se refere aos pesos, trata-se dos parâmetros aprendidos pelo modelo, ajustados durante o processo de otimização e responsáveis por determinar sua capacidade de generalização [83]. Já o treinamento consiste no processo pelo qual esses parâmetros são ajustados por meio de aprendizado profundo aplicado a grandes volumes de dados textuais,

utilizando algoritmos de retropropagação para internalizar padrões linguísticos [79]. Esse pré-treinamento confere aos modelos competências gerais de compreensão e de geração de linguagem, que posteriormente podem ser especializadas por meio de *fine-tuning* para tarefas específicas [74].

Os modelos proprietários são disponibilizados ao público por meio de interfaces web e APIs, oferecendo alto desempenho, estabilidade operacional e integração com ecossistemas consolidados [59, 35, 16, 8]. Contudo, esse caráter fechado impõe limitações à transparência, à auditabilidade e à reprodutibilidade científica, demandando cautela especial em contextos acadêmicos e industriais.

No escopo desta proposta, são analisados quatro modelos proprietários amplamente utilizados: ChatGPT (OpenAI) [59], Claude (Anthropic) [8], Gemini (Google DeepMind) [35] e GitHub Copilot (GitHub/OpenAI) [16]. Cada modelo apresenta características específicas relacionadas à multimodalidade, à profundidade de raciocínio, à integração com ferramentas de desenvolvimento e ao suporte a fluxos de trabalho da ES. As subseções a seguir descrevem cada uma delas com base em suas documentações oficiais, destacando funcionalidades, modos de acesso e aspectos relevantes para manutenção, compreensão de código e refatoração assistida.

ChatGPT (OpenAI)

O ChatGPT, desenvolvido pela OpenAI, é um modelo conversacional baseado na família GPT (*Generative Pre-trained Transformer*), cujas versões incluem GPT-3, GPT-3.5, GPT-4 e a variante multimodal GPT-4o [59, 57]. De acordo com as informações oficiais, essas versões incorporam aprimoramentos sucessivos em contextualização, raciocínio e suporte a diferentes modalidades de entrada, refletindo o avanço contínuo da plataforma [57].

O modelo é acessado por meio de uma interface interativa em formato de diálogo, que possibilita a inserção de *prompts* em linguagem natural, o envio de arquivos e o uso de funcionalidades multimodais conforme o plano habilitado [59]. Além da interface web, a OpenAI oferece uma API pública que permite integrações com aplicações externas, agentes inteligentes e ferramentas de apoio ao desenvolvimento de software. O serviço é disponibilizado em diferentes modalidades (*Free*, *Plus*, *Pro* e *Enterprise*), que variam em capacidade, prioridade de uso e recursos adicionais [58].

A Figura 2.15 exibe a interface oficial do ChatGPT, estruturada em um ambiente de mensagens organizado, facilitando o acompanhamento do histórico, o refinamento iterativo de instruções e o uso de interações persistentes, incluindo capacidades multimodais quando disponíveis [59].

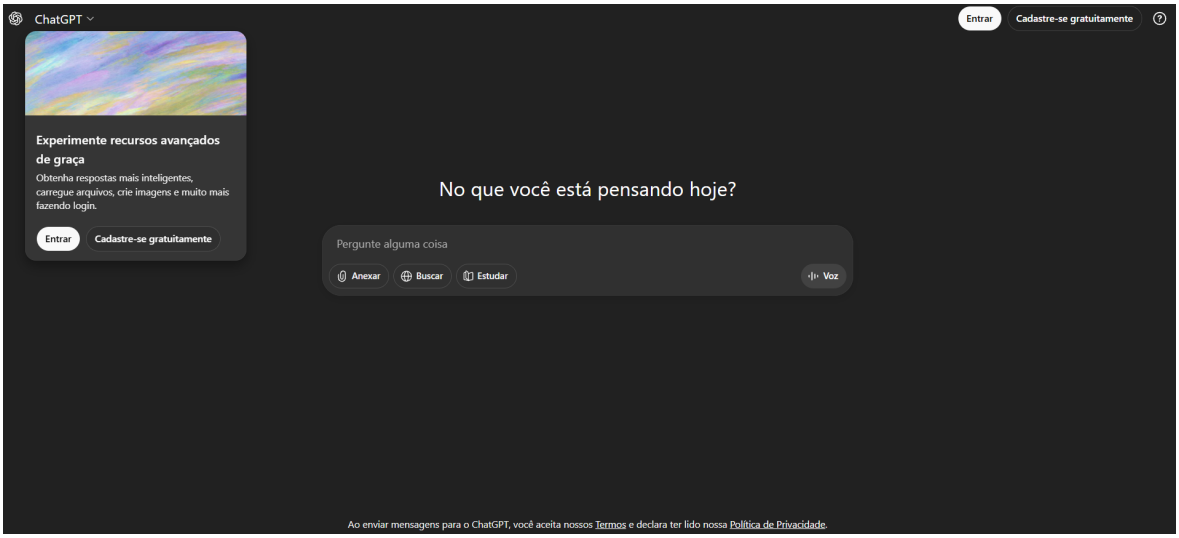

Figura 2.15: Interface Web do ChatGPT. Fonte: OpenAI (2025) [59].

No contexto da ES, a documentação oficial indica que o ChatGPT pode auxiliar na geração inicial de trechos de código, na explicação textual de funções, na elaboração de descrições técnicas e na análise preliminar de requisitos [57]. A OpenAI enfatiza, entretanto, a necessidade de verificação humana contínua, uma vez que o modelo pode apresentar respostas incorretas ou inconsistentes, incluindo fenômenos associados às alucinações [59].

Claude (Anthropic)

O Claude, desenvolvido pela Anthropic, é apresentado como um assistente de IA projetado para operar com altos níveis de segurança, precisão e alinhamento ético, fundamentado nos princípios de *Constitutional AI* [10, 8]. A família de modelos inclui versões como Claude 3, Claude 3.5 Sonnet e a variante mais recente, Claude Sonnet 4.5, que incorpora avanços expressivos em raciocínio, codificação e autonomia no uso de ferramentas computacionais [11].

O acesso ao modelo ocorre por meio de uma interface de conversação baseada em mensagens, disponível no navegador e em dispositivos móveis, com integração adicional via API

[8, 9]. Essa interface — ilustrada na Figura 2.16 — permite o envio de textos, arquivos e instruções complexas, preservando o histórico da sessão e suportando interações prolongadas. A Anthropic enfatiza ainda o suporte a janelas de contexto ampliadas, adequadas ao processamento de documentos extensos, bases de código e artefatos técnicos volumosos [10]. Complementarmente, a API fornece recursos voltados à automação corporativa e à criação de agentes inteligentes [9].

Figura 2.16: Interface Web do Claude. Fonte: Anthropic (2025) [8].

Entre as funcionalidades oferecidas, destaca-se o recurso *Artifacts*, que permite gerar e editar websites, documentos, gráficos e código diretamente ao lado da conversa [8]. Com o Claude Sonnet 4.5, foram introduzidas melhorias substanciais para tarefas de programação, incluindo suporte a fluxos de raciocínio longos, integração com o VS Code por meio de extensão nativa e recursos como *checkpoints* de codificação e memória para execução contínua de agentes [11]. A plataforma também inclui mecanismos de mitigação contra ataques de injeção de *prompt* [11].

A API do Claude disponibiliza funcionalidades avançadas, como *Tool Use*, suporte multimodal, o protocolo *Model Context Protocol* (MCP), guias de integração, exemplos de código e estratégias para engenharia de *prompt*, além de ferramentas de *cache* de respostas para otimização de custo e desempenho [9].

Não obstante a Anthropic forneça documentação detalhada sobre funcionalidades e aplicações, elementos internos, como o número de parâmetros, *datasets* utilizados e a arquitetura completa, permanecem não divulgados, o que limita análises técnicas aprofundadas. Mesmo

assim, a ênfase contínua em segurança, alinhamento e mitigação de comportamentos inadequados torna Claude uma alternativa relevante para ambientes de produção que exigem previsibilidade e confiabilidade [11].

No contexto da ES, Claude destaca-se por suas capacidades de análise e explicação de código, sumarização de módulos, apoio à documentação, depuração preliminar e auxílio ao design de componentes [10]. A versão Sonnet 4.5, segundo informações oficiais, figura entre os melhores modelos recentes em *benchmarks* específicos, apresentando avanços notáveis em raciocínio e execução de fluxos de trabalho extensos [11].

Gemini (Google DeepMind)

O Gemini, desenvolvido pelo Google DeepMind, representa a geração mais recente de modelos multimodais da empresa, sucedendo sistemas anteriores como o Bard. Segundo o site oficial [35], trata-se de um modelo projetado para operar de forma intrinsecamente multimodal, processando texto, imagens, áudio, vídeo e código em um único fluxo de interação. Essa característica favorece seu uso em tarefas que envolvem análise integrada de diferentes modalidades, como documentos extensos, bases de código ou artefatos audiovisuais.

A família de modelos inclui variantes como *Ultra*, *Pro*, *Nano*, *2.5 Pro* e *3 Pro*, cada uma destinada a cenários computacionais específicos — desde dispositivos móveis até aplicações corporativas em larga escala [1, 15]. Essas versões diferem em profundidade de raciocínio, capacidade multimodal, velocidade de inferência e suporte a janelas de contexto ampliadas.

A interface principal do Gemini, exibida na Figura 2.17, combina recursos de conversação com ferramentas multimodais, permitindo o envio de arquivos, imagens, vídeos e documentos diretamente no ambiente interativo [35]. O modelo também está integrado ao ecossistema Google, abrangendo serviços como Gmail, Google Workspace, YouTube e Google Fotos, o que facilita sua incorporação em fluxos de trabalho já consolidados [35].

Figura 2.17: Interface Web do Gemini. Fonte: Google (2025) [35].

Para uso corporativo, o Gemini é disponibilizado por meio do *Google Cloud Vertex AI*, que oferece ambientes especializados para automação, integração e desenvolvimento de soluções baseadas em modelos generativos [15]. A plataforma inclui documentação técnica, APIs e ferramentas específicas para manipulação de grandes bases de código, análise de dados e geração de conteúdo em escala.

No contexto da ES, o Gemini demonstra aplicabilidade em etapas como concepção, codificação, depuração, testes e operações. A documentação do Google Cloud destaca sua capacidade de analisar milhares de linhas de código em uma única sessão, identificar padrões estruturais, sugerir refatorações e apoiar a compreensão de bases extensas [15]. Informações oficiais também apontam que o modelo oferece suporte ao raciocínio avançado e ao processamento de grandes volumes de dados multimodais, reforçando sua relevância para aplicações em manutenção e Engenharia de Software Assistida por IA [1].

GitHub Copilot (GitHub/OpenAI)

O GitHub Copilot é um assistente de programação baseado em modelos da família GPT, desenvolvido pela GitHub em colaboração com a OpenAI. Conforme descrito pela documentação oficial [16], o Copilot fornece assistência contextualizada ao longo de todo o ciclo de desenvolvimento, atuando como um "programador em par" capaz de apoiar desde a escrita de código até explicações funcionais e respostas a consultas em linguagem natu-

ral. A ferramenta tem sido amplamente adotada por desenvolvedores individuais e equipes, consolidando-se como um dos recursos de IA mais difundidos no ecossistema de software [22].

Sua operação ocorre principalmente por meio de integração com IDEs como Visual Studio Code, Visual Studio, JetBrains IDEs e Neovim [16]. A interface conversacional integrada ao editor — ilustrada na Figura 2.18 — permite que o usuário solicite explicações, refatorações, geração de arquivos, criação de testes e análise de mudanças diretamente no ambiente de desenvolvimento. O Copilot também está disponível no GitHub Mobile, na linha de comando via GitHub CLI e em painéis interativos no próprio site do GitHub [22].

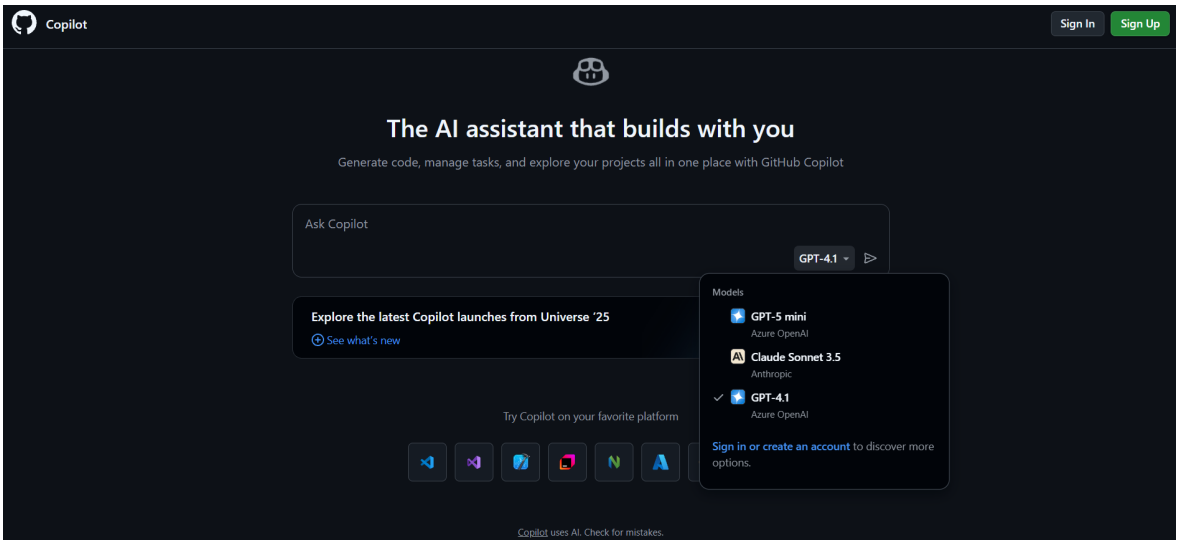

Figura 2.18: Interface Web do GitHub Copilot. Fonte: GitHub (2025) [16].

As funcionalidades do Copilot incluem sugestões de código baseadas no arquivo em edição, geração completa de funções, assistência na escrita de testes, explicação de comportamentos e criação de *pull requests* com descrições automáticas [16]. Planos individuais e corporativos (*Free*, *Pro*, *Pro+* e *Business*) organizam o acesso conforme limites de uso, velocidade de resposta e capacidades adicionais [23].

No contexto da ES, o Copilot possui aplicabilidade direta em atividades de compreensão de código, revisão de trechos existentes, análise de repositórios, exploração de projetos legados e identificação preliminar de melhorias [16]. O chat integrado permite solicitar explicações detalhadas sobre funções, recomendações de refatoração, instruções de correção de erros e estratégias de teste [23]. Estudos citados pela própria plataforma apontam ganhos

de até 55% na velocidade de escrita de código sem perda de qualidade [16], reforçando seu papel como ferramenta de apoio cognitivo.

Em síntese, os modelos proprietários analisados – ChatGPT (OpenAI), Claude (Anthropic), Gemini (Google DeepMind) e GitHub Copilot (GitHub/OpenAI) – representam soluções maduras e amplamente utilizadas no ecossistema de desenvolvimento contemporâneo. Cada um deles apresenta arquiteturas, capacidades multimodais e formas de integração que os tornam especialmente relevante para tarefas de apoio à ES, como a compreensão de código, a automação de rotinas, a sugestão de refatorações e o suporte cognitivo ao desenvolvedor.

Ainda que ofereçam desempenho elevado e uma forte integração com plataformas consolidadas, esses modelos operam sob restrições típicas de soluções fechadas, como a ausência de transparência completa, a dependência de provedores e a menor auditabilidade científica. Assim, sua utilização deve considerar tanto o potencial técnico quanto os limites estruturais impostos pelo seu caráter proprietário.

Por fim, é apresentada a Figura 2.19, que sintetiza visualmente as características comparativas dos quatro modelos proprietários analisados. O mapa mental organiza de forma estruturada os principais aspectos discutidos no texto, distribuídos em quatro dimensões centrais: (i) multimodalidade; (ii) aplicações em Engenharia de Software; (iii) pontos fortes; e (iv) limitações. Essa representação gráfica permite uma visualização rápida das similaridades e diferenças entre os modelos, reforçando elementos já discutidos na análise textual e facilitando a compreensão integrada das capacidades e restrições de cada solução proprietária no contexto desta proposta.

Em seguida, a Subseção 2.3.2 apresenta modelos *open source* e ferramentas para execução local, ampliando o panorama de alternativas disponíveis e permitindo uma análise comparativa entre abordagens fechadas e abertas no uso de LLMs aplicados à Engenharia de Software.

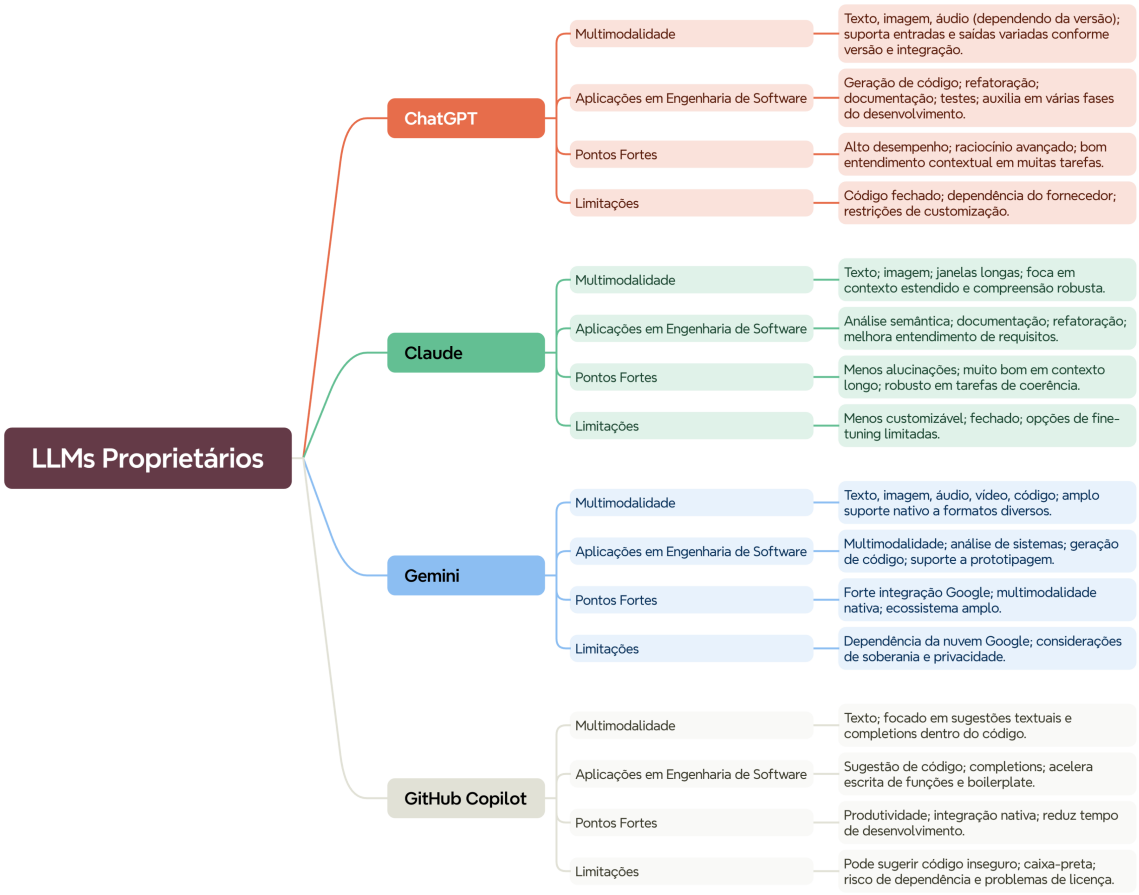

Figura 2.19: Mapa Mental Comparativo dos Modelos Proprietários

2.3.2 Modelos Open Source

A expansão dos modelos de linguagem de código aberto tem redefinido o ecossistema de LLMs ao oferecer um conjunto de capacidades que priorizam a autonomia tecnológica, a inspeção transparente e maior controle na experimentação. Diferentemente das soluções proprietárias, as arquiteturas *open source* permitem execução local, adaptação por *fine-tuning* e integração direta em pipelines personalizados, eliminando a dependência de provedores fechados.

Nesse âmbito, tanto a família LLaMA quanto os modelos da Mistral AI têm evoluído com foco em multimodalidade, janelas de contexto ampliadas e variantes otimizadas para tarefas específicas, como programação, raciocínio e análise de documentos.

Esses avanços mostram como a abertura dos modelos tem impulsionado novas possibili-

dades de pesquisa e desenvolvimento, permitindo abordagens especializadas de manutenção de software, experimentação em larga escala e fluxos de trabalho compatíveis com requisitos de privacidade, governança e controle de dados. Nas subseções seguintes, são detalhadas as arquiteturas LLaMA e Mistral AI, suas capacidades, formas de acesso e aplicações práticas no contexto de ES.

LLaMA (Meta AI)

A família LLaMA [47], desenvolvida pela Meta, representa uma das mais importantes iniciativas de modelos de linguagem de código aberto da atualidade. Segundo a documentação oficial [47, 46], esse conjunto de modelos foi projetado para oferecer alta eficiência, execução local, facilidade de customização e ampliada acessibilidade para pesquisadores e desenvolvedores. As versões mais recentes, reunidas sob o nome LLaMA 4, introduzem avanços significativos em multimodalidade, raciocínio e escalabilidade, estabelecendo a linha entre as alternativas abertas mais competitivas do ecossistema de LLMs [47, 46].

A arquitetura LLaMA 4 inclui variações como *Llama 4 Scout*, *Llama 4 Maverick* e *Llama 4 Behemoth Preview*, cada uma orientada a diferentes exigências de capacidade computacional, profundidade de raciocínio e suporte multimodal [47]. Esses modelos adotam multimodalidade nativa, com fusão precoce de texto e visão (integração conjunta entre entradas textuais e visuais durante o processamento), permitindo o uso integrado de imagens e instruções textuais.

Entre as funcionalidades principais, destacam-se o suporte à execução local, quantização, *fine-tuning*, personalização para tarefas específicas e compatibilidade com múltiplos ambientes, incluindo Linux, Windows, macOS e plataformas em nuvem [46]. A documentação oficial também disponibiliza recursos como *Llama Cookbook*, *Llama Stack*, guias de integração e materiais suplementares para facilitar a adoção dessas versões em contextos de pesquisa e desenvolvimento [46].

Do ponto de vista operacional, a família LLaMA pode ser utilizada de diferentes maneiras, incluindo execução local em infraestrutura própria, hospedagem em nuvem ou acesso por meio de APIs disponibilizadas pela Meta [47, 46]. Além dessas formas tradicionais de uso, a empresa também oferece interfaces próprias que permitem interação direta com os modelos, ampliando as possibilidades de acesso sem necessidade de configuração técnica.

A Figura 2.20 ilustra um desses modos de acesso: a integração da Meta AI — baseada nos modelos LLaMA — ao WhatsApp, possibilitando interação com o modelo diretamente em aplicativos móveis [45]. Essa solução demonstra a versatilidade das arquiteturas LLaMA e sua capacidade de operar em plataformas amplamente disseminadas.

Figura 2.20: Interface Meta AI via WhatsApp. Fonte: Whatsapp (2025) [45].

De forma complementar, a Meta disponibiliza uma interface web integrada à família LLaMA, voltada a consultas, experimentação e uso multimodal no navegador [2]. Essa interface, ilustrada na Figura 2.21, oferece um ambiente acessível para testar funcionalidades do modelo, sem dependência de configuração local, complementando o uso via WhatsApp ao apresentar outro caminho de interação, amplo e facilmente acessível.

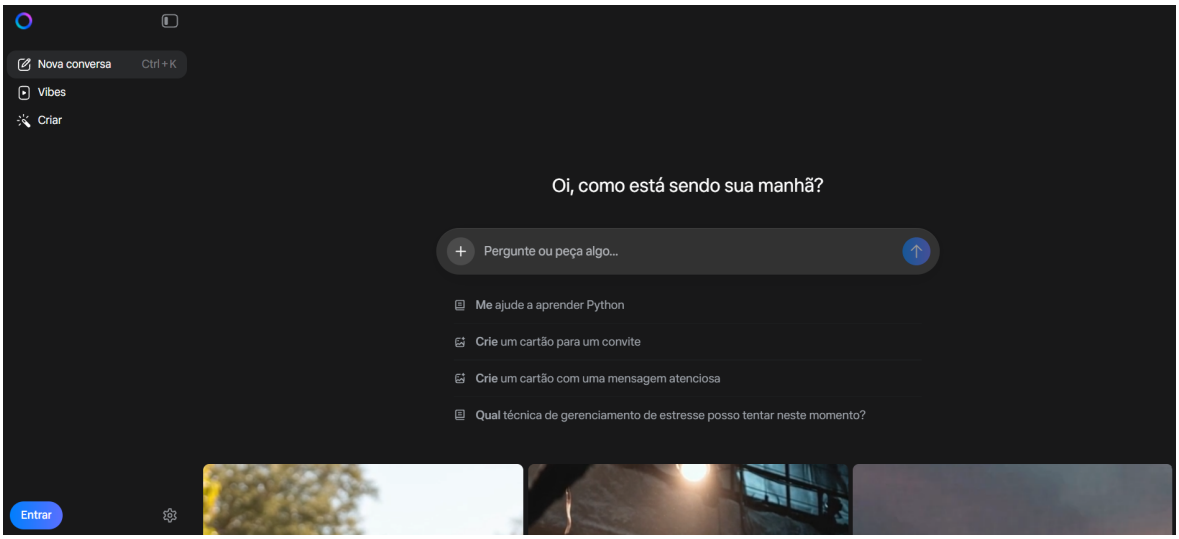

Figura 2.21: Interface Meta AI via Web. Fonte: Meta (2025) [2].

No contexto da ES, a família LLaMA tem se mostrado particularmente útil para atividades que exigem análise profunda de bases de código extensas, explicação de módulos, compreensão de estruturas arquiteturais, auxílio à refatoração e suporte a pipelines de manutenção. A janela de contexto ampliada das versões mais recentes amplia as possibilidades de uso em tarefas que demandam visão global de grandes sistemas, enquanto sua natureza aberta favorece a experimentação, a auditabilidade e a autonomia tecnológica [46].

Apesar de seu potencial, a família LLaMA apresenta limitações relevantes: nem todas as variantes são totalmente abertas. Algumas versões são disponibilizadas sob modelos de licença "*open-weight*", o que restringe parcialmente a liberdade de redistribuição e de uso comercial irrestrito. Além disso, faltam informações completas sobre *datasets* e parâmetros, e os modelos maiores exigem hardware especializado para execução local. Esses fatores reduzem a transparência científica e podem restringir sua adoção em ambientes com recursos limitados.

Por fim, a disponibilização das arquiteturas por meio de plataformas como Hugging Face [30] e Kaggle [42] facilita a distribuição e a reprodutibilidade científica, enquanto a execução local reduz a dependência de provedores proprietários. A natureza *open-source* da família LLaMA torna esses modelos particularmente relevantes para esta proposta, pois amplia o conjunto de alternativas para análise, detecção de *CS* e experimentação em fluxos de Engenharia de Software apoiada por IA.

Mistral AI

A família de modelos da Mistral AI [49] representa uma iniciativa de fronteira em LLMs, com foco em eficiência, customização e execução em diferentes infraestruturas, incluindo ambientes locais, nuvem pública e cenários híbridos [49, 48]. No catálogo de modelos, a Mistral AI disponibiliza arquiteturas otimizadas para tarefas gerais e para domínios especializados, incluindo variantes dedicadas ao raciocínio, à análise de documentos e à geração de código [48].

Entre os exemplos, destacam-se modelos de uso geral para o diálogo e a compreensão da linguagem, bem como modelos voltados à programação, como *Codestral*, projetado especificamente para fluxos de trabalho de desenvolvimento de software [48]. A empresa também oferece modelos *edge*, desenhados para execução em dispositivos com recursos computacionais limitados, mantendo desempenho competitivo em cenários de baixa latência [48].

Do ponto de vista operacional, a Mistral AI oferece três modalidades principais de uso: (i) implantação *self-hosted* (incluindo ambientes *on-premises* e *edge*); (ii) acesso via Mistral Cloud; e (iii) integração com provedores de nuvem externos, como Google Cloud, AWS, Azure e outras plataformas parceiras [49, 48].

Em paralelo aos modelos, a empresa disponibiliza uma plataforma de desenvolvimento (*AI Studio*) voltada à criação de aplicações, agentes e fluxos de trabalho baseados em LLMs, com suporte a *fine-tuning*, destilação e ajustes orientados ao domínio [49].

No eixo de interação conversacional, a Mistral AI disponibiliza o *Le Chat* [3], um assistente multilíngue e multimodal que permite explorar seus modelos por meio de uma interface de *chat* acessível no navegador e em aplicativos móveis. Essa plataforma suporta o envio de *prompts* em linguagem natural, a análise de documentos, o auxílio em tarefas de programação e a criação de agentes personalizados, atuando como um ponto de entrada intuitivo para experimentação e uso cotidiano [49, 3].

A Figura 2.22 ilustra a interface central do ecossistema Mistral AI, destacando a organização visual que integra o acesso aos modelos, às ferramentas de desenvolvimento e ao próprio assistente conversacional. No contexto da ES, a Mistral AI oferece recursos diretamente aplicáveis ao desenvolvimento e à manutenção de software. O modelo *Codestral*, por exemplo, é descrito como otimizado para a geração e a compreensão de código, com foco em fluxos de trabalho de desenvolvimento [48].

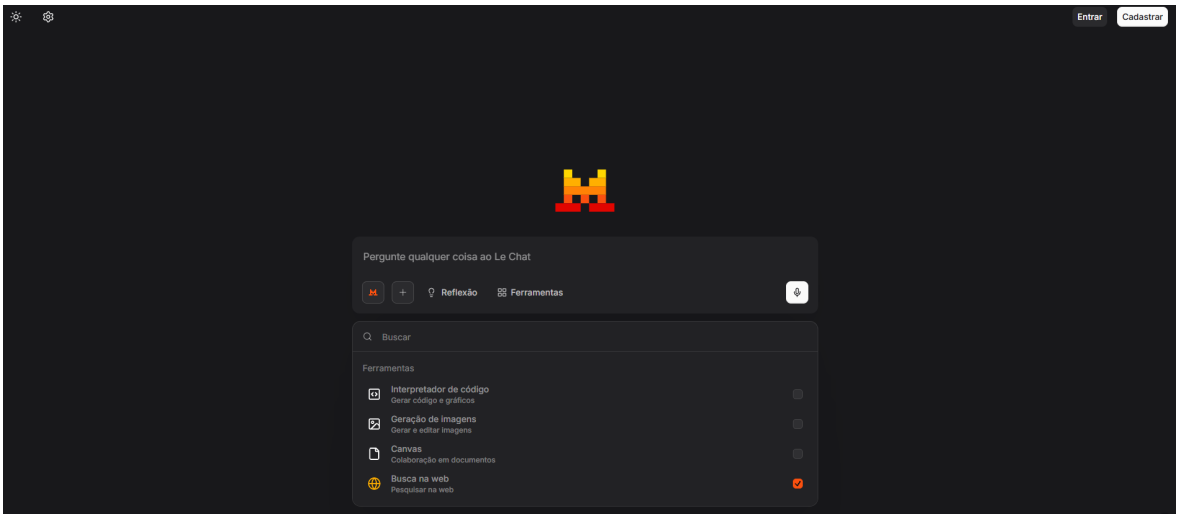

Figura 2.22: Interface do Mistral AI. Fonte: Mistral AI (2025) [49].

Como desafio, a adoção dos modelos Mistral pode ser prejudicada pela escassez de informações técnicas detalhadas e pela existência de versões exclusivamente comerciais. Além disso, a execução local de modelos grandes requer recursos computacionais que nem sempre estão disponíveis em ambientes de pesquisa.

Apesar disso, o suporte à implantação local, às licenças *open-weight* e aos mecanismos de personalização (como *fine-tuning* e treinamento especializado) torna essa família de modelos particularmente atraente para cenários que requerem autonomia tecnológica, maior controle sobre dados sensíveis e experimentação em tarefas como a análise de bases de código, o apoio a refatorações [49, 48, 3] e a investigação de *CS*.

De modo geral, as arquiteturas abertas analisadas – LLaMA e Mistral – ampliam significativamente as possibilidades de uso de LLMs em ES. Seus principais pontos fortes concentram-se na execução local, na personalização por *fine-tuning*, na autonomia sobre os pesos e no controle de dados sensíveis. Esses fatores tornam os modelos particularmente adequados para cenários que requerem reprodutibilidade, experimentação e integração em pipelines de manutenção, de análise de código e de apoio à refatoração.

Entretanto, tais benefícios coexistem com limitações relevantes. A ausência de parâmetros completamente transparentes em todas as variantes de cada modelo, a necessidade de maior familiaridade técnica para a implantação local e eventuais lacunas de documentação, quando comparadas aos ecossistemas proprietários, ainda representam barreiras à adoção mais ampla. Além disso, em alguns casos, o desempenho desses modelos pode depender

fortemente de ajustes específicos ou de infraestrutura especializada.

Assim, LLaMA e Mistral compõem um conjunto de alternativas valiosas. Todavia, requerem avaliação criteriosa conforme o ambiente de uso, o equilíbrio entre custo computacional, governança e a maturidade das ferramentas.

Adicionalmente, apresenta-se a Figura 2.23, a qual sintetiza visualmente as principais características dos modelos *open source* analisados. O mapa mental organiza, de maneira estruturada, os aspectos essenciais discutidos no texto, distribuídos nas quatro dimensões centrais que orientam esta avaliação: (i) multimodalidade; (ii) aplicações em ES; (iii) pontos fortes; e (iv) limitações.

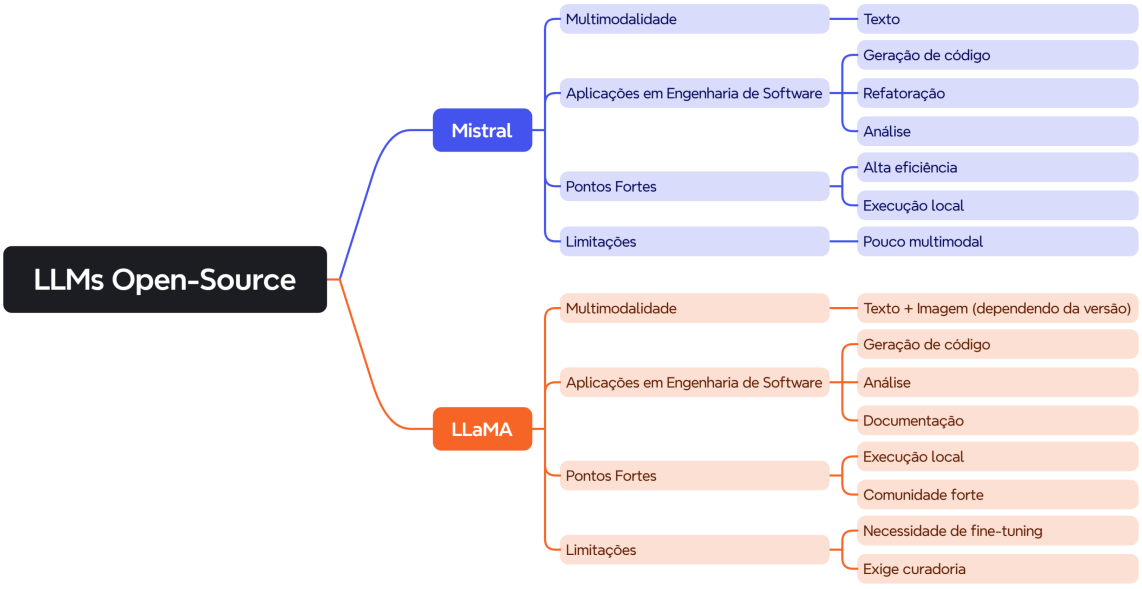

Figura 2.23: Mapa Mental Comparativo dos Modelos Open Source

Essa representação gráfica mostrada na Figura 2.23, possibilita uma visualização imediata das similaridades e diferenças entre Mistral e LLaMA, reforçando os elementos já discutidos na análise textual e facilitando a compreensão integrada de suas capacidades, potencialidades e restrições no contexto do desenvolvimento e da pesquisa com modelos de código aberto.

Já a Subseção 2.3.3 apresenta plataformas que viabilizam a execução e distribuição desses modelos, em particular o Ollama [56] e o Hugging Face Hub [30], essenciais tanto para o uso local quanto para o acesso colaborativo à comunidade de modelos, *datasets* e ferramentas associados ao ecossistema *open source*.

2.3.3 Plataformas de Execução Local e Ecossistemas Open Source

A crescente adoção de modelos abertos, como LLaMA e Mistral, impulsionou o uso de ferramentas que permitam executar, gerenciar e integrar LLMs fora de serviços proprietários. Nesse contexto, plataformas como Ollama [56] e Hugging Face Hub [30] oferecem a infraestrutura necessária para utilização desses modelos de forma flexível, seja em ambientes locais, híbridos ou conectados à nuvem.

Essas plataformas possibilitam operar LLMs em cenários que demandam privacidade, controle sobre dados, personalização ou independência tecnológica. Estas condições são frequentes em atividades de manutenção de software, análise de bases de código, detecção de *CS* e condução de experimentos reprodutíveis. Sua adoção também favorece a criação de *pipelines* específicos e de ambientes de teste controlados, reduzindo a dependência de infraestruturas fechadas e facilitando a integração de modelos em fluxos de ES.

Assim, as subseções a seguir apresentam as duas plataformas utilizadas neste trabalho: o Ollama, voltado à execução local e ao controle direto da inferência, e o Hugging Face Hub, que atua como o principal repositório colaborativo de modelos e *datasets* de IA de código aberto. Em conjunto, essas soluções complementam o ecossistema de LLMs discutido anteriormente e ampliam as possibilidades de experimentação em ES.

Ollama

A plataforma Ollama [56] permite executar modelos de linguagem diretamente em computadores pessoais ou servidores locais, sem necessidade de depender exclusivamente de serviços em nuvem. Compatível com Windows, macOS e Linux, oferece instalação simplificada e suporte a uma ampla variedade de modelos abertos, incluindo *DeepSeek-R1*, *Gemma 3*, *GPT-OSS*, *Codestral*, *Code Llama* e outros especializados em programação, visão computacional e raciocínio estruturado [55, 54].

Entre suas funcionalidades, o Ollama permite baixar, organizar e executar modelos localmente, ajustar configurações básicas, criar variantes personalizadas e integrar os modelos a projetos de software. A interface de uso simplificada, tanto via aplicativo quanto por linha de comando, facilita a execução de experimentos e a incorporação dos modelos em ferramentas de análise e manutenção [55].

Operacionalmente, a plataforma oferece três modos de uso: (i) execução local direta, (ii) execução em contêineres Docker e (iii) acesso a modelos disponibilizados no Ollama Cloud [56]. Essa flexibilidade torna o Ollama adequado para ambientes com exigências de privacidade, restrições à nuvem ou necessidade de controle total sobre a inferência.

No contexto desta proposta, o Ollama é especialmente relevante por oferecer acesso local a modelos de geração e compreensão de código, permitindo experimentos reprodutíveis com autonomia tecnológica. Sua natureza aberta e sua comunidade ativa facilitam a personalização e a condução de estudos voltados à detecção e à análise de *CS*.

Hugging Face Hub

O Hugging Face Hub [30] constitui o maior ecossistema colaborativo de IA de código aberto, reunindo modelos, *datasets* e aplicações interativas em múltiplas modalidades (incluindo texto, imagem, áudio, vídeo e 3D). A plataforma oferece um ambiente abrangente para a descoberta, a experimentação e a integração de artefatos de IA em fluxos de trabalho personalizados.

Entre suas funcionalidades, o Hugging Face Hub disponibiliza repositórios versionados com suporte a *Git*, APIs para gerenciamento de modelos e *datasets*, ferramentas para *fine-tuning* e para implantação, além de filtros avançados por tarefa, biblioteca, idioma, licença e tipo de modelo [30]. Modelos amplamente utilizados, como, por exemplo, *Llama-2*, *Falcon*, *Stable Diffusion* e *WizardCoder*, possuem métricas de uso, histórico de versões e demonstrações integradas que auxiliam a seleção para diferentes cenários de pesquisa.

Os modos de acesso incluem navegação web gratuita, integração programática via biblioteca `huggingface_hub` e opções corporativas com segurança reforçada. A comunidade global do Hugging Face também contribui para a manutenção dos repositórios, favorecendo o compartilhamento científico e a reprodutibilidade.

Para esta proposta, o Hugging Face Hub desempenha um papel fundamental ao fornecer modelos abertos e *datasets* adequados para experimentos com LLMs voltados à ES. Sua organização colaborativa e a ampla disponibilidade de artefatos facilitam o acesso a modelos especializados, a construção de *pipelines* reprodutíveis e a comparação consistente entre diferentes arquiteturas.

A análise de Ollama e do Hugging Face Hub revela dois componentes complementares

do ecossistema de modelos abertos. Enquanto o Ollama privilegia execução local, controle sobre dados e facilidade de experimentação *offline*, o Hugging Face se destaca como um repositório global voltado à distribuição, à colaboração e à integração programática em larga escala. Em conjunto, essas plataformas ampliam significativamente a autonomia tecnológica e sustentam as etapas experimentais desta proposta, permitindo explorar modelos especializados em código, realizar análises reprodutíveis e apoiar a investigação de *CS* em sistemas reais, conforme discutido na Subseção 2.3.4.

2.3.4 LLMs na ES: Aplicações, Benefícios e Desafios

No domínio da ES, os LLMs têm se consolidado como ferramentas de apoio capazes de atuar em diferentes etapas do ciclo de desenvolvimento, ampliando a capacidade dos desenvolvedores de compreender, sintetizar e manipular código-fonte em ambientes complexos. Estudos recentes apontam que esses modelos podem auxiliar na geração contextualizada de trechos de código, na explicação de funcionalidades, na elaboração de documentação técnica e no suporte à depuração inicial [21, 12, 17, 84]. Esse conjunto de aplicações tem permitido reduzir o esforço cognitivo envolvido na navegação de bases de código extensas e heterogêneas, especialmente em sistemas legados ou em equipes de grande escala.

Ferramentas amplamente utilizadas no mercado reforçam esse papel. De acordo com a documentação oficial da OpenAI, o ChatGPT pode auxiliar na escrita de código, na análise de requisitos e na explicação de funções, atuando como um assistente generalista de ES [59, 57]. A família Claude, por sua vez, destaca-se por oferecer capacidades de raciocínio estendido, análise de documentos técnicos volumosos e apoio à revisão de artefatos de software [10, 11]. Já o Gemini, desenvolvido pelo Google DeepMind, incorpora processamento multimodal nativo e demonstra aplicabilidade na análise de milhares de linhas de código, na identificação de padrões e na sugestão de refatorações [15, 1]. Em ambiente de desenvolvimento integrado, o GitHub Copilot fornece assistência contextual direta em IDEs, facilitando a escrita de testes, a revisão de trechos existentes e a automatização de tarefas repetitivas [16, 22].

Modelos *open source* também ampliam a disponibilidade de recursos aplicáveis à ES. A família LLaMA, segundo a documentação oficial da Meta, oferece suporte à execução local, janelas de contexto expandidas e personalização por *fine-tuning*, possibilitando aná-

lises aprofundadas de bases de código e experimentos reprodutíveis [47, 46]. Os modelos da Mistral AI, incluindo variantes especializadas como o *Codestral*, foram projetados para tarefas de programação e automação, oferecendo eficiência e opções de implantação local ou híbrida [49, 48, 3]. A disponibilidade desses modelos em plataformas como Hugging Face [30] e sua integração facilitada por ferramentas como Ollama [56] reforçam o papel da comunidade aberta na expansão do uso de LLMs voltados ao desenvolvimento e à manutenção de software.

Apesar das oportunidades, a aplicação dos LLMs em ES apresenta desafios que vão além dos observados em tarefas de PLN. A literatura científica evidencia, por exemplo, que esses modelos ainda enfrentam dificuldades para capturar dependências semânticas profundas, interpretar implicações arquiteturais e compreender relações estruturais complexas entre componentes, conforme discutido por Szabó [75].

Além disso, estudos sobre ferramentas baseadas em LLMs, como o GitHub Copilot, demonstram que recomendações automáticas podem introduzir antipadrões, dependências obsoletas ou trechos vulneráveis [19, 70, 67, 38, 82]. Tais problemas estão diretamente associados ao fato de o modelo ser treinado em grandes repositórios públicos que incluem código de qualidade variável [22].

Outros trabalhos reforçam limitações adicionais dos LLMs, incluindo inconsistência lógica, falhas em raciocínio estruturado e dificuldade em manter alinhamento contínuo com boas práticas de ES, especialmente em cenários que envolvem múltiplos módulos, fluxos complexos ou análises profundas de sistemas [31, 62, 4].

Diante dessas evidências, autores [31, 70, 19, 75] e fornecedores [59, 8, 35, 49] enfatizam que o uso de LLMs em ES deve ser acompanhado de estratégias de verificação e curadoria, garantindo que as sugestões geradas sejam validadas conforme padrões técnicos e de qualidade. A natureza potencialmente falível desses modelos reforça a necessidade de supervisão humana contínua, especialmente em atividades críticas como refatoração, inspeção arquitetural ou análise de confiabilidade.

Assim, mesmo que os LLMs representem ferramentas poderosas para ampliar a produtividade, apoiar tarefas cognitivas e acelerar processos de desenvolvimento, sua adoção efetiva demanda integração criteriosa aos fluxos de trabalho e compreensão clara de suas limitações técnicas. Essa perspectiva fundamenta a Subseção 2.3.5, dedicada ao emprego direto desses

modelos na detecção e interpretação de *CS*, eixo central desta proposta.

2.3.5 Aplicação de LLMs no contexto de CS

A investigação sobre o uso de LLMs na detecção e interpretação de *CS* tem emergido como um eixo crescente na literatura de ES, sobretudo diante das limitações persistentes das ferramentas tradicionais de análise estática. Conforme discutido na Seção 2.1, a detecção automática de CS permanece restrita em cobertura e profundidade, enquanto a Seção 2.3 evidenciou que os LLMs possuem potencial para interpretar estruturas semânticas complexas, analisar dependências e produzir explicações contextuais. A articulação entre esses dois domínios tem motivado pesquisas que investigam até que ponto os LLMs podem complementar ou superar soluções clássicas baseadas em regras.

Estudos empíricos [65, 13] demonstram que ferramentas tradicionais fornecem suporte desigual aos *smells* clássicos de Fowler [33, 32], com forte predominância em linguagens como Java e cobertura limitada em linguagens amplamente utilizadas na indústria moderna, como JavaScript e TypeScript. Para os seis *CS* analisados nesta proposta (S29, S30, S34, S36, S37 e S39 – ver Subseção 2.1.1), observa-se falta de suporte direto nas ferramentas de mercado [73, 25, 78, 28]. Essa ausência de cobertura reduz a capacidade das ferramentas de detecção estrutural e reforça a necessidade de abordagens capazes de interpretar nuances semânticas e relações de dependência entre classes.

Nesse âmbito, a literatura recente tem investigado LLMs como mecanismos alternativos ou complementares de detecção de *CS*. Estudos [81, 12, 41] mostram que modelos modernos são capazes de reconhecer padrões estruturais recorrentes, analisar trechos de código em diferentes níveis de granularidade e propor recomendações de melhoria. Esses trabalhos empregam desde estratégias de *prompting* até instruções altamente especializadas, com ganhos variáveis de precisão e capacidade explicativa. De modo geral, os resultados sugerem que os LLMs podem identificar sintomas semânticos dificilmente capturados por heurísticas estáticas, ainda que a sensibilidade às instruções e ao contexto permaneça elevada.

Apesar do avanço recente das pesquisas, a literatura apresenta limitações metodológicas importantes. Estudos apontam a ausência de *benchmarks* padronizados para a avaliação comparativa de LLMs [68, 31, 37], o que dificulta a mensuração consistente do desempenho entre modelos, tarefas e estratégias de *prompting*. Também se observa forte heterogeneidade

nas métricas adotadas, variando de análises qualitativas a medidas subjetivas de acurácia, e falta de consenso sobre quais indicadores capturam adequadamente a capacidade de um modelo em detectar *CS*.

Além disso, a predominância de experimentos conduzidos em Java limita a generalização dos achados, deixando linguagens como TypeScript substancialmente subexploradas, apesar de sua ampla adoção no desenvolvimento de sistemas modernos [13]. Essa lacuna restringe o entendimento sobre como LLMs se comportam em ambientes de tipagem gradual ou estrutural, nos quais determinados *smells* podem manifestar-se de maneiras distintas.

Outro desafio substancial refere-se às limitações históricas das ferramentas tradicionais de análise estática. Conforme discutido por Sharma [66], essas ferramentas apresentam baixa cobertura para *CS* menos frequentes, como *Parallel Inheritance Hierarchies* e *Cyclic Hierarchy*, além de inconsistência no suporte a diversas linguagens e a ecossistemas multiplataforma. Essa restrição compromete a reprodutibilidade dos estudos e dificulta a construção de avaliações comparativas que abranjam diferentes cenários de desenvolvimento.

Diante disto, os LLMs surgem como uma alternativa notável para ampliar a cobertura e interpretar relações estruturais não capturadas por heurísticas rígidas. No entanto, sua adoção ainda carece de validação sistemática e de estudos controlados que comparem diretamente seu desempenho ao de ferramentas tradicionais, sobretudo em linguagens além de Java. Essa necessidade é especialmente crítica para os seis *CS* analisados nesta proposta (S29, S30, S34, S36, S37 e S39), cuja detecção depende de nuances semânticas e de relações entre classes frequentemente negligenciadas pelas ferramentas disponíveis.

Em síntese, apesar do potencial demonstrado pelos LLMs para reconhecer padrões semânticos e ampliar a cobertura da detecção automatizada, sua aplicação à identificação de *CS* ainda carece de validação rigorosa. Persistem lacunas fundamentais relacionadas à ausência de protocolos experimentais padronizados, à predominância de estudos centrados em Java e à heterogeneidade metodológica entre trabalhos recentes.

Portanto, torna-se necessária uma avaliação sistemática que permita mensurar, de forma comparável e reproduzível, o desempenho desses modelos em linguagens amplamente utilizadas na indústria. É exatamente esse espaço que a presente proposta ocupa, ao conduzir um estudo empírico, comparativo e orientado a projetos reais em TypeScript, com foco nos seis *CS* selecionados.

2.4 Considerações Finais do Capítulo

Encerrando este capítulo, observa-se que os conceitos aqui apresentados constituem a base indispensável para compreender os fenômenos investigados nesta proposta. A discussão sobre *CS* (Seção 2.1) evidenciou tanto sua relevância para a qualidade do software quanto as limitações das ferramentas tradicionais de detecção, especialmente em linguagens amplamente utilizadas como TypeScript. De modo complementar, a revisão das técnicas de refatoração (Seção 2.2) reforçou sua responsabilidade na mitigação desses problemas estruturais e na garantia da manutenibilidade ao longo do ciclo de vida dos sistemas.

No que diz respeito aos LLMs, a discussão apresentada na Seção 2.3 mostrou como suas arquiteturas, formas de acesso, tipos de modelos, bem como as plataformas que viabilizam sua execução, compõem um ecossistema robusto que amplia significativamente as possibilidades de aplicação de IA na ES. Esses elementos revelam capacidades como (i) autonomia tecnológica; (ii) controle de dados; (iii) escalabilidade; (iv) multimodalidade; e (v) customização. Isto torna os LLMs particularmente promissores para tarefas complexas de ES, incluindo a análise semântica de código, o apoio à refatoração e a detecção de *CS*.

Ao mesmo tempo, ficou evidente que LLMs, de forma geral, ainda enfrentam limitações relevantes (*e.g.* elevada sensibilidade ao contexto das instruções). Esses fatores mostram que, embora os LLMs tenham potencial para transformar práticas de ES, sua adoção efetiva ainda requer investigações sistemáticas e comparativas que permitam validar, de forma consistente, o alcance e a confiabilidade de suas contribuições.

Com isso, o presente capítulo consolida o arcabouço teórico que sustenta esta proposta, estabelecendo a base conceitual necessária para orientar as análises, as decisões metodológicas e as interpretações dos resultados, que serão aprofundadas nos capítulos seguintes.

Capítulo 3

Trabalhos Relacionados

Neste capítulo, são apresentados os trabalhos relacionados a esta tese, com o propósito de contextualizar os progressos recentes observados na literatura sobre o uso de LLMs na detecção e na refatoração de *CS* na área de ES. A seleção dos estudos examinados fundamenta-se no Estudo de Mapeamento Sistemático (EMS) descrito no Apêndice A, cujo protocolo metodológico permitiu identificar tendências emergentes e lacunas relevantes na literatura, assegurando que a revisão empreendida neste capítulo esteja apoiada em um levantamento sistemático, abrangente e reprodutível.

Os estudos analisados exploram diferentes vertentes da aplicação de LLMs, abrangendo a detecção estrutural de *CS*, a recomendação de refatorações, a integração com arquiteturas neurais híbridas, o uso de hipergrafos e de mecanismos de validação estática. Embora distintos em escopo e metodologia, esses estudos convergem ao evidenciar o potencial e os desafios do uso de modelos generativos na manutenção de software.

Cada trabalho é apresentado segundo uma estrutura uniforme composta por seis pontos principais: (i) introdução contextual; (ii) descrição técnica da proposta; (iii) metodologia adotada; (iv) resultados; (v) limitações; e (vi) relação com os objetivos desta tese, o que viabiliza a comparabilidade entre as análises.

A seguir, apresenta-se a Tabela 3.1, que sintetiza os 8 (oito) trabalhos relacionados, organizando-os por ID, título, autor e foco temático, oferecendo uma visão global das contribuições discutidas ao longo do capítulo.

Tabela 3.1: Roadmap dos trabalhos relacionados

ID	Título	Autores	Tema
3.1	*AI-Driven Refactoring: A Pipeline for Identifying and Correcting Data Clumps in Git Repositories*	Baumgartner *et al.* [12]	LLMs para refatoração de *Data Clumps*.
3.2	*Three Heads Are Better Than One: Suggesting Move Method Refactoring Opportunities with Inter-class Code Entity Dependency Enhanced Hybrid Hypergraph Neural Network*	Cui *et al.* [17]	LLMs para refatoração *Move Method*.
3.3	*One-to-One or One-to-Many? Suggesting Extract Class Refactoring Opportunities with Intra-class Dependency Hypergraph Neural Network*	Cui *et al.* [18]	LLMs e hipergrafos neurais para refatoração *Extract Class*.
3.4	*Unearthing Gas-Wasting Code Smells in Smart Contracts With Large Language Models*	Jiang *et al.* [41]	LLMs para detecção e refatoração de *GWCS*.
3.5	*Next-Generation Refactoring: Combining LLM Insights and IDE Capabilities for Extract Method*	Pomian *et al.* [62]	LLMs, *Reprompting* e *Ranking* para Refatoração *Extract Method*.
3.6	*EM-Assist: Safe Automated Extract Method Refactoring with LLMs*	Pomian *et al.* [63]	LLMs na refatoração *Extract Method*.
3.7	*iSMELL: Assembling LLMs with Expert Toolsets for Code Smell Detection and Refactoring*	Wu *et al.* [81]	LLMs com Arquitetura *MoE* para detecção e refatoração de *CS*.
3.8	*Move Method Refactoring Recommendation Based on Deep Learning and LLM-Generated Information*	Zhang *et al.* [84]	Aprendizado profundo híbrido e LLM para refatoração *Move Method*.

3.1 Trabalho de Baumgartner *et al.* (2024)

Entre os estudos recentes que exploram a integração de LLMs na Engenharia de Software, destaca-se a proposta de Baumgartner *et al.* (2024) [12], que apresenta uma *pipeline* modular para a detecção e a refatoração automatizada de *Data Clumps*, um tipo recorrente de CS caracterizado pela associação repetida de grupos de variáveis no código-fonte. A metodologia compreende etapas sucessivas de extração, análise e refatoração assistida, estruturadas para integrar algoritmos determinísticos e intervenções por meio de LLMs.

O processo inicia-se pela análise do código-fonte, que pode ser convertido em *Abstract Syntax Trees* (ASTs) a fim de reduzir o volume de dados processados e contornar as restrições da janela de contexto dos modelos de linguagem, como o GPT-4 Turbo [57]. Em seguida, o LLM é empregado para sugerir nomes de novas classes e efetuar transformações no código, utilizando técnicas de engenharia de *prompt* para maximizar a qualidade das respostas. A abordagem oferece três modos de automação: (i) detecção, (ii) refatoração assistida e (iii) combinação de ambas, permitindo diferentes graus de intervenção humana no processo.

A avaliação experimental abrangeu dois cenários distintos. O primeiro, conduzido em um projeto reduzido (cerca de 90 linhas de código), demonstrou a eficácia do *pipeline* em contextos controlados e evidenciou ganhos de sensibilidade ao utilizar ASTs em comparação com o código completo. O segundo experimento, aplicado a um projeto real de grande porte (aproximadamente 180 mil linhas), revelou limitações de escalabilidade, incluindo a geração de código não compilável e a extração incompleta de classes, o que exigiu supervisão humana para a validação e a correção das refatorações.

Esses resultados ressaltam desafios técnicos relacionados à consistência sintática e à limitação de contexto dos LLMs, bem como a importância de integrar mecanismos determinísticos de validação e verificação contínua do código refatorado. Apesar dos avanços obtidos, o estudo concentrou-se exclusivamente na refatoração de *Data Clumps*, não abrangendo outros tipos de CS nem comparações diretas com ferramentas tradicionais de análise estática, o que limita a generalização dos achados.

Essas restrições reforçam a necessidade de abordagens mais amplas e comparativas, capazes de avaliar a aplicabilidade dos LLMs em diferentes contextos e linguagens, especialmente por meio da análise de múltiplos CS e da comparação sistemática com ferramentas

clássicas de detecção, aspectos que orientam a presente tese.

3.2 Trabalho de Cui *et al.* (2024a)

Com a crescente adoção de LLMs em tarefas de manutenção e refatoração de código, o trabalho de Cui *et al.* (2024) [17] apresenta uma solução para o problema clássico de má alocação de métodos em sistemas orientados a objetos, associado ao *CS* conhecido como *Feature Envy*. Nesse tipo de ocorrência, um método passa a depender excessivamente de dados de outras classes, em vez de operar principalmente com os atributos da própria classe.

Esse tipo de anomalia estrutural compromete a coesão interna, dificulta a manutenção e aumenta a dívida técnica do sistema. Para mitigar esses efeitos, os autores propuseram o modelo HMove, baseado em aprendizado profundo voltado à recomendação de refatorações do tipo *Move Method*, que combina redes neurais baseadas em hipergrafos de dependência interclasse e modelos de código pré-treinados, capazes de capturar dependências complexas entre múltiplas entidades distribuídas em diferentes classes.

A metodologia adotada compreendeu duas etapas principais: uma fase de treinamento, em que hipergrafos interclasse foram construídos a partir do código-fonte e enriquecidos com atributos semânticos extraídos de modelos pré-treinados de código, e uma fase de inferência, na qual o modelo treinado sugere oportunidades de refatoração *Move Method*, sendo o LLM empregado em uma etapa posterior de filtragem e validação das recomendações, propiciando maior precisão e consistência dos resultados.

O estudo testou sistematicamente 6 modelos de código pré-treinados, 3 arquiteturas de grafos neurais e 3 variantes de redes neurais baseadas em hipergrafos, evidenciando a robustez do processo experimental e a diversidade das combinações avaliadas. Entre as configurações testadas, a combinação `GraphSAGE + HGNN + CodeT5` apresentou o melhor desempenho global. O HMove superou nove ferramentas de referência (incluindo *RMove* e *FeTruth*), apresentando ganhos expressivos de 27,8% na precisão, 2,5% no *recall* e 18,5% na métrica F1, além de maior utilidade percebida por 68% dos participantes.

Os resultados demonstraram que o modelo HMove é capaz de identificar dependências interclasse com maior precisão e sugerir realocações de métodos mais alinhadas à estrutura lógica dos sistemas avaliados. Um estudo de usabilidade adicional, realizado com desenvol-

vedores da indústria, indicou que a maioria considerou as sugestões do modelo mais úteis e coerentes do que as de ferramentas tradicionais, evidenciando ganhos em explicabilidade e relevância prática.

Apesar dos resultados promissores, o estudo apresenta algumas limitações. A integração com LLMs restringe-se à etapa final de filtragem e validação, sem uma exploração aprofundada do seu impacto na generalização dos resultados. Além disso, a avaliação concentrou-se exclusivamente na refatoração *Move Method*, não contemplando outros tipos de CS nem cenários de larga escala e heterogêneos. Essas lacunas apontam oportunidades relevantes para investigações futuras, especialmente quanto à ampliação da cobertura de refatorações e ao fortalecimento de métricas comparativas.

O estudo de Cui *et al.* (2024) [17] representa um marco na aplicação de hipergrafos neurais híbridos à recomendação de refatorações assistidas por LLMs, constituindo uma das bases conceituais mais avançadas na literatura recente sobre automação de manutenção de software. A presente tese se apoia em princípios semelhantes de integração entre aprendizado profundo e modelos de linguagem, expandindo-os para outros contextos e tipos de CS, além de explorar linguagens dinamicamente tipadas, como o TypeScript, com o objetivo de avaliar a capacidade de generalização e as fragilidades dessas abordagens no cenário de detecção e refatoração automatizada.

3.3 Trabalho de Cui *et al.* (2024b)

Entre as contribuições mais recentes que exploram a aplicação combinada de hipergrafos neurais e LLMs na recomendação de refatorações, destaca-se o estudo de Cui *et al.* (2024) [18], que representa uma evolução direta da abordagem proposta no trabalho anterior, o HMove [17].

Enquanto o HMove trata da refatoração *Move Method*, capturando dependências estruturais entre classes distintas (interclasse), o novo modelo — denominado HECS (*Hypergraph-based Extract Class Suggestion*) [18] — estende essa linha de pesquisa ao domínio intraclasse, abordando o problema de classes extensas e multifuncionais, cuja divisão em partes menores e coesas pode ser realizada por meio da refatoração *Extract Class*.

No cerne da proposta, o HECS utiliza hipergrafos de dependência intraclasse para mode-

lar relações complexas do tipo “um-para-muitos”, garantindo a representação simultânea de múltiplos métodos e campos relacionados. Esse tipo de modelagem supera a limitação das abordagens tradicionais baseadas apenas em grafos, que capturam primordialmente dependências diretas "um-para-um"entre pares de entidades.

A metodologia segue uma estrutura semelhante à do HMove: construção de hipergrafos intraclasse, atribuição de características semânticas aos nós por meio de modelos de código pré-treinados, treinamento de uma rede neural de hipergrafos com atributos enriquecidos e, por fim, refinamento das sugestões com o apoio de um LLM para maior precisão.

A avaliação experimental foi conduzida em larga escala e indicou ganhos substanciais em relação às ferramentas do estado da arte. O HECS apresentou melhorias de 38,5% na precisão, 9,7% no *recall* e 44,4% na métrica F1, quando comparado a soluções consolidadas como *JDeodorant*, *SSECS* e *LLMRefactor*. Além disso, 64% dos participantes consideraram as recomendações mais úteis e alinhadas às decisões humanas reais de refatoração, reforçando a capacidade do modelo de compreender dependências semânticas avançadas e de propor sugestões contextualizadas.

A continuidade autoral e a progressão conceitual do modelo interclasse (HMove) para o intraclasse (HECS) evidenciam uma linha de pesquisa coesa, sustentada pelo uso de hipergrafos neurais e LLMs para identificar oportunidades de refatoração em múltiplos níveis de granularidade. Essa abordagem híbrida demonstra forte convergência com os objetivos desta tese, que investiga a aplicabilidade e a eficácia desses modelos em linguagens dinamicamente tipadas, como TypeScript, ampliando o potencial de generalização dos avanços metodológicos observados.

Por fim, o HECS consolida uma extensão natural do HMove, adaptando com sucesso a arquitetura híbrida de hipergrafos neurais e LLMs ao domínio da refatoração *Extract Class*. A modelagem "um-para-muitos"introduzida no estudo amplia a capacidade de análise estrutural e semântica do código, fornecendo uma base sólida para pesquisas futuras voltadas à detecção automatizada de *CS* e à recomendação de refatorações orientadas por aprendizado profundo.

3.4 Trabalho de Jiang *et al.* (2024)

No contexto das aplicações baseadas em *blockchain*, contratos inteligentes escritos em Solidity [71] executam operações cujo custo é medido em "gás", unidade que representa o esforço computacional requerido pela *Ethereum Virtual Machine* (EVM) [29]. Como esse custo afeta diretamente a viabilidade econômica de sistemas descentralizados, padrões de implementação ineficientes podem ampliar significativamente o consumo de recursos e, por essa razão, passam a ser denominados "*Gas-Wasting Code Smells*"(GWCS). Diante desse cenário, Jiang *et al.* (2024) [41] investigaram como LLMs podem apoiar a identificação de *GWCS*, propondo uma abordagem voltada à detecção e à refatoração de padrões que elevam o consumo de gás em contratos inteligentes.

Para enfrentar esse desafio, os autores propuseram um *pipeline* semiautomatizado composto por três etapas fundamentais: (i) levantamento preliminar dos potenciais *GWCS* a partir de revisão sistemática e análise de contratos reais; (ii) interação iterativa com o GPT-4 [57], utilizando *prompts* elaborados com raciocínio explicativo e mecanismo de autoinflexão para detecção e classificação dos *GWCS*; e (iii) validação manual conduzida por dois especialistas com experiência em desenvolvimento *blockchain*, assegurando a precisão dos diagnósticos.

O conjunto inicial continha mais de três mil contratos coletados por meio de *crawling* de projetos recentemente verificados. Após filtros que garantiram diversidade e complexidade, tais como (i) remoção de funções duplicadas; (ii) descarte de funções com menos de 5 linhas; e (iii) seleção do topo de 10% das funções mais extensas, o conjunto final foi reduzido para 311 contratos. Esses trechos foram enviados ao GPT-4 em formato de código textual, sem transformação em AST (*Abstract Syntax Tree*) ou *bytecode*, devido ao uso da janela de contexto padrão do modelo.

Os resultados revelaram 26 categorias distintas de *GWCS*, das quais 13 eram inéditas na literatura. Exemplos dentre os 13 novos *smells* identificados são: (i) *Repeated Computation of the Same Expression*; (ii) *Extractable Code Chunks*; (iii) *State Variable Refactoring*; e (iv) *Pre-Computable Operations on Constants*. Cada uma dessas categorias emergiu da colaboração entre GPT-4 e avaliadores humanos, que confirmaram sua validade conceitual e relevância prática.

A aplicação das refatorações sugeridas resultou em ganhos significativos: economia mé-

dia de 10,53% no custo de *deployment* dos contratos e de 21,53% nas chamadas de mensagem. Esses resultados demonstram o potencial dos LLMs como ferramentas de apoio à auditoria de contratos inteligentes, especialmente em tarefas de otimização de gás.

Do ponto de vista qualitativo, os autores observaram que, embora GPT-4 tenha desempenhado o papel de identificar novos *GWCS*, também apresentou restrições importantes. Alucinações, interpretações equivocadas de trechos complexos e dificuldade em lidar com funções extensas devido ao limite de contexto foram algumas dessas fragilidades. Por esse motivo, a etapa de validação humana foi essencial para mitigar falsos positivos e garantir a robustez das categorias propostas.

O estudo também discutiu fragilidades relacionadas à escalabilidade e generalização. A necessidade de validação manual reduz o grau de automação do processo. O trabalho também não realizou experimentos com variantes da EVM nem com outros LLMs além do GPT-4. Além disso, não foram conduzidos testes em ambientes reais de produção, o que restringe a extrapolação dos resultados. Esses fatores indicam oportunidades futuras para reduzir dependências humanas, ampliar a cobertura de linguagens e explorar comparações mais abrangentes entre modelos e arquiteturas.

A proposta de Jiang *et al.* (2024) converge com os objetivos desta tese ao demonstrar que LLMs podem identificar padrões estruturais complexos mesmo em domínios altamente especializados, reforçando a responsabilidade dessas tecnologias como apoio complementar (e não substitutivo) à análise estática. Assim, o estudo contribui para a compreensão de como modelos generativos podem auxiliar na detecção e na refatoração de *CS* em diferentes contextos, incluindo linguagens dinamicamente tipadas, como TypeScript, foco desta pesquisa.

3.5 Trabalho de Pomian *et al.* (2024b)

Em continuidade aos avanços obtidos na primeira versão do *EM-Assist* [63], Pomian *et al.* (2024) [62] apresentaram uma evolução substancial da ferramenta, propondo uma arquitetura mais abrangente para a automação da refatoração *Extract Method*. Essa nova etapa do projeto aprofunda a integração entre LLMs e capacidades analíticas provenientes da IDE, consolidando uma abordagem conceitualmente caracterizada como “refatoração de próxima

geração".

A versão expandida do *EM-Assist* manteve o uso do modelo GPT-3.5-turbo [57] como base experimental, mencionando o GPT-4 [57] apenas como uma perspectiva de avanços futuros. Sua arquitetura passou a incorporar mecanismos como *reprompting*, ranqueamento e filtros automáticos para aprimorar a qualidade das sugestões.

O *reprompting* consiste em realizar múltiplas requisições sucessivas ao LLM, mitigando o não determinismo inerente aos modelos e gerando um conjunto mais rico e diverso de respostas. A partir desse superconjunto, o sistema aplica filtros estruturais e semânticos, ranqueando as recomendações mais promissoras antes de submetê-las à validação humana.

Do ponto de vista metodológico, o estudo incluiu avaliações quantitativas conduzidas em dois conjuntos de dados (*Community Corpus* e *Extended Corpus*), complementadas por um estudo empírico com desenvolvedores da JetBrains, realizado por meio de *firehouse surveys*. Os resultados indicaram que 81,3% dos participantes consideraram as recomendações úteis, especialmente em cenários complexos que envolvem métodos extensos, fortemente acoplados e compostos por blocos não contíguos. Esses achados evidenciam a eficácia dos mecanismos iterativos introduzidos e reforçam a função complementar dos LLMs no processo de refatoração.

Apesar dos avanços obtidos, o trabalho apresenta fragilidades que permanecem consistentes com a versão anterior [63]. O sistema continua restrito à refatoração *Extract Method* e à aplicação em linguagens estáticas como Java e Kotlin, além de exigir intervenção humana para a validação final das sugestões. Essas restrições indicam a necessidade de expandir a investigação para outros tipos de refatoração e para linguagens com características distintas, bem como de aperfeiçoar a autonomia e a robustez do processo iterativo.

Os resultados apresentados por Pomian *et al.* (2024) [62] fornecem subsídios relevantes para esta tese ao demonstrar como estratégias híbridas baseadas em *reprompting*, ranqueamento e filtros adaptativos podem aprimorar a precisão e a adequação contextual das recomendações de refatoração. Esses achados reforçam a importância de investigar a generalização dessas abordagens em linguagens dinamicamente tipadas, como TypeScript, e de comparar sistematicamente seu desempenho com ferramentas tradicionais de análise estática.

3.6 Trabalho de Pomian *et al.* (2024a)

No contexto do avanço das técnicas híbridas que combinam LLMs e mecanismos de análise estática, destaca-se o estudo de Pomian *et al.* (2024) [63], que apresentou o EM-Assist, um *plugin* para a IDE IntelliJ IDEA [40] projetado para automatizar de forma segura a refatoração *Extract Method*. Esse trabalho representa um esforço sistemático dos autores para avaliar empiricamente a viabilidade, a precisão e a utilidade prática do uso de LLMs na geração de recomendações de refatoração.

O estudo parte da constatação de que ferramentas tradicionais, baseadas em regras estáticas e métricas convencionais, embora tecnicamente corretas, não capturam adequadamente as intenções subjetivas dos desenvolvedores, o que reduz sua adoção prática. Para lidar com essa restrição, o EM-Assist combina o raciocínio contextual de LLMs com verificações estruturais derivadas da IDE, removendo "alucinações" das respostas geradas pelo modelo e validando automaticamente as recomendações antes de apresentá-las ao usuário.

Do ponto de vista metodológico, a abordagem foi conduzida em múltiplas etapas: (i) construção de um oráculo empírico composto por 1.752 casos reais de refatoração *Extract Method*, extraídos do histórico de versões de projetos *open source* por meio da ferramenta *RefactoringMiner*; (ii) execução do EM-Assist e de seis ferramentas concorrentes, dentre elas *JExtract*, *IntelliJ Extract Method* e *Eclipse Extract Method*, com o objetivo de comparar precisão e *recall* em relação ao oráculo; e (iii) realização de um estudo qualitativo com 18 desenvolvedores profissionais, destinado a avaliar a usabilidade, clareza e adequação prática das recomendações geradas.

Os resultados demonstraram que o EM-Assist superou as ferramentas concorrentes de forma significativa, alcançando 53,4% de *recall* entre suas cinco melhores recomendações, em contraste com os 39,4% obtidos pelo *JExtract*. Além disso, 94,4% dos participantes do estudo qualitativo avaliaram positivamente a ferramenta, evidenciando seu potencial para conciliar automação e alinhamento às práticas reais de desenvolvimento.

Apesar dos ganhos observados, o estudo também identificou fragilidades importantes. O desempenho do EM-Assist permanece dependente de características específicas da linguagem (Java/Kotlin) e do ecossistema da IDE, além de incorporar um componente de não determinismo inerente às respostas dos LLMs. Ademais, sua avaliação concentrou-se exclu-

sivamente na refatoração *Extract Method*, não abrangendo outros tipos de refatoração nem cenários de desenvolvimento mais amplos e heterogêneos.

Os resultados apresentados por Pomian *et al.* (2024) [63] fornecem subsídios relevantes para esta tese, ao demonstrar o potencial dos LLMs na recomendação de refatorações automatizadas e ao evidenciar restrições estruturais decorrentes da dependência de linguagens específicas e de ambientes integrados de desenvolvimento. Tais evidências reforçam a necessidade de estudos que avaliem a capacidade de generalização dessas abordagens em diferentes linguagens, como TypeScript, bem como sua eficácia comparativa frente a ferramentas tradicionais de análise estática.

3.7 Trabalho de Wu *et al.* (2024)

No âmbito da automação da qualidade de software, Wu *et al.* (2024) [81] apresentaram uma abordagem híbrida destinada a superar restrições presentes tanto em heurísticas clássicas quanto em *pipelines* baseados exclusivamente em modelos generativos denominada iSMELL. Os autores partem do reconhecimento de que soluções tradicionais carecem de sensibilidade semântica e apresentam baixa cobertura de diferentes tipos de *CS*. Por outro lado, embora expressivos, os LLMs ainda sofrem com instabilidade nas respostas, inconsistências estruturais e dificuldades na refatoração complexa de forma confiável.

Para mitigar essas restrições, os autores desenvolveram uma arquitetura baseada na estratégia *Mixture of Experts* (*MoE*), na qual diversos *toolsets* especializados atuam como “especialistas” responsáveis pela detecção de tipos específicos de *CS*. Os "*experts*"englobam três *CS* principais: (i) *Feature Envy*; (ii) *God Class*; e (iii) *Refused Bequest*.

Cada módulo opera de forma independente, emitindo diagnósticos específicos, enquanto o LLM atua como "orquestrador", integrando e harmonizando as saídas do *MoE* para propor transformações estruturais e ajustar o código resultante, proporcionando a coerência sintática e semântica. Essa integração modular permite combinar a precisão de detectores especializados com a flexibilidade dos modelos generativos.

A avaliação experimental do *iSMELL* considerou tanto métricas quantitativas quanto uma análise qualitativa conduzida por três avaliadores experientes em Java, que atribuíram notas de 0 a 4 a cada refatoração proposta. Os resultados quantitativos indicaram que o *iSMELL*

superou ferramentas tradicionais e abordagens baseadas exclusivamente em LLMs, especialmente no *Feature Envy*.

Na avaliação qualitativa, os desempenhos observados foram os seguintes:

- **Feature Envy**: média de 2,60 – o melhor desempenho entre os três smells. Os avaliadores relataram que as sugestões produzidas pelo sistema foram, em sua maioria, apropriadas e eficazes.
- **Refused Bequest**: média de 2,36 – desempenho moderado, porém comprometido por um número maior de falsos positivos, o que reflete maior incidência de notas baixas.
- **God Class**: média de 2,08 – o desempenho menos expressivo. O sistema enfrentou dificuldades significativas em classes extensas e altamente complexas, exigindo múltiplas iterações e, por vezes, resultando em código parcial ou semanticamente inconsistente.

No caso específico de *God Class*, os autores destacaram que a limitação da janela de contexto dos LLMs agravou o problema, uma vez que classes muito extensas — como o exemplo apresentado no estudo, com 1.992 linhas — excederam a capacidade de análise completa do modelo, resultando em refatorações incompletas ou inconsistentes. Os avaliadores também relataram que, em tais casos, a preservação da funcionalidade exigiu uma validação humana mais intensiva, reforçando a complexidade inerente a esse tipo de *CS*.

De forma geral, os resultados evidenciaram que o *iSMELL* apresenta avanços significativos na integração entre especialistas heurísticos e LLMs, especialmente no tratamento de *CS* de granularidade intermediária e de dependência semântica acentuada. No entanto, também revelaram que o sistema enfrenta limitações quando múltiplas refatorações dependem umas das outras, o que resulta em artefatos incompletos ou incoerentes. Tais restrições foram atribuídas, entre outros fatores, à incapacidade dos LLMs de analisar comportamento em tempo de execução e à dificuldade de manter consistência global em modificações que exigem coordenação entre diferentes especialistas.

Como direções futuras, Wu *et al.* [81] propuseram ampliar o repertório de especialistas do *MoE*, incorporar um repositório de exemplos de refatorações bem-sucedidas e estender a arquitetura para tarefas mais complexas, como a revisão automatizada de código e a mitigação de vulnerabilidades. Essa visão modular e cooperativa evidencia o potencial de

abordagens híbridas nas quais os LLMs atuam de forma complementar, e não substitutiva, aos mecanismos tradicionais de análise estática. Tal perspectiva dialoga diretamente com os objetivos desta tese ao explorar como LLMs podem apoiar a detecção e a refatoração de *CS* em projetos em TypeScript.

3.8 Trabalho de Zhang *et al.* (2025)

Entre as propostas mais recentes que combinam modelos semânticos e aprendizado profundo para apoiar a refatoração automatizada, destaca-se o trabalho de Zhang *et al.* (2025) [84], que introduziu o *MoveRec*, um método voltado à recomendação precisa da refatoração *Move Method*. A proposta aborda o desafio de identificar, de forma confiável, métodos que deveriam ser realocados para outras classes, considerando não apenas características estruturais, mas também informações textuais que refletem a intenção de design e o contexto semântico, aspectos difíceis de capturar apenas com métricas estáticas.

Para lidar com essa complexidade, o *MoveRec* adota uma arquitetura híbrida que integra informações estruturais e semânticas em um único vetor de características. As descrições textuais de métodos e classes são produzidas por LLMs (como o ChatGPT-3.5), sintetizando elementos semânticos relevantes para a compreensão das dependências internas do código.

Em paralelo, métricas estáticas extraídas de 58 projetos de código aberto em Java complementam a representação utilizada. Essas informações estruturais e textuais são processadas por uma arquitetura de rede neural composta por três ramos especializados: (i) uma CNN (Convolutional Neural Network) para capturar padrões estruturais das métricas estáticas; (ii) um LSTM (Long Short-Term Memory) e (iii) um GRU (Gated Recurrent Unit), ambos dedicados à modelagem das dependências temporais e semânticas presentes nas descrições textuais geradas por LLMs. As saídas desses três ramos são integradas para formar um modelo unificado de recomendação, consolidando padrões espaciais, temporais e semânticos e aprimorando a identificação de oportunidades de refatoração do tipo *Move Method*.

Os experimentos demonstraram que a incorporação de representações textuais geradas por LLMs elevou substancialmente o desempenho do sistema. Em particular, o modelo obteve ganhos de até 15% na métrica F1 em comparação a abordagens baseadas exclusivamente em métricas estáticas. Entre os modelos pré-treinados avaliados, o *RoBERTa* destacou-se por

capturar padrões semânticos mais refinados, aprimorando a capacidade do sistema de lidar com métodos conceitualmente ambíguos ou com dependências pouco explícitas.

Apesar dos resultados promissores, Zhang *et al.* (2025) [84] reconheceram limitações importantes: (i) redução de desempenho em cenários que demandam elevado *recall*; (ii) risco de dependência excessiva das descrições geradas pelos LLMs, que podem conter abstrações incorretas ou generalizações inadequadas; e (iii) ausência de experimentos envolvendo outras linguagens além de Java, bem como a falta de avaliação sistemática com diferentes LLMs. Tais restrições apontam oportunidades para o desenvolvimento de arquiteturas híbridas que integrem informações semânticas consistentes a modelos avançados de aprendizado profundo.

A proposta de Zhang *et al.* (2025) [84] converge com os objetivos desta tese ao demonstrar o potencial das descrições semânticas geradas por LLMs para aprimorar as recomendações de refatoração. Ao destacar benefícios, desafios e fragilidades do uso combinado de LLMs e modelos estruturais, o estudo fornece subsídios valiosos para compreender como abordagens semelhantes podem ser aplicadas à detecção e refatoração de *CS* em linguagens dinamicamente tipadas, como TypeScript — tema da presente pesquisa.

3.9 Considerações Finais do Capítulo

A análise dos trabalhos relacionados apresentados ao longo deste capítulo evidencia o rápido avanço das pesquisas que investigam o uso de LLMs na detecção e na refatoração de *CS* na ES. Embora tais contribuições tenham ampliado significativamente o entendimento sobre a atribuição de modelos generativos na manutenção de software, também revelam restrições que delineiam o estado atual da arte.

De modo geral, os estudos revisados concentram-se predominantemente em linguagens estaticamente tipadas, como Java, e abordam refatorações específicas – a exemplo de *Move Method*, *Extract Method* e *Data Clumps* – por meio de arquiteturas especializadas, incluindo hipergrafos neurais [18], reprompting e ranqueamento [62] e estratégias baseadas em MoE [81]. Em grande parte dessas abordagens, os LLMs desempenham funções complementares, atuando como geradores de informações semânticas, validadores de sugestões ou mecanismos auxiliares às técnicas tradicionais de análise estática.

Em contraste, esta tese avança em uma direção distinta ao investigar a capacidade dos LLMs de atuarem como detectores diretos de múltiplos *CS* em projetos em TypeScript, uma linguagem amplamente utilizada no ecossistema de desenvolvimento web, mas ainda pouco explorada no contexto da detecção automatizada de anomalias estruturais. Ao posicionar os modelos generativos não apenas como elementos auxiliares, mas também como agentes centrais de análise estrutural e semântica, a pesquisa amplia o escopo investigativo e oferece novas perspectivas sobre a aplicabilidade dos LLMs em linguagens dinamicamente tipadas.

Outro diferencial relevante decorre da estrutura metodológica adotada. Enquanto os trabalhos relacionados tendem a avaliar técnicas específicas ou mecanismos pontuais de recomendação, a investigação conduzida nesta tese organiza-se em cinco etapas: (i) revisão de ferramentas tradicionais; (ii) revisão de LLMs; (iii) definição sistemática do escopo conceitual; (iv) construção de *prompts* especializados; e (v) experimentação empírica em projetos reais. Essa abordagem integrada possibilita uma comparação direta, rigorosa e controlada entre ferramentas tradicionais e LLMs, fornecendo evidências mais abrangentes sobre suas capacidades, limitações e contextos de aplicabilidade.

Em síntese, a comparação crítica apresentada neste capítulo demonstra que, apesar dos avanços recentes em arquiteturas híbridas e mecanismos sofisticados de recomendação, persiste uma lacuna significativa na generalização dessas abordagens para linguagens modernas e dinamicamente tipadas, como TypeScript. A pesquisa desenvolvida nesta tese busca preencher essa lacuna ao oferecer uma análise sistemática e comparativa do desempenho dos LLMs na detecção e na refatoração de *CS*, contribuindo para o avanço de métodos mais eficazes, generalizáveis e alinhados às demandas contemporâneas da ES.

Capítulo 4

Metodologia

A metodologia adotada nesta pesquisa foi concebida para sustentar, de forma sistemática, rigorosa e reprodutível, a proposta de tese apresentada no Capítulo 1. Diferentemente de abordagens centradas exclusivamente na comparação entre técnicas, o presente estudo estrutura-se a partir da construção de um *benchmark* como artefato científico central, destinado a apoiar a investigação, a avaliação e a compreensão das limitações e potencialidades de diferentes abordagens para a detecção de *CS* em projetos desenvolvidos em TypeScript.

Nesse contexto, a metodologia não se restringe à execução de experimentos comparativos, mas organiza-se como um percurso investigativo orientado à sistematização do conhecimento, à produção de evidências empíricas reprodutíveis e à proposição de diretrizes fundamentadas para o uso integrado de ferramentas tradicionais e de LLMs. Assim, a comparação entre abordagens é tratada como um meio para a geração de evidências, e não como o objetivo final da pesquisa.

A estrutura metodológica foi delineada de forma a atender a três propósitos centrais: (i) estabelecer um escopo conceitual e operacional claro para a identificação de *CS* em TypeScript, fundamentado na taxonomia de Fowler [32]; (ii) viabilizar a construção de um *benchmark* reprodutível, composto por dados rotulados, protocolos experimentais e instrumentos de análise; e (iii) permitir a investigação sistemática das condições sob as quais abordagens baseadas em LLMs podem complementar ou ampliar técnicas tradicionais de análise estática.

Para atingir esses objetivos, a metodologia integra procedimentos teóricos, experimentais e empíricos, organizados de forma progressiva e interdependente. Esse encadeamento asse-

gura a coerência entre os objetivos da proposta (Subseções 1.2.1 e 1.2.2), as QPs formuladas (Subseção 1.2.3) e os resultados preliminares (Capítulo 5), além de favorecer a rastreabilidade das decisões metodológicas adotadas ao longo do estudo.

O percurso metodológico está estruturado em três grandes fases, que se desdobram em etapas específicas. Essas fases foram concebidas para refletir o ciclo completo de construção do *benchmark*, sua aplicação experimental e a consolidação dos resultados. Em síntese:

- Fase 1: Concentra-se na fundamentação conceitual e na definição do escopo do estudo, incluindo a análise das abordagens existentes, a seleção dos *CS* e a definição dos critérios operacionais;
- Fase 2: Dedica-se à operacionalização empírica da pesquisa, envolvendo a construção do conjunto de dados, a implementação dos protocolos experimentais e a execução controlada das abordagens baseadas em ferramentas tradicionais e LLMs;
- Fase 3: Contempla a consolidação dos resultados, a ampliação dos experimentos, a validação dos achados e a definição de diretrizes para a integração de LLMs em processos de detecção e refatoração de *CS*.

Esse arranjo metodológico permite que o *benchmark* desempenhe um papel central ao longo de toda a pesquisa, atuando simultaneamente como instrumento de experimentação, mecanismo de comparação e artefato científico reutilizável. Dessa forma, a metodologia adotada sustenta não apenas a análise empírica proposta, mas também a contribuição científica mais ampla da proposta da tese, alinhando-se ao objetivo de avançar o estado da arte na detecção de *CS* em TypeScript.

A Figura 4.1 apresenta uma visão geral da estrutura metodológica proposta, destacando as fases, suas inter-relações e o fluxo lógico que orienta a execução do estudo.

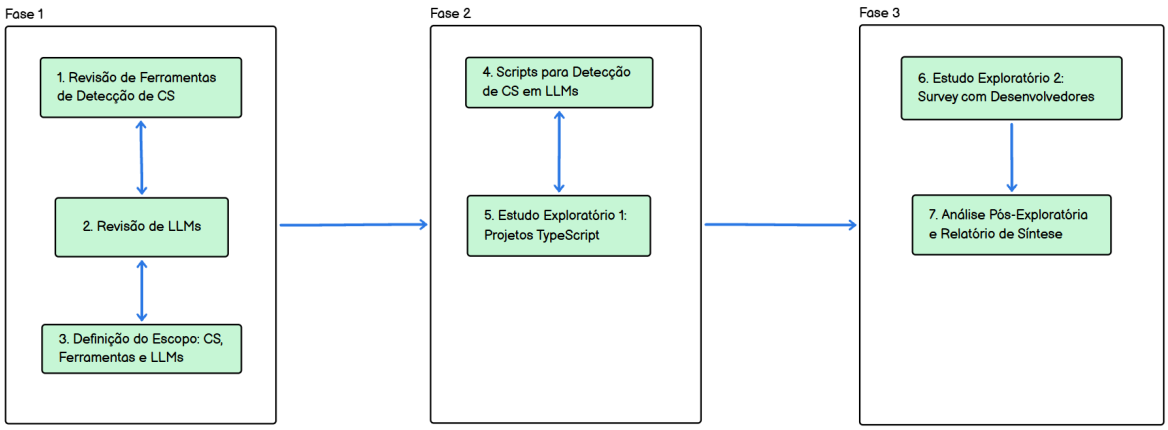

Figura 4.1: Metodologia

A seguir, apresenta-se a Seção 4.1, na qual cada uma das três fases metodológicas é descrita detalhadamente, destacando-se a constituição de suas etapas, bem como as entradas e saídas que interligam e orientam o fluxo da presente investigação. As Figuras 4.2, 4.3 e 4.4 ilustram o funcionamento interno de cada fase, permitindo visualizar, de forma lúdica e estruturada, a progressão lógica do estudo e o encadeamento entre suas partes.

4.1 Fases e Etapas

Com base na estrutura metodológica geral apresentada anteriormente, a pesquisa foi organizada em três fases, de modo a refletir o caráter progressivo, investigativo e reprodutível da proposta de tese. Essa organização tem como eixo central a construção e validação progressivas de um *benchmark* para a detecção de *CS* em projetos desenvolvidos em TypeScript.

Diferentemente de abordagens estritamente comparativas, o presente estudo estrutura suas etapas de modo a possibilitar a sistematização do conhecimento, a experimentação controlada e a consolidação de evidências empíricas ao longo de todo o processo investigativo, em consonância com os objetivos e as Questões de Pesquisa definidos no Capítulo 1.

Nesse contexto, a metodologia foi organizada em três fases complementares, representadas esquematicamente na Figura 4.1, as quais se articulam de maneira sequencial e interdependente:

- Fase 1 – Revisão e Planejamento (Subseção 4.1.1);

- Fase 2 – Preparação e Testes Iniciais (Subseção 4.1.2);
- Fase 3 – Expansão, Validação e Síntese (Subseção 4.1.3).

Essas fases refletem o ciclo completo de construção do *benchmark*, desde sua fundamentação conceitual até sua aplicação experimental e validação empírica, assegurando a coerência metodológica e a rastreabilidade entre as etapas do estudo.

4.1.1 Fase 1 – Revisão e Planejamento

A Fase 1, ilustrada na Figura 4.2, corresponde ao alicerce conceitual e metodológico da pesquisa. Seu objetivo principal foi estabelecer as bases teóricas, técnicas e operacionais que orientariam todas as etapas subsequentes, assegurando a coerência científica e o rigor metodológico à proposta.

Essa fase caracterizou-se por atividades de natureza exploratória e analítica, conduzidas parcialmente em paralelo, de modo a permitir a realimentação contínua dos resultados obtidos. O foco esteve na compreensão do estado da arte sobre detecção de *CS*, nas limitações das abordagens existentes e no potencial de uso de LLMs nesse contexto.

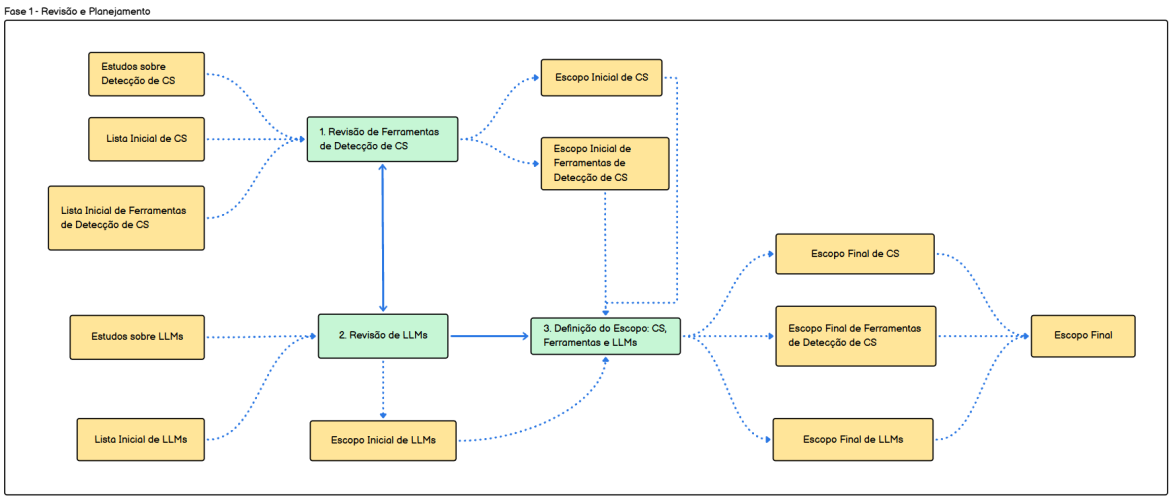

Figura 4.2: Metodologia – Fase 1

No âmbito desta pesquisa, o EMS descrito no Apêndice A constituiu o núcleo da Fase 1, fornecendo evidências empíricas que fundamentam a definição do escopo experimental, a seleção dos *CS* e a escolha das abordagens investigadas.

A Etapa 1 concentrou-se na revisão das ferramentas tradicionais de detecção de *CS*, abrangendo soluções amplamente utilizadas na indústria e na literatura científica, como SonarQube [72], ESLint [28], typescript-eslint [78] e Embold [25]. A análise dessas ferramentas permitiu identificar seus princípios de funcionamento, limitações estruturais e lacunas na interpretação semântica do código.

Como resultado, definiu-se o conjunto inicial de *CS* relevantes para o estudo (ver Subseção 2.1.1), bem como o conjunto de ferramentas que serviriam como referência experimental, estabelecendo uma linha de base para as etapas subsequentes.

A Etapa 2 dedicou-se à revisão dos LLMs disponíveis, considerando aspectos como a arquitetura, as capacidades de compreensão semântica, as limitações conhecidas e a adequação à análise de código-fonte. Essa etapa permitiu estabelecer critérios objetivos para a seleção dos modelos a serem investigados, bem como compreender seus potenciais e restrições no contexto da detecção de *CS*.

A Etapa 3 consolidou os resultados das etapas anteriores, culminando na definição final do escopo da pesquisa. Nessa etapa foram estabelecidos, de forma integrada: (i) os *CS* investigados; (ii) as ferramentas tradicionais consideradas; e (iii) os LLMs selecionados. Esse refinamento constitui o núcleo conceitual do *benchmark* proposto, atuando simultaneamente como recorte metodológico, base experimental e mecanismo de reprodutibilidade.

4.1.2 Fase 2 – Preparação e Testes Iniciais

A Fase 2, apresentada na Figura 4.3, correspondeu à operacionalização empírica da proposta, transformando o escopo definido na Fase 1 em instrumentos, procedimentos e experimentos concretos. Essa fase teve como objetivo avaliar, de forma controlada, a viabilidade da abordagem proposta e produzir os primeiros resultados empíricos sustentados pelo *benchmark*.

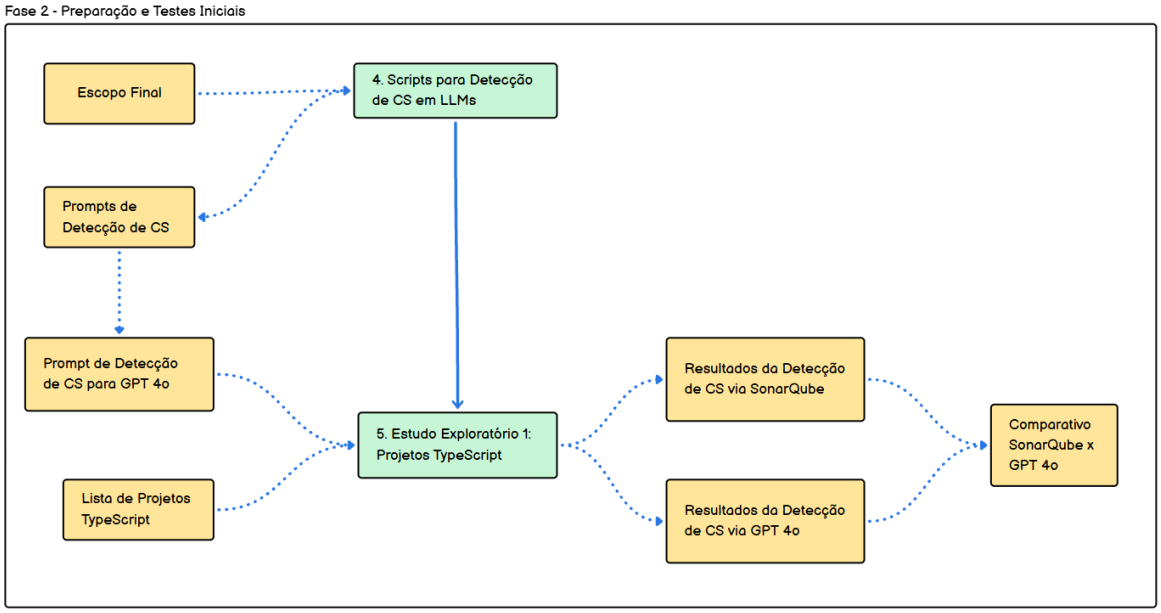

Figura 4.3: Metodologia – Fase 2

Na Etapa 4, foram desenvolvidos os *scripts* e *prompts* utilizados para a detecção de *CS* por meio de LLMs. Essa etapa envolveu a definição do formato de entrada, a padronização das instruções e o controle das variáveis experimentais, assegurando a consistência e a reprodutibilidade. Os artefatos gerados nesta etapa constituem componentes centrais do *benchmark* proposto.

A Etapa 5 consistiu na condução do Estudo Exploratório 1, realizado com base em projetos reais em TypeScript, obtidos do GitHub. Nessa etapa, foram aplicadas tanto as ferramentas tradicionais quanto os LLMs selecionados, com o objetivo de avaliar o funcionamento inicial do *benchmark*, observar padrões de comportamento das abordagens e identificar limitações preliminares na detecção de *CS*.

Os resultados obtidos permitiram verificar a viabilidade do protocolo experimental, analisar o comportamento das abordagens em cenários reais e gerar evidências iniciais que subsidiaram o refinamento metodológico, bem como a definição das análises aprofundadas previstas para a fase subsequente (Subseção 4.1.3).

4.1.3 Fase 3 – Expansão, Validação e Síntese

A Fase 3, ilustrada na Figura 4.4, representa a etapa de consolidação da pesquisa e tem como objetivo ampliar, validar e sintetizar os resultados obtidos anteriormente. Nessa fase, o *benchmark* é explorado de forma mais abrangente, com a inclusão de novos modelos, novas análises e validações empíricas adicionais.

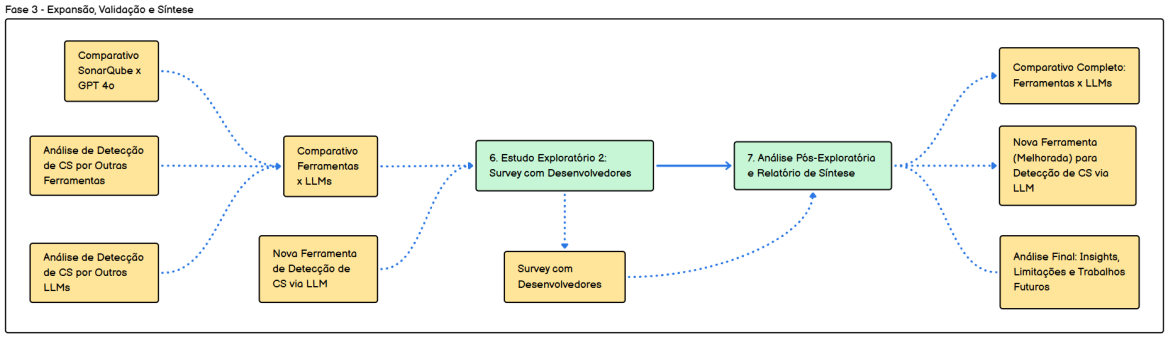

Figura 4.4: Metodologia – Fase 3

A Etapa 6 compreende a execução de experimentos ampliados, incluindo a avaliação de novos LLMs, a comparação com outras ferramentas e a realização de um *survey* junto a desenvolvedores. Os dados coletados são submetidos a um protocolo de rotulagem com dupla revisão independente, seguido de análise de concordância e arbitragem, assegurando o rigor metodológico e a validade interna.

A Etapa 7 finaliza o processo metodológico por meio da análise integrada dos resultados, da consolidação do *benchmark* e da síntese dos achados da pesquisa. Essa etapa resulta na produção dos principais artefatos científicos da tese: (i) o *benchmark* reprodutível; (ii) a análise comparativa consolidada; (iii) as diretrizes para integração de LLMs em processos de detecção de *CS*; e (iv) o relatório final de síntese, que fundamenta as conclusões e contribuições do trabalho.

4.2 Alinhamento entre a Metodologia e QPs

A metodologia delineada neste capítulo foi concebida para garantir correspondência direta entre as etapas executadas e as Questões de Pesquisa apresentadas no Subseção 1.2.3. Cada fase da metodologia contribui para responder, de forma sistemática e empiricamente funda-

mentada, às lacunas identificadas no Capítulo 1, assegurando rigor investigativo, rastreabilidade e coerência metodológica.

- **QP1**: É endereçada pela Etapa 1 da Fase 1 e pela Etapa 5 da Fase 2. Busca avaliar em que medida ferramentas tradicionais de análise estática são capazes de detectar *CS* dependentes de relações semânticas e estruturais em projetos em TypeScript.

  A revisão das ferramentas clássicas (Etapa 1) estabelece os fundamentos conceituais necessários para compreender seus mecanismos de detecção, enquanto o Estudo Exploratório 1 (Etapa 5) fornece evidências empíricas obtidas por meio da aplicação do SonarQube em projetos reais. Esses resultados compõem a linha de base necessária para a interpretação do desempenho dessas ferramentas à luz das limitações identificadas na literatura.

- **QP2**: É abordada progressivamente ao longo das Fases 1 e 2 e aprofundada na Fase 3. Investiga a eficácia de LLMs na detecção e no apoio à refatoração de *CS*

  A revisão dos modelos (Etapa 2) fundamenta a seleção dos LLMs analisados; a construção dos *scripts* e *prompts* (Etapa 4) operacionaliza sua aplicação; e o Estudo Exploratório 1 (Etapa 5) produz uma primeira análise comparativa com base nos resultados de detecção.

  Já na Fase 3, a Etapa 6 amplia essa investigação ao incorporar novos modelos e ferramentas, bem como a avaliação empírica das sugestões de refatoração por meio de um *survey* com desenvolvedores. Esse procedimento permite examinar não apenas a capacidade de detecção, mas também a utilidade prática das sugestões geradas pelos LLMs. O protocolo de rotulagem com dupla revisão independente reforça a confiabilidade dos dados analisados e contribui para a validade interna das evidências associadas à **QP2**.

- **QP3**: É abordada de forma integrada na Etapa 7. Busca identificar sob quais condições os LLMs podem superar ou complementar ferramentas tradicionais na detecção de *CS*.

  Nessa etapa, são sintetizados os resultados experimentais, as análises comparativas e os dados provenientes do *survey*, permitindo uma avaliação abrangente das condições de complementaridade, de limitação ou de superioridade das abordagens analisadas.

Dessa forma, a articulação entre as fases metodológicas assegura que cada QP seja tratada de maneira direta, estruturada e empiricamente sustentada, garantindo alinhamento entre os objetivos da proposta, os métodos empregados e os resultados esperados.

A Tabela 4.1 sintetiza a correspondência entre as QPs e as fases e etapas da metodologia, evidenciando a coerência interna do desenho metodológico adotado.

Tabela 4.1: Alinhamento entre QPs, Fases e Etapas da Metodologia

QP	**Fases Relacionadas**	**Etapas Correspondentes**
QP1	Fase 1 (4.1.1) Fase 2 (4.1.2)	*Etapa 1* – Revisão das ferramentas tradicionais de detecção de *CS*, com identificação de seus princípios, limitações e lacunas *Etapa 5* – Estudo Exploratório 1: aplicação das ferramentas tradicionais em projetos reais, fornecendo a linha de base empírica para análise do desempenho e das limitações observadas
QP2	Fase 1 (4.1.1) Fase 2 (4.1.2) Fase 3 (4.1.3)	*Etapa 2* – Revisão e seleção dos LLMs considerados *Etapa 4* – Construção dos *scripts* e *prompts* para aplicação dos LLMs *Etapa 5* – Avaliação inicial da detecção de *CS* via LLMs no contexto do *benchmark* *Etapa 6* – Ampliação dos experimentos e avaliação empírica da utilidade das sugestões de refatoração por meio de *survey*
QP3	Fase 3 (4.1.3)	*Etapa 7* – Síntese integrada dos resultados, envolvendo a análise comparativa entre ferramentas tradicionais e LLMs, a triangulação das evidências empíricas e a identificação das condições sob as quais os LLMs complementam ou superam as abordagens clássicas

Observa-se que o *benchmark* atua como elemento integrador entre as fases metodológicas, permitindo que as QPs sejam respondidas de forma incremental, controlada e empiricamente fundamentada.

4.3 Cronograma das Atividades Restantes

A Fase 3 da metodologia, detalhada na Subseção 4.1.3, compreende o conjunto de atividades finais da pesquisa e tem como objetivo consolidar, validar e sistematizar o *benchmark* proposto, bem como integrar e interpretar os resultados empíricos obtidos ao longo das fases anteriores.

Considerando a complexidade das análises envolvidas, a necessidade de maturação dos resultados e o fato de que parte das etapas previstas ainda se encontra pendente de execução, estabeleceu-se um cronograma prospectivo até agosto de 2027, conforme apresentado na Tabela 4.2. Nesse contexto, descrevem-se, a seguir, os principais conjuntos de atividades associados a cada coluna da tabela, de modo a explicitar o significado das ações representadas no planejamento temporal.

A coluna *Benchmark* refere-se ao conjunto de atividades relacionadas à execução, ampliação e refinamento do *benchmark*, incluindo a aplicação de novas ferramentas tradicionais, a execução de LLMs adicionais, a padronização dos experimentos e a consolidação dos dados experimentais. Essas atividades ocorrem entre dezembro de 2025 e agosto de 2026, período no qual o *benchmark* é efetivamente expandido e estabilizado.

A coluna *Survey* compreende as atividades associadas à elaboração, à submissão para avaliação ética, à aplicação e à organização dos dados do questionário junto aos desenvolvedores. Esse processo ocorre entre março e agosto de 2026, em paralelo às etapas finais de execução do *benchmark*, considerando o tempo necessário para a apreciação e aprovação da pesquisa pelo Comitê de Ética em Pesquisa da universidade, bem como para a coleta e sistematização dos dados. Essa organização permite que a avaliação empírica da utilidade das abordagens baseadas em LLMs seja fundamentada em resultados experimentais previamente consolidados.

A coluna *Análise* corresponde à etapa de integração e interpretação dos dados obtidos, abrangendo tanto os resultados experimentais do *benchmark* quanto os dados provenientes do *survey*. Essa fase ocorre entre agosto de 2026 e fevereiro de 2027 e contempla a análise cruzada dos resultados, a identificação de padrões, limitações e convergências, bem como a validação progressiva das evidências empíricas que sustentam as QPs.

A coluna *Síntese Final* representa o período de consolidação dos achados da pesquisa,

no qual os resultados são organizados, interpretados de forma integrada e estruturados para compor os capítulos finais da tese. Essa etapa ocorre entre fevereiro e junho de 2027 e envolve a consolidação do *benchmark*, a sistematização das contribuições e a articulação das conclusões.

Por fim, a coluna *Escrita Final* corresponde à redação e ao refinamento da versão definitiva da tese, incluindo ajustes estruturais, revisão textual, organização final dos capítulos e preparação para a submissão e a defesa. Essa etapa ocorre entre abril e agosto de 2027, acompanhando a consolidação final dos resultados e garantindo a coesão global do trabalho.

De forma complementar às atividades explicitamente representadas no cronograma, a publicação dos resultados científicos será conduzida de maneira contínua ao longo de todo o período da pesquisa, à medida que os resultados se tornarem disponíveis e suficientemente maduros para submissão. Essa atividade não está vinculada a uma etapa específica do cronograma, uma vez que a produção e a submissão de artigos científicos dependem tanto da consolidação dos resultados quanto de fatores externos, como prazos de submissão, calendários de eventos científicos, ciclos editoriais de periódicos e a anuência do orientador.

Nesse sentido, a disseminação dos achados ocorrerá em paralelo às demais atividades metodológicas, respeitando os critérios de qualidade, relevância e maturidade estabelecidos pelo programa de doutorado. O cronograma apresentado reflete, portanto, um planejamento realista e metodologicamente consistente, alinhado à proposta da tese, às exigências do programa de doutorado e à complexidade inerente à construção e à validação de um *benchmark* para a detecção de *CS* em projetos desenvolvidos em TypeScript.

Tabela 4.2: Cronograma das atividades restantes da pesquisa (2025–2027)

Mês	***Benchmark***	***Survey***	**Análise**	**Síntese Final**	**Escrita Final**
Dez/2025	X				
Jan/2026	X				
Fev/2026	X				
Mar/2026	X	X			
Abr/2026	X	X			
Mai/2026	X	X			
Jun/2026	X	X			
Jul/2026	X	X			
Ago/2026	X	X	X		
Set/2026			X		
Out/2026			X		
Nov/2026			X		
Dez/2026			X		
Jan/2027			X		
Fev/2027			X	X	
Mar/2027				X	
Abr/2027				X	X
Mai/2027				X	X
Jun/2027				X	X
Jul/2027					X
Ago/2027					X

Capítulo 5

Resultados Preliminares

Este capítulo apresenta os resultados preliminares obtidos na execução inicial do protocolo experimental descrito no Capítulo 4, no contexto da Fase 2 (Preparação e Testes Iniciais). Em particular, são reportados os achados decorrentes das Etapas 4 e 5, correspondentes, respectivamente, à construção dos *scripts* e *prompts* para aplicação dos LLMs e à realização do Estudo Exploratório 1 em projetos reais desenvolvidos em TypeScript.

Esses resultados decorrem de um recorte controlado do desenho experimental completo, adotado com o objetivo de validar a viabilidade operacional da abordagem proposta, examinar padrões iniciais de detecção de *CS* e avaliar a consistência dos instrumentos metodológicos empregados.

O objetivo central deste capítulo não é estabelecer conclusões definitivas, mas sim verificar a adequação do protocolo experimental, analisar o comportamento inicial das abordagens investigadas e produzir evidências empíricas preliminares que subsidiem o refinamento do desenho metodológico e a ampliação prevista para as etapas subsequentes da pesquisa, especialmente aquelas associadas à Fase 3.

Os achados apresentados refletem tanto execuções já concluídas quanto resultados ainda parciais, cuja consolidação depende da ampliação do escopo experimental prevista na metodologia. Dessa forma, este capítulo deve ser interpretado como um marco intermediário no desenvolvimento da proposta de tese, estabelecendo uma base empírica inicial que sustenta a continuidade da execução do protocolo experimental.

5.1 Caracterização do Estudo Exploratório Preliminar

A presente seção detalha a configuração experimental, os procedimentos adotados e os critérios de delimitação utilizados na realização do Estudo Exploratório 1, correspondente à execução inicial do protocolo experimental na Fase 2.

Nesse estudo exploratório, buscou-se verificar a consistência do desenho experimental, identificar limitações operacionais e avaliar a adequação dos instrumentos metodológicos empregados, de modo a subsidiar o refinamento do *benchmark* proposto.

A configuração experimental adotada nesta fase foi deliberadamente restrita, contemplando uma ferramenta tradicional de análise estática, o SonarQube, utilizada como *baseline*, e um modelo de linguagem, o GPT-4o. Esse recorte visa reduzir as variáveis intervenientes, facilitar o controle experimental e permitir a observação sistemática dos padrões iniciais de detecção.

As subseções a seguir descrevem, respectivamente, os procedimentos de pré-processamento e definição do recorte (Subseção 5.1.1), a caracterização do conjunto de dados (Subseção 5.1.2), o escopo dos *CS* considerados (Subseção 5.1.3) e as ferramentas e modelos avaliados (Subseção 5.1.4).

5.1.1 Processamento Experimental e Definição do Recorte Preliminar

Antes da execução do protocolo experimental propriamente dito, foi realizada uma etapa de pré-processamento com o objetivo de subsidiar a tomada de decisão quanto aos principais recortes adotados nesta fase da proposta de tese.

Essa etapa teve caráter exploratório e metodologicamente orientado, permitindo fundamentar escolhas relacionadas aos *prompts*, aos modelos de linguagem, às ferramentas tradicionais de análise estática para detecção de *CS* e aos procedimentos de organização e filtragem dos dados de entrada, antes da consolidação do desenho experimental final. Todos os artefatos produzidos ao longo dessa etapa encontram-se reunidos e indicados no Apêndice C.

No que se refere aos modelos de linguagem, foram conduzidos testes preliminares com diferentes LLMs, incluindo modelos proprietários, em versões gratuitas e comerciais, bem como modelos de código aberto executados localmente por meio de infraestrutura própria. Entre os modelos explorados incluem-se, por exemplo:

I. Claude 3.5 (web *premium*);

II. Gemini 1.0 (web *free*);

III. GPT (4, 4o, 4o mini, o1-preview, o1-mini, todos web *premium*);

IV. Codellama (modelo 7B local, via ollama);

V. LLaMA 2 (modelo 7B local, via ollama); e

VI. Mistral 0.2 (modelo 7B local, via ollama).

Esses experimentos tiveram como objetivo observar aspectos como a capacidade de detecção de *CS*, a estabilidade das respostas, a frequência de alucinações, a necessidade de pós-processamento e a aderência ao formato de saída previsto pelo protocolo. A partir dessas observações, foi possível definir um modelo mais adequado ao recorte preliminar adotado nesta fase. Os registros consolidados desses testes encontram-se organizados na planilha `TestesLLMs`, incluída na pasta `PreProcessamento`.

De forma complementar, foram realizados testes exploratórios com diferentes ferramentas tradicionais de análise estática, incluindo:

I. SonarQube;

II. LiveRefactorings;

III. Embold;

IV. ESLint; e

V. TSLint.

Esses testes permitiram avaliar a cobertura de detecção de *CS* em projetos em TypeScript, bem como o nível de abstração adotado por cada ferramenta na definição dos problemas reportados. Logo, observou-se que as ferramentas utilizam nomenclaturas próprias e frequentemente descrevem *CS* com base em características técnicas específicas, não necessariamente alinhadas, de forma direta, ao nível de abstração conceitual proposto por Fowler [33, 32].

Essa heterogeneidade torna necessária a realização de operações de tradução conceitual, mapeando os achados das ferramentas para o catálogo de Fowler, conforme documentado na planilha `GuideSmells`, incluída na pasta `PreProcessamento`.

Cabe destacar que, diferentemente da abordagem de Fowler, que define *CS* em um nível de abstração mais elevado e independente da linguagem de programação, as ferramentas tradicionais tendem a apresentar definições dependentes da linguagem analisada, podendo variar tanto na cobertura quanto na eficácia da detecção conforme o tipo de *CS* e a tecnologia empregada. Assim, uma mesma ferramenta pode apresentar maior capacidade de detecção para determinados *CS* em uma linguagem específica, ao mesmo tempo em que demonstra limitações em outros tipos ou contextos, o que reforça a necessidade de um recorte controlado nesta fase preliminar.

No que diz respeito ao conjunto de *CS* inicialmente considerados, partiu-se de um escopo amplo, combinando o catálogo clássico de Fowler [33, 32] com os conjuntos de *CS* descritos na documentação oficial das ferramentas avaliadas. Esse escopo inicial foi progressivamente refinado, considerando critérios como relevância conceitual, recorrência na literatura, viabilidade de detecção em projetos em TypeScript e limitações observadas durante os testes exploratórios, até a definição do subconjunto analisado neste capítulo.

Adicionalmente, foi realizado um processo sistemático de extração e organização local dos projetos selecionados do GitHub, com o objetivo de padronizar os dados de entrada e garantir a consistência da unidade de análise. Os repositórios foram armazenados localmente, organizados por identificador e submetidos a um pré-processamento que preserva exclusivamente os arquivos relevantes em TypeScript. Esse processo foi automatizado por meio de *scripts* desenvolvidos em Python, responsáveis pela criação da estrutura de diretórios, pela filtragem dos arquivos e pela extração de métricas básicas, como o número de linhas de código, organizadas na pasta `Fragmentacao`.

De modo geral, essa etapa de pré-processamento permitiu, a partir de pequenos experimentos controlados, identificar limitações práticas, refinar decisões metodológicas e definir um recorte experimental viável e coerente para a execução do protocolo de investigação adotado nesta fase preliminar. Ressalta-se que os resultados dessa etapa têm caráter exclusivamente instrumental e não constituem resultados formais da pesquisa, servindo apenas como base para a consolidação do protocolo experimental empregado nos experimentos apresen-

tados neste capítulo.

5.1.2 Dataset e Projetos Analisados

Após o pré-processamento dos repositórios selecionados, o conjunto de dados analisado nesta fase preliminar da proposta de tese é composto por 106 projetos desenvolvidos em TypeScript, totalizando 691.590 linhas de código (LOC). Todo o código-fonte analisado foi obtido de repositórios públicos na plataforma GitHub.

A amostra foi definida com base em critérios de relevância, diversidade estrutural e disponibilidade pública do código-fonte. O *dataset* resultante contempla repositórios de diferentes tamanhos e níveis de complexidade, buscando representar cenários realistas de desenvolvimento de software em TypeScript.

A análise exploratória da distribuição de LOC revelou uma assimetria acentuada, com a presença de projetos significativamente maiores do que a maioria da amostra, caracterizando *outliers* severos. Nesse contexto, a média aritmética mostrou-se inadequada como medida representativa, pois é fortemente influenciada por valores extremos. Assim, adotou-se a mediana como principal medida de tendência central, acompanhada do intervalo interquartil (IQR), por sua maior robustez estatística.

A Tabela 5.1 apresenta as estatísticas descritivas da distribuição de LOC após o pré-processamento. Observa-se uma mediana de 490 linhas de código, com primeiro quartil (Q1) de 417 LOC e terceiro quartil (Q3) de 1.021 LOC, indicando que 50% dos projetos concentram-se nesse intervalo. O valor máximo observado, de 531.471 LOC, evidencia a presença de *outliers* extremos, responsáveis por distorcer métricas baseadas na média.

Tabela 5.1: Estatísticas descritivas da distribuição de LOC dos projetos analisados

Métrica	**LOC**
Mínimo	53
1º Quartil (Q1 – 25%)	$\approx$ 417
Mediana (Q2 – 50%)	490
3º Quartil (Q3 – 75%)	$\approx$ 1.021
Máximo	531.471

Com base nos quartis da distribuição de LOC, os projetos analisados foram estratificados por porte. Aqueles com valores de LOC inferiores a 417 foram classificados como de pequeno porte; os que apresentam valores maiores ou iguais a 417 e menores ou iguais a 1.021, como de médio porte; e os com valores superiores a 1.021, como de grande porte. Essa estratificação fundamenta-se em critérios estatísticos e permite análises mais coerentes, reduzindo o impacto de valores extremos nas métricas globais.

A Tabela 5.2 apresenta a distribuição da amostra por porte, e a Figura 5.1 apresenta o *boxplot* da distribuição de LOC em escala logarítmica, na qual a região interquartil torna-se visualmente discernível.

Tabela 5.2: Classificação dos projetos por porte com base nos quartis da distribuição de LOC

Porte do Projeto	**Intervalo de LOC**	**Quantidade de Projetos**
Pequeno	$\text{LOC} < 417$	27
Médio	$417 \leq \text{LOC} \leq 1.021$	52
Grande	$\text{LOC} > 1.021$	27
Total		**106**

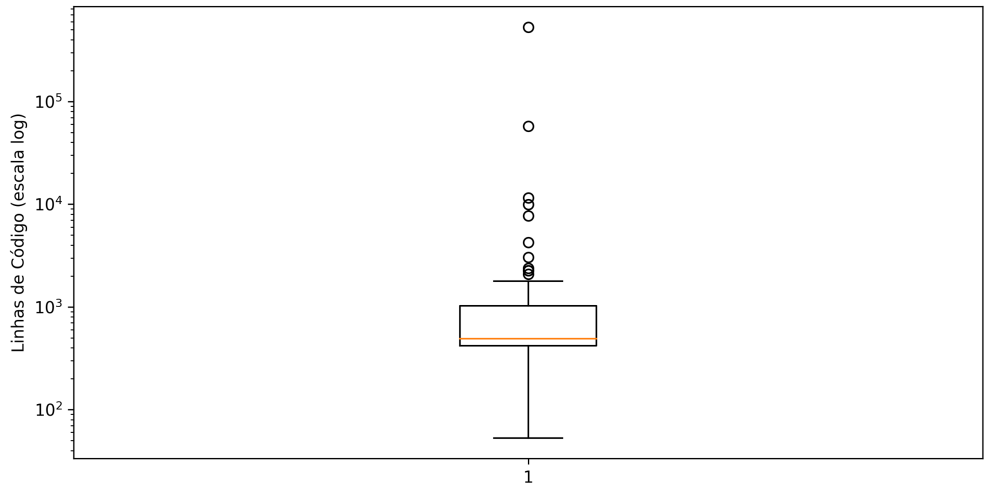

Figura 5.1: Boxplot da distribuição de LOC (Escala Logarítmica)

Do total de 106 projetos analisados, 27 foram classificados como de pequeno porte, 52 como de médio porte e 27 como de grande porte, evidenciando uma concentração predomi-

nante na faixa intermediária da distribuição.

Embora o *boxplot* tradicional represente corretamente a distribuição dos dados, a presença de *outliers* extremos compromete sua legibilidade gráfica, comprimindo a região interquartil próxima ao eixo inferior. Esse comportamento evidencia a forte assimetria da distribuição e justifica a adoção de visualizações complementares. Logo, observa-se na Figura 5.1 uma concentração dos projetos analisados no intervalo de 400 a 1.000 linhas de código. Além disso, uma parcela da amostra apresenta valores superiores, incluindo casos extremos (*outliers*) que motivam a adoção da escala logarítmica.

5.1.3 CS Considerados

Como resultado do processo de refinamento descrito na Subseção 5.1.1, a análise concentrou-se em um subconjunto de *CS* selecionados a partir do catálogo clássico proposto por Fowler [32]. A definição desse escopo não teve caráter exaustivo, sendo orientada por critérios metodológicos específicos da fase preliminar da proposta de tese.

A escolha desses *CS* foi motivada por sua relevância conceitual na literatura, pela recorrência em estudos empíricos sobre qualidade de software e, sobretudo, pela insuficiência de detecção observada em ferramentas tradicionais de análise estática, conforme identificado na revisão da literatura e em testes exploratórios conduzidos ao longo da pesquisa.

De acordo com o escopo definido na Tabela 2.1, os *CS* analisados nesta etapa foram:

- S29 – *Parallel Inheritance Hierarchies*;
- S30 – *Lazy Class*;
- S34 – *Middle Man*;
- S36 – *Alternative Classes with Different Interfaces*;
- S37 – *Incomplete Library Class*;
- S39 – *Refused Bequest*.

Ao restringir a análise a esse conjunto específico, buscou-se viabilizar uma investigação mais controlada e aprofundada do comportamento das abordagens avaliadas, estabelecendo

uma base sólida para a ampliação do escopo de *CS* nas etapas subsequentes do protocolo experimental.

5.1.4 Ferramentas e Modelos Avaliados

Nesta fase preliminar da pesquisa, os experimentos foram conduzidos utilizando uma ferramenta tradicional de análise estática, o SonarQube, e um LLM, o GPT-4o. Esse recorte experimental foi definido com o objetivo de validar o protocolo metodológico proposto e avaliar a viabilidade da abordagem antes de sua ampliação.

Antes da definição desse recorte, foram realizados testes exploratórios com múltiplos LLMs e ferramentas tradicionais, visando analisar aspectos como a capacidade de detecção de *CS*, o nível de detalhamento das justificativas, o alinhamento com o catálogo de Fowler e a necessidade de pós-processamento dos resultados. Esses testes tiveram caráter exclusivamente exploratório e não integram os resultados analisados neste capítulo. Os registros completos desses testes exploratórios, bem como informações adicionais sobre os modelos e as ferramentas avaliados, encontram-se descritos no Apêndice C.

5.2 Resultados da Detecção por Ferramenta Tradicional

Esta seção apresenta os resultados empíricos obtidos a partir da aplicação de uma ferramenta tradicional de análise estática, o SonarQube, utilizada como *baseline* nesta fase preliminar da proposta de tese. O objetivo não é avaliar exaustivamente o desempenho da ferramenta, mas caracterizar sua cobertura e comportamento frente ao escopo de *CS* definido, servindo como referência para as análises comparativas subsequentes.

5.2.1 Visão Geral dos Achados

Os resultados da detecção realizada pelo SonarQube foram consolidados em uma planilha analítica, construída especificamente para esta pesquisa, considerando o escopo dos seis *CS* selecionados com base no catálogo de Fowler e no processo de tradução conceitual descrito anteriormente. Essa planilha organiza, para cada projeto analisado, informações como o tamanho do projeto (LOC), o número de classes, o número de ocorrências detectadas para

cada *CS* do escopo e o total de *CS* reportados pela ferramenta. A versão completa dessa planilha encontra-se disponível no Apêndice C, no arquivo `analiseGeral_SonarQube`, localizado na pasta `Analise`.

A análise dos dados consolidados evidencia que, considerando o contexto de projetos desenvolvidos em TypeScript e o escopo de *CS* adotado neste trabalho, apenas o *CS Lazy Class* apresentou ocorrências detectáveis pelo SonarQube após o processo de mapeamento conceitual. Nenhum dos demais *CS* considerados (*Parallel Inheritance Hierarchies*, *Middle Man*, *Alternative Classes with Different Interfaces*, *Incomplete Library Class* e *Refused Bequest*) foi identificado pela ferramenta nesse contexto.

Em termos quantitativos, o SonarQube detectou um total de 430 ocorrências do *CS Lazy Class*, distribuídas entre os 106 projetos do *dataset*. Esses valores, embora indiquem a capacidade parcial da ferramenta para identificar esse *CS*, reforçam a limitação de cobertura quando comparados ao escopo conceitual adotado nesta pesquisa. Ressalta-se que os valores exatos por projeto, bem como os somatórios globais, encontram-se detalhados na planilha anexa completa. Ao estratificar as ocorrências do *CS Lazy Class* (S30) por porte de projeto, a distribuição é apresentada na Tabela 5.3.

Tabela 5.3: Distribuição das ocorrências de *Lazy Class* por porte de projeto

Porte do Projeto	**Nº de Ocorrências de S30**
Pequeno	39
Médio	155
Grande	236
Total	430

Conforme mostrado na Tabela 5.3, observa-se que a detecção realizada pelo SonarQube distribui-se de forma assimétrica entre as categorias consideradas. Projetos de grande porte concentram o maior número absoluto de ocorrências (236), seguidos pelos de médio porte (155), enquanto os de pequeno porte apresentam um número significativamente menor de detecções (39). Essa distribuição sugere que, no escopo analisado, a ocorrência e a detecção do *Lazy Class* tendem a se intensificar em sistemas maiores, ainda que tais resultados devam ser interpretados de forma estritamente descritiva e restrita ao recorte experimental adotado nesta fase preliminar.

Já a Tabela 5.4 apresenta uma visão sintética dos resultados de detecção do SonarQube considerando o escopo de *CS* definido neste estudo.

Tabela 5.4: Síntese da detecção de *CS* pelo SonarQube no escopo considerado

ID CS	**Detectado**	**Observação**
S29	Não	Não suportado no contexto analisado.
S30	Sim	Detecção parcial.
S34	Não	Não suportado no contexto analisado.
S36	Não	Não suportado no contexto analisado.
S37	Não	Não suportado no contexto analisado.
S39	Não	Não suportado no contexto analisado.

A indicação de que determinados *CS* não foram detectados deve ser interpretada como a ausência de suporte direto da ferramenta para esses casos, no contexto da linguagem TypeScript, à luz da taxonomia de Fowler adotada como referência conceitual. Em outras palavras, tais *CS* não possuem regras equivalentes no SonarQube que permitam sua identificação direta conforme o nível de abstração considerado. Por sua vez, o *CS Lazy Class* foi parcialmente detectado, evidenciando que, mesmo quando há suporte, a cobertura da ferramenta permanece limitada em relação ao escopo analisado.

Durante a execução da ferramenta, outros achados de análise foram reportados pelo SonarQube como *CS*. No entanto, tais ocorrências foram desconsideradas nesta análise por não integrarem o escopo de *CS* definido para o estudo.

De modo geral, os resultados indicam que, considerando exclusivamente o subconjunto de *CS* analisado neste experimento preliminar, a cobertura do SonarQube mostrou-se limitada. Essa constatação deve ser interpretada à luz do recorte metodológico adotado, uma vez que o escopo de *CS* foi deliberadamente definido a partir de casos que apresentam baixa ou nenhuma cobertura por ferramentas tradicionais de análise estática. Portanto, os achados não caracterizam uma limitação global da ferramenta, mas refletem sua adequação restrita ao conjunto específico de *CS* considerado, reforçando a motivação para investigar abordagens complementares no contexto do protocolo proposto.

5.2.2 Limitações Observadas

A principal limitação observada na aplicação do SonarQube refere-se à discrepância entre a taxonomia de problemas reportados pela ferramenta e o nível de abstração conceitual adotado no catálogo de Fowler. Enquanto Fowler define *CS* de forma mais geral e independente de linguagem, o SonarQube baseia-se em regras técnicas específicas, cuja cobertura e aplicabilidade variam conforme a linguagem de programação analisada.

No contexto de projetos TypeScript, essa característica impõe restrições à capacidade de detecção de determinados *CS* clássicos, tornando necessário um processo de tradução conceitual para mapear os problemas reportados pela ferramenta aos *CS* definidos no escopo deste estudo.

O mapeamento completo entre as regras do SonarQube e os *CS* de Fowler, bem como as justificativas adotadas para esse alinhamento, encontram-se documentados no Apêndice C, especificamente na planilha `GuideSmells`, disponível na pasta `PreProcessamento`. Essas limitações reforçam o papel do SonarQube como *baseline* tradicional nesta fase preliminar, evidenciando tanto sua utilidade prática quanto suas restrições conceituais, e motivam a investigação de abordagens complementares, como as baseadas em LLMs, discutidas nas seções subsequentes deste capítulo.

5.3 Resultados da Detecção por LLM

Esta seção apresenta os resultados preliminares obtidos a partir da aplicação de um LLM, o GPT-4o, no contexto do protocolo experimental definido para esta fase da proposta de tese. Diferentemente da ferramenta tradicional de análise estática, o LLM é avaliado aqui como uma abordagem emergente, capaz de explorar aspectos contextuais e semânticos do código-fonte, sem o uso explícito de regras pré-definidas.

O objetivo desta seção não é avaliar a correção semântica das detecções realizadas pelo modelo, tampouco estabelecer conclusões definitivas sobre seu desempenho, mas caracterizar a cobertura empírica observada no escopo de *CS* considerado, bem como descrever padrões iniciais de comportamento identificados durante os experimentos preliminares.

5.3.1 Desempenho Geral do Modelo

Os resultados da detecção realizada pelo GPT-4o foram consolidados em uma planilha analítica, construída especificamente para este estudo, de forma análoga à utilizada na análise do SonarQube. Essa planilha organiza, para cada projeto analisado, informações como o porte do projeto (LOC), o número de classes e a quantidade de ocorrências detectadas para cada *CS* definido no escopo experimental, bem como o total de *CS* identificados em cada projeto. A versão completa dessa planilha encontra-se disponível no Apêndice C, no arquivo `analiseGeral_GPT4o`, localizado na pasta `Analise`.

Considerando o conjunto de 106 projetos analisados, o GPT-4o identificou um total de 4.090 ocorrências de *CS* distribuídas ao longo do *dataset*. Diferentemente do que foi observado na ferramenta tradicional, o modelo apresentou capacidade de identificar ocorrências em todos os seis *CS* considerados no escopo desta fase preliminar. A Tabela 5.5 apresenta a distribuição agregada das ocorrências por tipo de *CS* detectado pelo GPT-4o.

Tabela 5.5: Síntese da detecção de *CS* pelo GPT-4o no escopo considerado

ID CS	**Nº de Ocorrências**
S29	63
S30	2.621
S34	109
S36	351
S37	439
S39	507
Total	**4.090**

Observa-se na Tabela 5.5 que o *CS Lazy Class* (S30) apresenta a maior incidência absoluta, correspondendo à maior parcela das ocorrências detectadas pelo modelo. Ainda assim, todos os demais *CS* do escopo foram identificados em múltiplos projetos, indicando cobertura empírica completa no contexto analisado.

5.3.2 Distribuição das Detecções por Porte de Projeto

Com base na estratificação dos projetos por porte, definida a partir dos quartis da distribuição de LOC, foi analisada a distribuição das ocorrências de *CS* detectadas pelo GPT-4o em projetos de pequeno, médio e grande porte. A Tabela 5.6 apresenta a distribuição agregada das ocorrências por tipo de *CS* e por porte de projeto.

Tabela 5.6: Distribuição das ocorrências de *CS* detectadas pelo GPT-4o por porte de projeto

Porte	**S29**	**S30**	**S34**	**S36**	**S37**	**S39**	**Total**
Pequeno	6	26	10	4	7	6	59
Médio	7	107	13	12	15	15	169
Grande	50	2.488	86	335	417	486	3.862
Total	63	2.621	109	351	439	507	4.090

Os dados indicam uma elevada concentração de ocorrências em projetos de grande porte, especialmente no *CS Lazy Class* (S30). Contudo, todos os *CS* do escopo foram identificados em projetos de pequeno, médio e grande porte, ainda que com incidências distintas. Essa distribuição sugere que, no contexto analisado, a ocorrência e a detecção de *CS* dependem fortemente das características estruturais de cada projeto, e não exclusivamente de seu porte. Assim, embora sistemas maiores registrem um número absoluto maior de ocorrências, a presença de *CS* em projetos menores reforça a heterogeneidade observada no *dataset*.

Além da estratificação por porte, a análise em nível de projeto evidencia uma distribuição fortemente assimétrica das ocorrências de *CS*. Observa-se que projetos pertencentes à mesma categoria de porte podem apresentar comportamentos substancialmente distintos, incluindo casos com elevada concentração de ocorrências e casos em que nenhum *CS* foi identificado pelo modelo. Esse padrão indica que a incidência de *CS* está mais fortemente associada às características estruturais e às decisões de projeto de cada sistema do que ao seu tamanho.

Adicionalmente, verifica-se a presença de *outliers* severos, em que um número reduzido de projetos concentra uma parcela significativa do total de ocorrências detectadas. Tal comportamento reforça a necessidade de cautela na interpretação de métricas agregadas e justifica a adoção de análises estratificadas, bem como o uso futuro de métricas normalizadas, que serão exploradas nas etapas subsequentes do estudo.

5.3.3 Análise do Comportamento do LLM

A análise qualitativa das respostas geradas pelo GPT-4o revelou variações no nível de detalhamento, na forma de justificativa apresentada e na estrutura das saídas. Em determinados casos, o modelo forneceu explicações extensas e contextualizadas para a identificação dos *CS*, enquanto, em outros, apresentou respostas mais sintéticas, exigindo etapas adicionais de pós-processamento para adequação ao formato esperado.

Observou-se, ainda, que o comportamento do modelo é sensível ao protocolo de *prompting* adotado, incluindo a forma de apresentação do código-fonte, a explicitação do escopo dos *CS* e as instruções fornecidas quanto ao formato da saída. Esses aspectos reforçam a importância de um protocolo cuidadosamente definido e documentado, de modo a reduzir variações indesejadas entre execuções sucessivas.

O *prompt* adotado neste estudo foi estruturado para explicitar o escopo dos *CS* analisados, definir a unidade de análise no nível de arquivos TypeScript e impor um formato de saída padronizado, compatível com o protocolo experimental proposto. Essa estrutura visa reduzir variações indesejáveis nas respostas do modelo, facilitar o pós-processamento dos resultados e garantir a reprodutibilidade das análises. O arquivo contendo o *prompt* utilizado (`PromptEntrada`) encontra-se disponível no Apêndice C, no diretório raiz da pasta `web`.

5.3.4 Casos de Detecção Não Capturados pela Ferramenta Tradicional

Foram identificados diversos casos em que o GPT-4o sinalizou a presença de *CS* que não haviam sido reportados pelo SonarQube no mesmo conjunto de projetos. Esses casos ilustram a capacidade do modelo de explorar relações contextuais e semânticas do código que não são diretamente capturadas por regras técnicas específicas.

A Figura 5.2 apresenta um exemplo ilustrativo de um *CS* identificado pelo GPT-4o, mas não detectado pela ferramenta tradicional. Esse tipo de evidência não deve ser interpretado, nesta etapa, como confirmação da correção da detecção, mas como um indicativo do potencial do modelo para identificar padrões distintos daqueles contemplados pelas abordagens baseadas em regras.

Figura 5.2: Exemplo de *CS* identificado pelo GPT-4o e não detectado pelo SonarQube

Embora a Figura 5.2 permita uma visualização sintética do resultado produzido pelo modelo, ela não abrange a totalidade das informações geradas durante o processo de detecção. Com o objetivo de detalhar a estrutura de saída produzida pelo GPT-4o e explicitar o nível de informação fornecido pelo modelo, apresenta-se, a seguir, a saída completa correspondente ao exemplo ilustrado, configurando um caso real de detecção.

Classe analisada: `main-layout.component.ts`
Intervalo de linhas (LOC): 7–415
***CS* detectados:**

- Lazy Class
- Refused Bequest

Descrição: A classe `MainLayoutComponent` apresenta diversos atributos e métodos pouco utilizados ou de contribuição limitada para sua funcionalidade principal, indicando possível subutilização. Além disso, a classe herda comportamentos parcialmente sobrescritos ou minimamente utilizados, o que caracteriza uso inadequado da herança.
Refatorações sugeridas:

- Move Method
- Replace Inheritance with Delegation

Processo de refatoração: Realocar métodos e propriedades subutilizados para classes ou serviços mais adequados. Substituir herança por delegação para funcionalidades que não se alinham ao propósito principal da classe, simplificando a hierarquia e reduzindo dependências desnecessárias.

O exemplo evidencia a identificação simultânea de múltiplos *CS* em uma mesma classe

TypeScript, bem como a indicação do intervalo de linhas afetado, a descrição do problema e as sugestões de refatoração associadas. Esses achados reforçam a necessidade de análises comparativas mais aprofundadas e de processos sistemáticos de validação, que serão conduzidos nas etapas subsequentes do protocolo experimental definido nesta proposta de tese.

5.4 Análise Comparativa entre a Ferramenta Tradicional e o LLM

Esta seção apresenta uma análise comparativa preliminar entre os resultados obtidos com a ferramenta tradicional de análise estática, o SonarQube, e com o LLM GPT-4o, considerando exclusivamente o escopo de *CS* definido para esta fase da proposta de tese. O objetivo não é estabelecer uma avaliação conclusiva de desempenho entre as abordagens, mas examinar diferenças, sobreposições e padrões de complementaridade observados na detecção dos *CS* analisados.

A comparação é conduzida à luz do recorte metodológico adotado, considerando tanto as limitações impostas pelo escopo reduzido de *CS* quanto o caráter preliminar dos experimentos realizados. Assim, os resultados discutidos nesta seção devem ser interpretados como evidências iniciais que subsidiam a ampliação e o refinamento do protocolo experimental nas etapas subsequentes da pesquisa.

5.4.1 Sobreposição e Divergências de Detecção

A análise comparativa dos resultados evidencia uma sobreposição limitada entre os *CS* detectados pelo SonarQube e pelo GPT-4o no contexto dos projetos analisados. Considerando o escopo definido, apenas o *CS Lazy Class* (S30) foi identificado por ambas as abordagens, ainda que com diferenças relevantes na cobertura e no volume de ocorrências reportadas.

Enquanto o SonarQube apresentou detecção parcial, restrita ao *Lazy Class*, o GPT-4o foi capaz de identificar ocorrências em todos os seis *CS* considerados neste estudo. Essa divergência decorre, em grande medida, das diferenças fundamentais entre as abordagens: a ferramenta tradicional baseia-se em regras técnicas específicas, sensíveis à linguagem de programação, enquanto o LLM opera com base em inferências semânticas e estruturais extraídas

do código-fonte. A Tabela 5.7 sintetiza a comparação preliminar entre as duas abordagens, indicando, para cada *CS* do escopo, se há detecção ou não.

Tabela 5.7: Comparação preliminar da detecção de *CS* pelo SonarQube e pelo GPT-4o

ID CS	**SonarQube**	**GPT-4o**
S29	Não	Sim
S30	Sim (parcial)	Sim
S34	Não	Sim
S36	Não	Sim
S37	Não	Sim
S39	Não	Sim

Além da dimensão conceitual da cobertura, observa-se uma diferença expressiva no volume de *CS* detectados pelas duas abordagens. Considerando o mesmo conjunto de 106 projetos e o escopo definido, o SonarQube reportou um total de 430 ocorrências, todas associadas ao *Lazy Class*. Em contraste, o GPT-4o identificou 4.090 ocorrências de *CS*, distribuídas entre os seis tipos analisados. A Tabela 5.8 apresenta uma comparação direta dos valores absolutos de ocorrências reportadas por cada abordagem, evidenciando a assimetria quantitativa observada no contexto analisado.

Tabela 5.8: Comparação quantitativa da detecção de *CS* pelo SonarQube e pelo GPT-4o

Code Smell	**SonarQube**	**GPT-4o**
Parallel Inheritance Hierarchies (S29)	0	63
Lazy Class (S30)	430	2.621
Middle Man (S34)	0	109
Alternative Classes with Different Interfaces (S36)	0	351
Incomplete Library Class (S37)	0	439
Refused Bequest (S39)	0	507
Total	**430**	**4.090**

Os resultados quantitativos reforçam a limitação de cobertura da ferramenta tradicional sob a taxonomia de Fowler no contexto da linguagem TypeScript, bem como o maior alcance

empírico do modelo de linguagem. Ressalta-se que a ausência de detecção de determinados *CS* pelo SonarQube não implica a inexistência desses problemas nos projetos analisados, mas reflete a falta de regras equivalentes na ferramenta para capturá-los no nível de abstração considerado.

Por fim, essa assimetria de cobertura deve ser interpretada à luz do recorte metodológico adotado, uma vez que o conjunto de *CS* analisado foi deliberadamente selecionado a partir de casos que apresentam baixa ou nenhuma cobertura por ferramentas tradicionais de análise estática.

5.4.2 Padrões de Complementaridade

A comparação entre os resultados obtidos com o SonarQube e o GPT-4o revela padrões claros de complementaridade entre as abordagens. A ferramenta tradicional mostrou-se eficaz na identificação de problemas associados a regras técnicas bem definidas, ainda que de forma limitada no escopo considerado, enquanto o LLM demonstrou maior capacidade de capturar *CS* que dependem de interpretação semântica, de relações estruturais entre entidades e de análise contextual do código.

Em particular, *CS* como *Middle Man*, *Parallel Inheritance Hierarchies* e *Alternative Classes with Different Interfaces*, cujas identificações exigem a análise de interações entre múltiplas classes ou a compreensão de papéis arquiteturais, foram detectados exclusivamente pelo GPT-4o. Esses casos ilustram o potencial dos LLMs para atuar como complemento às ferramentas tradicionais, especialmente em cenários em que regras sintáticas isoladas são insuficientes para capturar padrões de degradação de projeto em níveis mais elevados de abstração.

Por outro lado, a análise também evidenciou limitações inerentes ao uso de LLMs, como a sensibilidade ao protocolo de *prompting*, a necessidade de pós-processamento das respostas e a ausência de garantias formais quanto à consistência das detecções. Esses fatores indicam que, no estágio atual da pesquisa, os LLMs não substituem integralmente ferramentas tradicionais de análise estática, mas ampliam o espaço de análise possível quando utilizados de forma integrada.

Dessa forma, os resultados preliminares sugerem que abordagens híbridas, que combinam ferramentas tradicionais de análise estática e LLMs, constituem uma direção promissora

para a detecção mais abrangente de *CS*. Essa constatação fundamenta a continuidade do protocolo experimental proposto, que prevê a ampliação do conjunto de ferramentas, modelos e *CS* analisados nas etapas subsequentes da pesquisa.

5.5 Síntese dos Resultados Preliminares

Os resultados preliminares apresentados neste capítulo evidenciam a viabilidade operacional do protocolo experimental proposto e fornecem indícios iniciais relevantes sobre o comportamento de diferentes abordagens de detecção de *CS* em projetos desenvolvidos em TypeScript. Embora ainda não permitam conclusões definitivas, esses achados cumprem o papel de validar escolhas metodológicas, identificar limitações práticas e orientar a continuidade do estudo.

A análise realizada com a ferramenta tradicional SonarQube, adotada como *baseline*, indicou uma cobertura limitada no escopo de *CS* selecionado. Em particular, apenas o *CS Lazy Class* apresentou ocorrências detectáveis após o processo de tradução conceitual para a taxonomia de Fowler. Esse resultado não caracteriza uma limitação global da ferramenta, mas reflete o recorte metodológico deliberadamente adotado, voltado à investigação de *CS* que apresentam baixa ou nenhuma cobertura por ferramentas tradicionais de análise estática no contexto da linguagem TypeScript.

Em contraste, os experimentos preliminares com o GPT-4o evidenciaram a capacidade do modelo de identificar ocorrências em todos os *CS* considerados no escopo, incluindo aqueles não reportados pela ferramenta tradicional. A análise qualitativa das respostas revelou que o modelo é capaz de capturar aspectos semânticos e estruturais do código-fonte que extrapolam o alcance de regras técnicas isoladas, ainda que apresente desafios relacionados à variabilidade das respostas, à necessidade de pós-processamento e à sensibilidade ao protocolo de *prompting* adotado.

A comparação preliminar entre as abordagens reforça a existência de padrões claros de complementaridade. Enquanto ferramentas tradicionais oferecem detecções mais estáveis e baseadas em regras bem definidas, os LLMs demonstram potencial para ampliar a cobertura da análise, especialmente em *CS* que dependem de contextos mais amplos, de relações interclasse ou de interpretação arquitetural. Esses resultados sustentam a hipótese de que

abordagens híbridas podem oferecer benefícios significativos na detecção de *CS*, quando comparadas ao uso isolado de uma única técnica.

De forma geral, os achados apresentados confirmam que o desenho metodológico adotado é adequado aos objetivos da pesquisa e que o protocolo experimental pode ser executado de forma controlada e reprodutível. Ao mesmo tempo, os resultados evidenciam a necessidade de ampliar o escopo experimental, tanto em termos de ferramentas e modelos avaliados quanto do conjunto de *CS* considerados, conforme previsto na metodologia.

Assim, esta síntese consolida o Capítulo 5 como um marco intermediário da proposta de tese, fornecendo uma base empírica inicial que justifica a continuidade da execução do protocolo experimental e a realização de experimentos adicionais, necessários à obtenção de resultados mais robustos e conclusivos nas etapas subsequentes da pesquisa.

Com o objetivo de consolidar os principais achados discutidos ao longo deste capítulo, a Tabela 5.9 apresenta um resumo comparativo dos resultados preliminares obtidos com a ferramenta tradicional e com o LLM.

Tabela 5.9: Síntese dos resultados preliminares por abordagem

Dimensão	**SonarQube**	**GPT-4o**
Cobertura de CS	Restrita ao *Lazy Class* (S30).	Abrange todos os seis *CS* do escopo.
Natureza da Detecção	Baseada em regras técnicas específicas.	Baseada em inferências semânticas e estruturais.
Sensibilidade ao Contexto	Limitada ao nível local e sintático.	Capaz de capturar relações interclasse e contextuais.
Volume de Ocorrências	Baixo, concentrado em um único CS.	Elevado e distribuído entre diferentes CS.
Estabilidade da Saída	Alta estabilidade e previsibilidade.	Variável, dependente do protocolo de *prompting*.
Necessidade de Pós-Processamento	Mínima.	Presente para padronização e validação.
Papel no Estudo	*Baseline* tradicional.	Abordagem complementar exploratória.

5.6 Limitações dos Resultados Preliminares

Os resultados apresentados neste capítulo devem ser interpretados à luz das limitações inerentes ao caráter preliminar da investigação e ao recorte metodológico adotado nesta fase da proposta de tese. Tais limitações não comprometem a validade do estudo, mas delimitam o alcance das inferências a partir dos achados iniciais.

Uma primeira limitação refere-se à escala experimental. Embora o *dataset* analisado contemple um número expressivo de projetos, a execução do protocolo ainda se restringe a um recorte inicial, sem a exploração sistemática de múltiplos modelos de linguagem e de ferramentas tradicionais, conforme previsto nas etapas subsequentes da metodologia. Assim, os resultados obtidos refletem o comportamento específico do SonarQube e do GPT-4o, não sendo generalizáveis para outras ferramentas ou LLMs neste estágio da pesquisa.

Outra limitação relevante diz respeito ao escopo dos *CS* considerados. O conjunto analisado foi deliberadamente selecionado dentre *CS* que apresentam baixa ou nenhuma cobertura por ferramentas tradicionais, o que influencia diretamente os resultados observados no *baseline*. Esse recorte metodológico é coerente com os objetivos exploratórios do estudo, mas implica que os achados não representam o desempenho global das ferramentas avaliadas em relação ao catálogo completo de Fowler.

No caso dos experimentos com LLMs, destacam-se limitações associadas à variabilidade das respostas e à dependência do protocolo de *prompting*. Apesar da adoção de um *prompt* estruturado e documentado, observou-se a necessidade de etapas de pós-processamento para padronizar as saídas, bem como variações no nível de detalhamento das justificativas fornecidas pelo modelo. Esses fatores introduzem desafios adicionais à reprodutibilidade estrita e à análise quantitativa direta dos resultados.

Adicionalmente, os LLMs operam com acesso restrito ao contexto global dos projetos analisados, limitado ao conteúdo fornecido como entrada. A ausência de informações históricas, métricas evolutivas e relações dinâmicas entre módulos pode afetar a identificação de determinados *CS*, especialmente aqueles que dependem de uma visão longitudinal ou arquitetural mais abrangente do projeto.

Foram também observadas limitações de natureza operacional associadas ao uso do GPT-4o durante a execução dos experimentos preliminares. Em particular, identificou-se a exis-

tência de restrições no consumo de *tokens* ao longo de janelas temporais curtas, o que pode comprometer a continuidade de tarefas que demandam múltiplas análises de código consecutivas. O uso intensivo do modelo em períodos concentrados mostrou-se potencialmente limitante, mesmo em ambientes de acesso *premium*, restringindo a escalabilidade imediata do processo de análise automatizada. Ressalta-se que essa observação refere-se especificamente ao GPT-4o no contexto experimental adotado, não sendo possível generalizar essa limitação para outros modelos de linguagem sem avaliações adicionais.

Observou-se também a ocorrência pontual de alucinações, manifestadas, por exemplo, na indicação de *CS* fora do escopo previamente definido ou na geração de saídas que não aderem estritamente ao formato especificado no *prompt* de entrada. Esses comportamentos tendem a se intensificar em cenários que envolvem grande volume de análises consecutivas, reforçando a necessidade de protocolos de *prompting* rigorosos, etapas de validação posteriores e mecanismos de controle para assegurar a consistência dos resultados.

Adicionalmente, foram identificadas limitações operacionais relacionadas ao tamanho de algumas classes analisadas. Em casos pontuais, determinadas classes apresentavam um volume de código (LOC) superior à capacidade de processamento do modelo em uma única entrada, considerando as restrições de *tokens* do GPT-4o. Nesses cenários, foi necessário realizar um pré-processamento adicional, segmentando o código-fonte em partes menores para viabilizar sua análise pelo modelo.

Ressalta-se que essa situação ocorreu em um número reduzido de casos, sendo que a ampla maioria das classes analisadas possuía tamanho compatível com a inserção integral no *prompt* de entrada. O procedimento adotado foi conduzido de forma controlada e consistente, permitindo a continuidade da análise, mas introduzindo uma etapa adicional no fluxo experimental. Essa limitação reforça desafios práticos associados ao uso de LLMs em cenários reais, especialmente em sistemas com classes de grande porte, e será considerada nas etapas subsequentes do estudo.

Por fim, a análise estatística apresentada neste capítulo é de natureza descritiva e exploratória. A presença de *outliers* severos no *dataset*, bem como a heterogeneidade dos projetos analisados, impõe cautela na interpretação de métricas agregadas e reforça a necessidade de análises estratificadas e de métricas robustas, que serão aprofundadas nas próximas fases da pesquisa.

Em suma, as limitações aqui discutidas são compatíveis com o estágio atual da investigação e reforçam a importância da continuidade do protocolo experimental. Os resultados preliminares devem, portanto, ser compreendidos como evidências iniciais que orientam refinamentos metodológicos e fundamentam a ampliação do estudo, sem pretensão de estabelecer conclusões finais nesta etapa da proposta de tese. A seguir, a Tabela 5.10 mostra um resumo das limitações identificadas.

Tabela 5.10: Síntese das limitações identificadas nos resultados preliminares

Categoria	**Descrição da Limitação**
Escala Experimental	Análise restrita a um recorte inicial do protocolo, considerando apenas uma ferramenta tradicional e um LLM, preliminarmente.
Escopo de *CS*	Conjunto de *CS* deliberadamente limitado àqueles com baixa cobertura por ferramentas tradicionais, não representando o catálogo completo de Fowler.
Dependência de *Prompting*	Variabilidade nas respostas do LLM associada ao protocolo, exigindo etapas de pós-processamento para padronização das saídas.
Contexto Limitado	Análise restrita ao código fornecido como entrada, sem acesso a histórico evolutivo, métricas temporais ou relações dinâmicas entre módulos.
Restrições Operacionais do LLM	Limitações relacionadas ao consumo de *tokens* em janelas temporais curtas, impactando a execução de análises consecutivas em larga escala.
Classes de Grande Porte	Necessidade pontual de segmentação de classes com elevado número de LOC para viabilizar a análise pelo modelo, introduzindo etapa adicional de pré-processamento.
Alucinações	Ocorrência pontual de detecções fora do escopo definido ou de saídas em formato divergente do especificado no *prompt*.
Análise Estatística	Resultados de natureza descritiva e exploratória, com presença de *outliers* severos e de alta heterogeneidade entre projetos.

Com o objetivo de simplificar a visualização das principais limitações discutidas ao longo desta seção, a Tabela 5.10 apresenta um quadro-resumo das restrições metodológicas, opera-

cionais e analíticas identificadas nos resultados preliminares. A ordem das limitações apresentadas segue a progressão adotada na discussão textual desta seção, de modo a facilitar a correspondência entre a análise detalhada e um resumo visual.

5.7 Encaminhamento para a Fase Final do Estudo

Os resultados preliminares apresentados neste capítulo cumprem o papel de validar empiricamente a viabilidade do protocolo experimental proposto e de fornecer evidências iniciais que orientam a continuidade da investigação. A partir desses achados, tornam-se mais claros os encaminhamentos necessários para a consolidação da proposta de tese, bem como os ajustes metodológicos a serem incorporados nas etapas subsequentes.

Em relação à **RQ1**, que investiga a capacidade de ferramentas tradicionais de análise estática na detecção de *CS* em projetos desenvolvidos em TypeScript, os resultados obtidos com o SonarQube evidenciam lacunas relevantes no escopo analisado. Esses achados reforçam a necessidade de ampliar a investigação para outras ferramentas tradicionais, de modo a avaliar sistematicamente sua cobertura, limitações e comportamento diante de diferentes tipos de *CS*, conforme delineado na metodologia.

No que se refere à **RQ2**, voltada à avaliação da capacidade de LLMs na identificação de *CS*, os experimentos iniciais com o GPT-4o demonstram o potencial desses modelos para detectar *CS* não capturados por ferramentas tradicionais, especialmente aqueles que demandam maior contextualização semântica e análise estrutural do código-fonte. Esses resultados justificam a ampliação do estudo para múltiplos LLMs, contemplando diferentes arquiteturas e configurações, com o propósito de possibilitar uma análise comparativa mais abrangente, controlada e reprodutível.

A análise comparativa preliminar conduzida neste capítulo fornece subsídios iniciais para a investigação da **RQ3**, que aborda a complementaridade entre ferramentas tradicionais e LLMs. Os padrões de divergência e sobreposição observados indicam que essas abordagens desempenham papéis distintos e, potencialmente, complementares no processo de detecção de *CS*. Essa hipótese será aprofundada nas próximas fases do estudo por meio da incorporação de métricas quantitativas, de análises estatísticas mais robustas e de avaliações estratificadas por tipo de *CS* e por porte de projeto.

Adicionalmente, os resultados preliminares evidenciam a necessidade de aprofundar aspectos metodológicos relacionados à padronização das saídas dos LLMs, à mitigação de variações induzidas pelo *prompting* e à definição de critérios objetivos para avaliar a precisão, a cobertura e a consistência. Esses elementos são centrais para responder de forma sistemática à **RQ4**, que aborda a confiabilidade e a aplicabilidade prática de abordagens baseadas em LLMs no contexto da ES.

Dessa forma, a fase final do estudo concentrar-se-á na ampliação sistemática do *dataset*, na execução integral do protocolo experimental com múltiplas ferramentas tradicionais e diferentes modelos de linguagem, na consolidação das métricas de avaliação definidas e na realização de análises comparativas estatisticamente fundamentadas. A condução dessas etapas permitirá responder de forma integrada às questões de pesquisa propostas e sustentar, de maneira empírica e metodologicamente consistente, as conclusões finais desta tese.

Bibliografia

[1] Google AI. Gemini api - modelos. https://ai.google.dev/gemini-api/docs/models/gemini. Acesso em: 20/11/2025.

[2] Meta AI. Meta ai - home. https://ai.meta.com/. Acesso em: 20/11/2025.

[3] Mistral AI. Le chat. https://mistral.ai/products/le-chat. Acesso em: 20/11/2025.

[4] Microsoft Research AI4Science and Microsoft Azure Quantum. The impact of large language models on scientific discovery: a preliminary study using gpt-4, 2023.

[5] Humaid Al Naqbi, Zied Bahroun, and Vian Ahmed. Enhancing work productivity through generative artificial intelligence: A comprehensive literature review. *Sustainability*, 16(3), 2024.

[6] Danyllo Albuquerque, Everton Guimarães, Graziela Tonin, Pilar Rodríguezs, Mirko Perkusich, Hyggo Almeida, Angelo Perkusich, and Ferdinandy Chagas. Managing technical debt using intelligent techniques-a systematic mapping study. *IEEE Transactions on Software Engineering*, 49(4):2202–2220, 2022.

[7] Danyllo Albuquerque, S Everton Guimarães, Mirko Perkusich, Thiago Rique, Felipe Cunha, Hyggo Almeida, and Angelo Perkusich. On the assessment of interactive detection of code smells in practice: A controlled experiment. *IEEE Access*, 2023.

[8] Anthropic. Claude - interface conversacional. https://claude.ai/. Acesso em: 20/11/2025.

[9] Anthropic. Claude api. https://claude.com/platform/api. Acesso em: 20/11/2025.

[10] Anthropic. Claude overview. `https://claude.com/platform/api`. Acesso em: 20/11/2025.

[11] Anthropic. Site oficial. `https://www.anthropic.com/news/claude-sonnet-4-5`. Acesso em: 20/11/2025.

[12] Nils Baumgartner, Padma Iyenghar, Timo Schoemaker, and Elke Pulvermüller. Ai-driven refactoring: A pipeline for identifying and correcting data clumps in git repositories. *Electronics (Switzerland)*, 13(9), 2024.

[13] Justus Bogner and Manuel Merkel. To type or not to type? a systematic comparison of the software quality of javascript and typescript applications on github. In *Proceedings of the 19th International Conference on Mining Software Repositories*, pages 658–669, 2022.

[14] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[15] Google Cloud. Gemini no vertex ai. `https://cloud.google.com/vertex-ai/generative-ai/docs/model-reference/gemini`. Acesso em: 20/11/2025.

[16] GitHub Copilot. Site oficial. `https://github.com/features/copilot`. Acesso em: 20/11/2025.

[17] Di Cui, Jiaqi Wang, Qiangqiang Wang, Peng Ji, Minglang Qiao, Yutong Zhao, Jingzhao Hu, Luqiao Wang, and Qingshan Li. Three heads are better than one: Suggesting move method refactoring opportunities with inter-class code entity dependency enhanced hybrid hypergraph neural network. *Proceedings - 2024 39th ACM/IEEE International Conference on Automated Software Engineering, ASE 2024*, page 745 – 757, 2024.

[18] Di Cui, Qiangqiang Wang, Yutong Zhao, Jiaqi Wang, Minjie Wei, Jingzhao Hu, Luqiao Wang, and Qingshan Li. One-to-one or one-to-many? suggesting extract class refactoring opportunities with intra-class dependency hypergraph neural network. *ISSTA*

2024 - Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis, page 1529 – 1540, 2024.

[19] Arghavan Moradi Dakhel, Vahid Majdinasab, Amin Nikanjam, Foutse Khomh, Michel C Desmarais, and Zhen Ming Jack Jiang. Github copilot ai pair programmer: Asset or liability? *Journal of Systems and Software*, 203:111734, 2023.

[20] Elder Vicente de Paulo Sobrinho, Andrea De Lucia, and Marcelo de Almeida Maia. A systematic literature review on bad smells–5 w's: which, when, what, who, where. *IEEE Transactions on Software Engineering*, 47(1):17–66, 2018.

[21] Kayla DePalma, Izabel Miminoshvili, Chiara Henselder, Kate Moss, and Eman Abdullah AlOmar. Exploring chatgpt's code refactoring capabilities: An empirical study. *Expert Systems with Applications*, 249:123602, 2024.

[22] GitHub Docs. About github copilot for individuals. `https://docs.github.com/en/copilot/overview-of-github-copilot/about-github-copilot-for-individuals`. Acesso em: 20/11/2025.

[23] GitHub Docs. Getting started with github copilot. `https://docs.github.com/pt/copilot/getting-started-with-github-copilot`. Acesso em: 20/11/2025.

[24] Embold. Documentação oficial. `https://docs.embold.io/`. Acesso em: 20/11/2025.

[25] Embold. Site oficial. `https://embold.io/`. Acesso em: 20/11/2025.

[26] Interface Embold. Site oficial. `https://marketplace.visualstudio.com/items?itemName=Embold-VSCode.Embold`. Acesso em: 20/11/2025.

[27] ESLint. Playground de testes online oficial. `https://eslint.org/play/`. Acesso em: 20/11/2025.

[28] ESLint. Site oficial. `https://eslint.org/`. Acesso em: 20/11/2025.

[29] Ethereum. Site oficial. `https://ethereum.org/pt-br/`. Acesso em: 20/11/2025.

[30] Hugging Face. Site oficial. `https://huggingface.co/`. Acesso em: 20/11/2025.

[31] Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M. Zhang. Large language models for software engineering: Survey and open problems. In *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)*, pages 31–53, 2023.

[32] Martin Fowler. *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 2019.

[33] Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don-Refactoring Roberts. Improving the design of existing code. *Addisonâ^'Wesley*, 6, 1999.

[34] Google. Google acadêmico. `https://scholar.google.com/?hl=pt-BR`. Acesso em: 20/11/2025.

[35] Google. Site oficial. `https://gemini.google/br/about/?hl=pt-BR`. Acesso em: 20/11/2025.

[36] Max Hort, Anastasiia Grishina, and Leon Moonen. An exploratory literature study on sharing and energy use of language models for source code. *International Symposium on Empirical Software Engineering and Measurement*, 2023.

[37] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. Large language models for software engineering: A systematic literature review. *ACM Trans. Softw. Eng. Methodol.*, 33(8), December 2024.

[38] Saki Imai. Is github copilot a substitute for human pair-programming? an empirical study. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*, pages 319–321, 2022.

[39] Marcel Jerzyk and Lech Madeyski. Code smells: A comprehensive online catalog and taxonomy. In *Developments in Information and Knowledge Management Systems for Business Applications: Volume 7*, pages 543–576. Springer, 2023.

[40] JetBrains. Site oficial. `https://www.jetbrains.com/idea/`. Acesso em: 20/11/2025.

[41] Jinan Jiang, Zihao Li, Haoran Qin, Muhui Jiang, Xiapu Luo, Xiaoming Wu, Haoyu Wang, Yutian Tang, Chenxiong Qian, and Ting Chen. Unearthing gas-wasting code smells in smart contracts with large language models. *IEEE Transactions on Software Engineering*, 2024.

[42] Kaggle. Site oficial. `https://www.kaggle.com/`. Acesso em: 20/11/2025.

[43] Pranjal Kumar. Large language models (llms): survey, technical frameworks, and future challenges. *Artificial Intelligence Review*, 57(9), 2024.

[44] Barbara Liskov. Keynote address - data abstraction and hierarchy. In *Addendum to the Proceedings on Object-Oriented Programming Systems, Languages and Applications (Addendum)*, OOPSLA '87, page 17–34, New York, NY, USA, 1987. Association for Computing Machinery.

[45] Meta. Llama no whatsapp. `https://faq.whatsapp.com/666767195111959/?helpref=uf_share`. Acesso em: 20/11/2025.

[46] Meta. Llama overview. `https://www.llama.com/llama-protections/`. Acesso em: 20/11/2025.

[47] Meta. Site oficial. `https://www.llama.com/`. Acesso em: 20/11/2025.

[48] Mistral. Modelos mistral. `https://mistral.ai/models`. Acesso em: 20/11/2025.

[49] Mistral. Site oficial. `https://mistral.ai`. Acesso em: 20/11/2025.

[50] Naouel Moha, Yann-Gaël Guéhéneuc, Laurence Duchien, and Anne-Francoise Le Meur. Decor: A method for the specification and detection of code and design smells. *IEEE Transactions on Software Engineering*, 36(1):20–36, 2009.

[51] Tobias Münch and Rainer Roosmann. Transfer, measure and extend maintainability metrics for web component based applications to achieve higher quality. In *WEBIST*, pages 105–112, 2022.

[52] Willian Nalepa Oizumi, Alessandro Fabricio Garcia, Thelma Elita Colanzi, Manuele Ferreira, and Arndt von Staa. When code-anomaly agglomerations represent architectural problems? an exploratory study. In *2014 Brazilian symposium on software engineering*, pages 91–100. IEEE, 2014.

[53] Rocco Oliveto, Malcom Gethers, Gabriele Bavota, Denys Poshyvanyk, and Andrea De Lucia. Identifying method friendships to remove the feature envy bad smell (nier track). In *Proceedings of the 33rd International Conference on Software Engineering*, pages 820–823, 2011.

[54] Ollama. Modelos. `https://ollama.com/search`. Acesso em: 20/11/2025.

[55] Ollama. Projeto ollama no github. `https://github.com/ollama/ollama`. Acesso em: 20/11/2025.

[56] Ollama. Site oficial. `https://ollama.com/`. Acesso em: 20/11/2025.

[57] OpenAI. Chatgpt overview. `https://chatgpt.com/pt-BR/overview`. Acesso em: 20/11/2025.

[58] OpenAI. Chatgpt — preços. `https://openai.com/chatgpt/pricing`. Acesso em: 20/11/2025.

[59] OpenAI. Site oficial. `https://openai.com/chatgpt`. Acesso em: 20/11/2025.

[60] Yousuke Ota. Playground eslint para regência. `https://ota-meshi.github.io/eslint-plugin-regexp/playground/`. Acesso em: 20/11/2025.

[61] Fabio Palomba, Gabriele Bavota, Massimiliano Di Penta, Fausto Fasano, Rocco Oliveto, and Andrea De Lucia. On the diffuseness and the impact on maintainability of code smells: a large scale empirical investigation. In *Proceedings of the 40th International Conference on Software Engineering*, pages 482–482, 2018.

[62] Dorin Pomian, Abhiram Bellur, Malinda Dilhara, Zarina Kurbatova, Egor Bogomolov, Timofey Bryksin, and Danny Dig. Next-generation refactoring: Combining llm insights and ide capabilities for extract method. *Proceedings - 2024 IEEE International*

Conference on Software Maintenance and Evolution, ICSME 2024, page 275 – 287, 2024.

[63] Dorin Pomian, Abhiram Bellur, Malinda Dilhara, Zarina Kurbatova, Egor Bogomolov, Andrey Sokolov, Timofey Bryksin, and Danny Dig. Em-assist: Safe automated extract-method refactoring with llms. *FSE Companion - Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, page 582 – 586, 2024.

[64] Refactoring.Guru. Refactoring.guru website. `https://refactoring.guru/pt-br/`. Acesso em: 20/11/2025.

[65] Amir Saboury, Pooya Musavi, Foutse Khomh, and Giulio Antoniol. An empirical study of code smells in javascript projects. In *2017 IEEE 24th international conference on software analysis, evolution and reengineering (SANER)*, pages 294–305. IEEE, 2017.

[66] Tushar Sharma and Diomidis Spinellis. A survey on software smells. *Journal of Systems and Software*, 138:158–173, 2018.

[67] Mohammed Latif Siddiq, Shafayat H. Majumder, Maisha R. Mim, Sourov Jajodia, and Joanna C. S. Santos. An empirical study of code smells in transformer-based code generation techniques. *Proceedings - 2022 IEEE 22nd International Working Conference on Source Code Analysis and Manipulation, SCAM 2022*, page 71 – 82, 2022.

[68] Luciana Lourdes Silva, Janio Rosa da Silva, João Eduardo Montandon, Marcus Andrade, and Marco Tulio Valente. Detecting code smells using chatgpt: Initial insights. In *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 400–406, 2024.

[69] Uanderson Silva, Matheus Andrade, José Cruz, Andresa Silva, Augusto Sampaio, and Kiev Gama. Micro frontend architecture for robotic systems: a systematic approach with design rationale. In *2025 IEEE/ACM 7th International Workshop on Robotics Software Engineering (RoSE)*, pages 1–8. IEEE, 2025.

[70] Dominik Sobania, Martin Briesch, and Franz Rothlauf. Choose your programming copilot: A comparison of the program synthesis performance of github copilot and

genetic programming. In *Proceedings of the genetic and evolutionary computation conference*, pages 1019–1027, 2022.

[71] Solidity. Site oficial. `https://www.soliditylang.org/`. Acesso em: 20/11/2025.

[72] Sonar. Sonarqube cloud. `https://www.sonarsource.com/products/sonarcloud/`. Acesso em: 20/11/2025.

[73] SonarQube. Documentação oficial. `https://docs.sonarsource.com/sonarqube-server/latest/`. Acesso em: 20/11/2025.

[74] Wei Song, Lu Gan, and Tie Bao. Software defect prediction via generative adversarial networks and pre-trained model. *International Journal of Advanced Computer Science and Applications*, 15(3):1196 – 1209, 2024.

[75] Zoltán Szabó and Vilmos Bilicki. A new approach to web application security: Utilizing gpt language models for source code inspection. *Future Internet*, 15(10), 2023.

[76] Seyyed Ehsan Salamati Taba, Foutse Khomh, Ying Zou, Ahmed E Hassan, and Meiyappan Nagappan. Predicting bugs using antipatterns. In *2013 IEEE International Conference on Software Maintenance*, pages 270–279. IEEE, 2013.

[77] Kristín Fjóla Tómasdóttir, Mauricio Aniche, and Arie Van Deursen. The adoption of javascript linters in practice: A case study on eslint. *IEEE Transactions on Software Engineering*, 46(8):863–891, 2018.

[78] Typescript-eslint. Site oficial. `https://typescript-eslint.io/`. Acesso em: 20/11/2025.

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

[80] Claes Wohlin. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, pages 1–10, 2014.

[81] Di Wu, Fangwen Mu, Lin Shi, Zhaoqiang Guo, Kui Liu, Weiguang Zhuang, Yuqi Zhong, and Li Zhang. ismell: Assembling llms with expert toolsets for code smell detection and refactoring. *Proceedings - 2024 39th ACM/IEEE International Conference on Automated Software Engineering, ASE 2024*, page 1345 – 1357, 2024.

[82] Burak Yetistiren, Isik Ozsoy, and Eray Tuzun. Assessing the quality of github copilot's code generation. In *Proceedings of the 18th International Conference on Predictive Models and Data Analytics in Software Engineering*, pages 62–71, 2022.

[83] Fengji Zhang, Jin Liu, Yao Wan, Xiao Yu, Xiao Liu, and Jacky Keung. Diverse title generation for stack overflow posts with multiple-sampling-enhanced transforme. *Journal of Systems and Software*, 200, 2023.

[84] Yang Zhang, Yanlei Li, Grant Meredith, Kun Zheng, and Xiaobin Li. Move method refactoring recommendation based on deep learning and llm-generated information. *Information Sciences*, 697, 2025.

Apêndice A

Estudo de Mapeamento Sistemático – Investigando Detecção de *CS* por LLMs

Nas últimas décadas, a ES tem aprimorado o uso de técnicas e ferramentas para apoiar atividades de manutenção, evolução e melhoria contínua da qualidade do código, principalmente no tratamento de *CS* e na mitigação de dívidas técnicas [32, 66]. Mais recentemente, com a popularização do uso de IA na ES, especialmente de LLMs, novas abordagens têm sido exploradas para apoiar essas atividades [36, 31, 37].

Com o avanço recente dos LLMs, esse movimento adquire nova dimensão: esses modelos passaram a ser empregados em tarefas como detecção e refatoração de *CS*, explorando sua capacidade de reconhecer padrões estruturais, apoiar decisões de manutenção e interpretar texto, inclusive código-fonte [31, 37].

Apesar desse potencial, a literatura recente [19, 14, 70, 63, 81] evidencia limitações importantes: (i) inconsistências semânticas; (ii) alucinações; (iii) sensibilidade ao *prompt* de entrada; (iv) restrições de contexto; e (v) forte heterogeneidade metodológica. Isto ainda dificulta a consolidação dessas abordagens. Como consequência, permanece pouco evidente qual é o papel efetivo dos LLMs na detecção e refatoração de *CS*, quais tipos de *smells* têm sido explorados, quais evidências empíricas sustentam seu uso e quais lacunas permanecem em aberto na literatura.

Diante desse panorama, este apêndice apresenta um Estudo de Mapeamento Sistemático (EMS) cujo objetivo é organizar e sintetizar o estado da arte sobre o uso de LLMs na detecção e na refatoração de *CS*. Neste trabalho, o EMS desempenha dois objetivos complementares:

I. Oferecer uma visão estruturada das abordagens de detecção propostas na literatura, dos tipos de *smells* abordados, das evidências empíricas reportadas e das limitações identificadas;

II. Fundamentar as escolhas metodológicas dos capítulos subsequentes, ao destacar a necessidade de estudos comparativos rigorosos entre LLMs e ferramentas tradicionais, sobretudo em linguagens contemporâneas como TypeScript, cuja cobertura em pesquisas sobre detecção de *CS* permanece limitada [68].

As questões de pesquisa apresentadas na Seção A.1 estruturam o protocolo de busca, seleção, extração e análise dos estudos que compõem este mapeamento.

A.1 Questões de Pesquisa do EMS (RQs)

A definição das Questões de Pesquisa do Estudo de Mapeamento Sistemático (RQs) constitui o núcleo metodológico deste estudo. Essas questões foram formuladas para caracterizar, de forma sistemática e abrangente, o papel dos LLMs na detecção de *CS*, bem como identificar desafios, evidências empíricas e oportunidades emergentes no campo. As RQs cumprem duas funções complementares: (i) sistematizar o estado atual da literatura, organizando seus principais eixos temáticos; e (ii) identificar lacunas que fundamentam a investigação empírica desenvolvida nesta tese.

As 5 (cinco) questões de pesquisa definidas para o EMS são apresentadas a seguir:

RQ1. *Quais abordagens têm sido empregadas na literatura para aplicar LLMs à detecção de CS?*

A primeira questão (**RQ1**) buscou identificar e caracterizar as abordagens propostas nos estudos analisados, descrevendo como LLMs têm sido integrados a *pipelines* de análise, técnicas híbridas, arquiteturas especializadas ou mecanismos auxiliares para apoiar a detecção de *CS*. O objetivo foi compreender o escopo, as estratégias e o grau de inovação metodológica das soluções avaliadas.

RQ2. *Quais tipos de CS têm sido investigados em conjunto com LLMs e como os estudos caracterizam seus desafios específicos?*

A segunda questão (**RQ2**) teve por finalidade mapear os diferentes tipos de *CS* tratados na literatura e analisar como cada estudo descreveu os desafios estruturais, semânticos ou contextuais associados à detecção desses tipos. Essa análise permitiu identificar padrões de escolha dos pesquisadores e lacunas em *smells* pouco explorados.

RQ3. *Quais evidências empíricas são reportadas quanto ao desempenho de LLMs na detecção de CS?*

A terceira questão (**RQ3**) sintetizou os resultados empíricos disponíveis, incluindo métricas de desempenho, experimentos controlados, análises comparativas e avaliações qualitativas. Seu propósito foi compreender o grau de efetividade das abordagens baseadas em LLMs e a robustez metodológica das evidências produzidas.

RQ4. *Quais limitações técnicas, conceituais e metodológicas são identificadas nos estudos que utilizam LLMs para detecção de CS?*

A quarta questão (**RQ4**) buscou identificar fragilidades, riscos, vieses e restrições apontados pela literatura, abrangendo desde limitações inerentes aos LLMs (*e.g.* alucinações e sensibilidade ao *prompt* de entrada) até limitações experimentais, conceituais e práticas. Essa síntese foi essencial para compreender os obstáculos que ainda impedem o uso consolidado dessas técnicas.

RQ5. *Quais tendências e oportunidades futuras emergem do uso de LLMs em tarefas de detecção de CS?*

Por fim, a quinta questão (**RQ5**) examina direções futuras apontadas pelos estudos, incluindo oportunidades de pesquisa, inovações tecnológicas, integração com ferramentas existentes e lacunas conceituais ou empíricas que demandam investigação adicional. Seu objetivo é delinear como o campo tem evoluído e quais caminhos se mostram mais promissores.

A.2 Configuração do Estudo

A metodologia adotada segue as diretrizes de Wohlin [80], contemplando: (i) definição das RQs; (ii) elaboração da estratégia de busca; (iii) aplicação de critérios de inclusão e exclusão;

(iv) execução da busca em cadeia (*forward snowballing*); e (v) extração e síntese dos dados. As Subseções A.2.1, A.2.2, A.2.3 e A.2.4 detalham cada etapa.

Para mitigar vieses na etapa de seleção dos estudos, o processo de triagem foi conduzido por meio de dupla revisão independente, em conformidade com o procedimento descrito no Capítulo 1. Cada artigo foi avaliado por dois pesquisadores de forma autônoma, e as divergências foram resolvidas por consenso, garantindo maior rigor metodológico e consistência conceitual.

A.2.1 Estratégia de Busca

A estratégia de busca adotada neste EMS foi conduzida exclusivamente na base Scopus, selecionada por sua ampla cobertura de publicações indexadas e por sua aderência às diretrizes metodológicas recomendadas por Wohlin [80].

A construção da *string* de busca seguiu um processo iterativo, no qual termos representativos dos conceitos centrais deste estudo (LLMs, *Code Smells*, Engenharia de Software e técnicas de detecção) foram refinados progressivamente com base em buscas-piloto e na literatura especializada.

A Tabela A.1 apresenta a *string* final utilizada na consulta à base Scopus.

Tabela A.1: *String* completa utilizada na busca

```
("Large Language Model" OR "LLM" OR "GPT" OR
"Generative Pre-trained Transformer") AND
("Code Smell" OR "Bad Smell" OR "Software Defect") AND
("Software Engineering" OR "Software Development") AND
("Detection" OR "Identification" OR "Detection
Technique") AND
LIMIT-TO(LANGUAGE, "English")
```

A *string* foi composta por blocos temáticos conectados pelo operador *booleano* AND, assegurando que apenas estudos relacionados simultaneamente aos quatro eixos conceituais fossem retornados: (i) LLMs, (ii) *Code Smells*, (iii) Engenharia de Software e (iv) Técnicas de Detecção.

Cada bloco interno utiliza o operador *booleano* OR para agrupar sinônimos ou variações terminológicas de um mesmo conceito, ampliando a cobertura da busca sem comprometer a precisão. Os termos incluídos entre aspas (" ") foram deliberadamente mantidos dessa forma para garantir que o mecanismo de busca interpretasse cada expressão como uma unidade lexical completa, assegurando correspondência exata com o termo composto (e não com partes isoladas da expressão). Já a combinação entre os blocos ocorre exclusivamente pelo conector AND, garantindo que todos os eixos conceituais fossem atendidos de forma simultânea.

Para fins de transparência e replicabilidade, a Tabela A.2 detalha a função conceitual de cada bloco da *string*. Para facilitar a organização, cada agrupamento temático foi identificado pelos rótulos Str1 – Str5, conforme descrito abaixo.

Tabela A.2: Descrição dos componentes da *string* de busca

ID	**Descrição**
Str1	Termos relacionados a LLMs.
Str2	Termos relacionados a Code Smells.
Str3	Termos relacionados a Engenharia de Software.
Str4	Termos relacionados à Detecção e/ou Identificação.
Str5	Restrição de idioma para Inglês.

A busca foi submetida aos critérios de inclusão e exclusão definidos no protocolo metodológico, conforme apresentado na Subseção A.2.2. A consulta foi realizada em 16 de fevereiro de 2025, retornando 202 publicações potenciais no intervalo de 2020 – 2025. Com o intuito de garantir transparência e replicabilidade, a planilha de extração e os demais artefatos utilizados neste EMS foram disponibilizados publicamente, conforme mostrado no Apêndice C.

A.2.2 Seleção dos Estudos

Concluída a etapa de busca, procedeu-se ao processo de seleção dos estudos, cuja finalidade foi garantir que apenas publicações efetivamente relevantes, metodologicamente rigorosas e alinhadas ao escopo deste EMS fossem incluídas na análise final. A seleção seguiu estritamente as diretrizes de Wohlin [80] e foi conduzida em múltiplas etapas sucessivas: (i)

remoção de duplicatas; (ii) leitura de títulos e resumos; (iii) leitura do texto completo; e (iv) avaliação por dupla revisão independente.

Inicialmente, todos os registros retornados pela busca automática foram importados em uma planilha de gerenciamento, na qual as duplicações foram eliminadas. Em seguida, realizou-se uma triagem baseada em títulos e resumos, aplicando-se os critérios de inclusão e exclusão estabelecidos no protocolo metodológico. Apenas os estudos que atenderam simultaneamente aos critérios de escopo — explicitamente abordar LLMs e manter relação direta com a detecção de *CS* — avançaram para a fase de leitura completa.

A avaliação do texto completo de cada estudo foi realizada por dois pesquisadores de forma independente, conforme o procedimento descrito no Capítulo 1. Divergências na classificação foram registradas e, posteriormente, resolvidas por consenso, assegurando rigor metodológico e mitigação de vieses individuais. Essa abordagem colaborativa permitiu verificar a aderência metodológica, a qualidade empírica e a relevância temática, reduzindo o risco de inclusão indevida de publicações inadequadas.

Os Critérios de Inclusão (CI) e de Exclusão (CE) utilizados encontram-se reunidos nas Tabelas A.3 e A.4, respectivamente.

Tabela A.3: Critérios de Inclusão (CI)

ID	**Critério**
CI1	Publicações revisadas por pares (periódicos ou conferências).
CI2	Estudos que abordam LLMs aplicados à detecção ou identificação de *CS*.
CI3	Publicações no intervalo de 2020 a 2025.
CI4	Estudos que apresentem fundamentação empírica (quantitativa ou qualitativa).

Tabela A.4: Critérios de Exclusão (CE)

ID	**Critério**
CE1	Documentos duplicados.
CE2	Materiais não revisados por pares (pré-prints, teses, relatórios técnicos).
CE3	Capítulos de livros ou obras não periódicas.
CE4	Publicações não disponíveis em inglês.
CE5	Estudos que não tratam simultaneamente de LLMs **e** de *CS*.
CE6	Publicações fora do domínio da Ciência da Computação.

Os critérios de inclusão estabelecem os requisitos mínimos para garantir a pertinência e consistência dos estudos analisados, enquanto os critérios de exclusão visam eliminar trabalhos que, por limitações de escopo, forma de publicação ou ausência de rigor, poderiam comprometer a qualidade do EMS. Destaca-se, em particular, o critério CE5, responsável por excluir estudos que tratam apenas de LLMs ou apenas de *CS*, mas não de sua interseção, requisito central deste mapeamento.

Após a aplicação rigorosa desses critérios e a consolidação das avaliações dos revisores, o conjunto final foi composto por 8 (oito) estudos publicados entre 2024 e 2025. Essa concentração temporal reforça tanto a atualidade do tema quanto seu caráter emergente, evidenciando a pertinência do EMS para estruturar um campo de pesquisa em formação.

A etapa posterior consistiu no uso da técnica de *forward snowballing*, descrita na Subseção A.2.3, com o objetivo de identificar possíveis estudos adicionais não capturados pela busca inicial.

A.2.3 Busca em Cadeia (*Snowballing Search*)

Após a definição do conjunto preliminar de estudos derivados da estratégia de busca principal, procedeu-se à aplicação da técnica de *forward snowballing search*, conforme recomendada por Wohlin [80]. Esse procedimento teve como objetivo identificar publicações adicionais que citassem os estudos previamente selecionados, ampliando a cobertura da revisão sem se limitar exclusivamente aos resultados da consulta direta à base Scopus.

A busca em cadeia foi realizada na plataforma Google Scholar [34], por sua capacidade de rastrear citações de forma abrangente e atualizada. Tanto os estudos primários quanto os

secundários identificados na etapa anterior serviram como conjunto inicial (*seed set*) para a primeira iteração do processo. Para assegurar a consistência metodológica, os mesmos critérios de inclusão e exclusão estabelecidos na Subseção A.2.2 foram rigorosamente aplicados aos estudos retornados nesta etapa.

O procedimento consistiu em examinar individualmente todas as publicações que citavam os artigos do conjunto inicial, verificando sua aderência ao escopo do EMS, especificamente quanto à presença simultânea dos seguintes elementos: (i) uso explícito de LLMs; (ii) relação direta com detecção ou refatoração de *CS*; e (iii) enquadramento no domínio da Engenharia de Software.

Apesar da amplitude das citações inspecionadas, nenhuma nova publicação atendeu aos critérios necessários à inclusão. Desse modo, o processo de *forward snowballing search* foi concluído sem a identificação de novos estudos relevantes, corroborando que o conjunto inicial, composto por 8 (oito) artigos publicados entre 2024 e 2025, já representava o corpo mais atual e pertinente de pesquisas sobre o tema.

A ausência de novos estudos nessa etapa reforça dois pontos: (a) a natureza emergente da área, ainda em consolidação, e (b) a adequação da estratégia de busca principal, capaz de recuperar a totalidade da literatura contemporânea diretamente relacionada ao uso de LLMs na detecção de *CS*.

A.2.4 Extração e Análise dos Dados

Após a definição do conjunto final de estudos incluídos no EMS, procedeu-se à etapa de extração sistemática de dados, conduzida com base em um protocolo previamente definido e alinhado às diretrizes metodológicas de Wohlin [80]. Esta etapa teve como finalidade organizar, estruturar e sintetizar as informações necessárias para responder às RQs estabelecidas na Seção A.1.

A extração foi realizada com base na leitura integral de cada um dos oito estudos selecionados. Para assegurar uniformidade e reduzir vieses interpretativos, utilizou-se uma planilha de extração padronizada, elaborada para registrar, de forma consistente, os seguintes elementos: metadados bibliográficos, objetivos do estudo, tipos de *CS* investigados, papel atribuído aos LLMs, métodos empregados, resultados empíricos reportados, limitações identificadas e contribuições principais.

Em conformidade com o procedimento recomendado por Albuquerque *et al.* [6], a extração foi realizada por, pelo menos, dois pesquisadores de maneira independente, garantindo avaliação cruzada e mitigação de vieses individuais. Divergências foram discutidas até o consenso, assegurando maior rigor e confiabilidade na consolidação dos dados.

A análise subsequente consistiu na categorização e síntese qualitativa , permitindo identificar padrões, convergências e lacunas entre os estudos, o que orientou a organização dos resultados apresentados na Seção A.3. A Tabela A.5 reúne os artigos selecionados, apresentados em ordem cronológica com seus principais metadados.

Tabela A.5: Resultado da extração de dados dos 8 estudos selecionados

ID	**Ano**	**Título**	**Autor(es)**	**Classificação**
P1	2024	*AI-Driven Refactoring: A Pipeline for Identifying and Correcting Data Clumps in Git Repositories*	Baumgartner *et al.* [12]	LLM, Detecção de *CS*, ES
P2	2024	*Three Heads Are Better Than One: Suggesting Move Method Refactoring Opportunities with Inter-class Code Entity Dependency Enhanced Hybrid Hypergraph Neural Network*	Cui *et al.* [17]	LLM, Detecção de *CS*, ES
P3	2024	*One-to-One or One-to-Many? Suggesting Extract Class Refactoring Opportunities via Hypergraph Neural Networks*	Cui *et al.* [18]	LLM, Detecção de *CS*, ES
P4	2024	*Unearthing Gas-Wasting Code Smells in Smart Contracts with LLMs*	Jiang *et al.* [41]	LLM, Detecção de *CS*, ES
P5	2024	*Next-Generation Refactoring: Combining LLM Insights and IDE Capabilities for Extract Method*	Pomian *et al.* [62]	LLM, Detecção de *CS*, ES
P6	2024	*EM-Assist: Safe Automated Extract-Method Refactoring with LLMs*	Pomian *et al.* [63]	LLM, Detecção de *CS*, ES
P7	2024	*iSMELL: Assembling LLMs with Expert Toolsets for Code Smell Detection and Refactoring*	Wu *et al.* [81]	LLM, Detecção de *CS*, ES
P8	2025	*Move Method Refactoring Recommendation with Deep Learning and LLM-Generated Information*	Zhang *et al.* [84]	LLM, Detecção de *CS*, ES

A.3 Resultados do EMS

Esta seção apresenta os resultados obtidos com a aplicação do protocolo metodológico descrito anteriormente. A análise seguiu uma abordagem orientada pelas RQs definidas na Seção A.1, de modo a reunir, de forma estruturada e crítica, o conhecimento disponível sobre o emprego de LLMs na detecção de *CS*.

Os resultados aqui descritos foram organizados em duas etapas complementares. A primeira (ver Subseção A.3.1) examina as características demográficas da literatura selecionada, permitindo compreender a distribuição temporal, os veículos de publicação e a diversidade de enfoques metodológicos presentes no conjunto final de estudos (P1 – P8).

A segunda etapa (ver Subseção A.3.2) apresenta, para cada RQ, uma interpretação analítica dos achados, destacando padrões recorrentes, divergências conceituais, limitações observadas e lacunas ainda não exploradas pela comunidade científica. Esse processo teve o objetivo não apenas de descrever o estado da arte, mas também de revelar tendências e direcionamentos que fundamentaram as decisões metodológicas desta tese.

A.3.1 Informações Demográficas dos Estudos

A análise demográfica do conjunto final de estudos (P1 – P8) permite caracterizar o estágio atual da pesquisa sobre o uso de LLMs para detecção de *CS*. Todos os estudos selecionados foram publicados entre 2024 e 2025, evidenciando o caráter emergente do tema e o crescente interesse da comunidade científica em explorar modelos generativos como instrumentos de apoio à manutenção de software.

Em termos temporais, observa-se uma forte concentração em 2024, ano no qual 7 (sete) dos 8 (oito) estudos selecionados foram publicados (P1 – P7), enquanto apenas um trabalho corresponde ao ano de 2025 (P8). Esse padrão indica que a aplicação de LLMs em tarefas de detecção de *CS* ganhou tração recentemente, impulsionada pelo amadurecimento dos LLMs e pelo aumento das investigações sobre seu potencial para análise de código.

Quanto aos canais de publicação, todos os estudos pertencem a veículos revisados por pares, predominantemente conferências e periódicos das áreas de ES e IA. Esse perfil reflete o alinhamento da pesquisa com as comunidades científicas consolidadas e com o debate contemporâneo sobre o uso de IA na qualidade de software.

A diversidade metodológica constitui outro aspecto relevante do conjunto analisado. Embora todos os estudos investiguem a interseção entre LLMs e a detecção de *CS*, eles o fazem a partir de diferentes perspectivas metodológicas . De modo geral, essas perspectivas podem ser organizadas em quatro categorias amplas, que refletem os principais tipos de abordagem empregados na literatura: (i) refatorações específicas; (ii) análise estrutural; (iii) investigações em domínios especializados; e (iv) arquiteturas híbridas.

Essas características demográficas contextualizam o conjunto de estudos analisados e subsidiam a interpretação crítica dos resultados discutidos na Subseção A.3.2.

A.3.2 Discussão dos Achados do EMS

A análise integrada das questões de pesquisa definidas neste EMS (**RQ1**, **RQ2**, **RQ3**, **RQ4** e **RQ5**) permite sistematizar o estado atual da literatura que investiga a aplicação de LLMs na detecção e na refatoração de *CS*. Os achados evidenciam um campo em expansão recente, marcado pela adoção predominante de arquiteturas híbridas, pela concentração em *CS* estruturalmente bem definidos e por evidências empíricas ainda fragmentadas.

Além disso, observa-se forte dependência de mecanismos auxiliares para garantir estabilidade e precisão, bem como a coexistência de limitações associadas às soluções propostas e de desafios estruturais inerentes ao próprio problema, aspectos que, em conjunto, caracterizam um estágio inicial de consolidação científica.

RQ1 – Abordagens Empregadas

A análise dos estudos selecionados indica que a aplicação de LLMs à detecção ou ao suporte à refatoração de *CS* ocorre predominantemente por meio de arquiteturas híbridas. Em nenhum dos trabalhos analisados os LLMs são empregados como detectores autônomos; ao contrário, sua atuação está sistematicamente integrada a mecanismos estruturais, estatísticos ou heurísticos, assumindo papéis complementares no processo de análise de código.

Uma primeira linha de abordagem recorre à combinação entre análise estática tradicional e LLMs, explorando verificações estruturais, árvores sintáticas abstratas (ASTs) e validações determinísticas como base para a detecção de *CS*. Nesses casos, os LLMs são utilizados principalmente para apoiar a interpretação semântica, a geração de sugestões de refatoração

ou a validação contextual dos resultados gerados por técnicas determinísticas (P1, P6).

Outra vertente recorrente fundamenta-se em modelos estruturais baseados em hipergráficos, tanto em nível interclasse quanto intraclasse. Esses trabalhos utilizam representações relacionais avançadas para modelar dependências entre entidades de código, enriquecendo tais estruturas com informações semânticas derivadas de LLMs, como descrições textuais ou *embeddings*. Essa integração visa ampliar a capacidade dos modelos de capturar relações conceituais que extrapolam métricas puramente sintáticas (P2, P3).

Também se observam abordagens baseadas em *pipelines* semiautomatizados, nos quais os LLMs são empregados em conjunto com validação humana explícita. Nesse âmbito, os LLMs atuam como mecanismos de apoio à identificação inicial de *CS*, enquanto a confirmação final e a aplicação das correções permanecem sob responsabilidade dos desenvolvedores, especialmente em domínios sensíveis, como contratos inteligentes em *blockchain* (P4).

Uma estratégia adicional envolve arquiteturas iterativas de *reprompting* e de ranqueamento, nas quais os LLMs são acionados em ciclos sucessivos de refinamento. Nessas abordagens, respostas intermediárias são filtradas, reordenadas e reavaliadas com base em critérios estruturais, visando reduzir instabilidades e melhorar a qualidade das recomendações de refatoração, como no caso de *Extract Method* (P5).

Por fim, alguns estudos exploram arquiteturas híbridas mais complexas, que combinam LLMs com *Mixture of Experts* (MoE) ou modelos de aprendizado profundo. Nesses cenários, especialistas heurísticos, ferramentas de análise estática e redes neurais especializadas cooperam com um LLM orquestrador ou utilizam *embeddings* gerados por LLMs como insumo semântico para modelos preditivos, ampliando a capacidade de recomendação em tarefas como *Move Method* (P7, P8).

Em suma, os resultados evidenciam que os LLMs desempenham predominantemente funções semânticas — como geração de descrições, produção de *embeddings*, refinamento iterativo e contextualização — enquanto a precisão estrutural e a confiabilidade das decisões dependem de mecanismos complementares. Esse padrão recorrente sugere que, no estado atual da literatura, os LLMs ainda não apresentam robustez suficiente para atuar isoladamente na detecção de *CS*, reforçando a necessidade de integrações entre LLMs e mecanismos complementares para alcançar resultados estáveis e reproduzíveis.

A Tabela A.6 oferece uma visão panorâmica das principais abordagens adotadas pelos

estudos analisados, relacionando cada trabalho ao tipo de arquitetura empregada.

Tabela A.6: Abordagens empregadas pelos estudos selecionados (RQ1)

ID	**Tipo de Abordagem**	**Descrição Sintética da Abordagem**
P1	*Pipeline* Híbrido + Análise Estática + LLM	Detecção e Correção de *Data Clumps* via ASTs, validações determinísticas e LLM.
P2	Hipergráficos Interclasse + Modelos Neurais + LLM	Representação relacional enriquecida com *embeddings* gerados por LLM para recomendar *Move Method*.
P3	Hipergráficos Intraclasse + Atributos Semânticos	Arquitetura baseada em HGNN com apoio semântico de LLM para recomendação de *Extract Class*.
P4	*Pipeline* Semiautomatizado + LLM + Validação Humana	Detecção de *GWCS* em contratos inteligentes com supervisão humana.
P5	*Reprompting* Iterativo + Ranqueamento + Filtros Estruturais	Arquitetura iterativa para recomendação de *Extract Method*.
P6	Análise Estática + Verificações Estruturais + LLM integrado à IDE	Extração segura de métodos com validação automatizada e apoio de LLM.
P7	*MoE* + LLM Orquestrador	Integração de especialistas, ferramentas estáticas e LLM para detecção e refatoração.
P8	Aprendizado Profundo Híbrido + LLM	Combinação de redes profundas e de informações semânticas geradas por LLM para recomendar *Move Method*.

RQ2 – Tipos de *CS* Investigados

A análise dos estudos selecionados indica que a investigação sobre *CS* no contexto do uso de LLMs permanece fortemente concentrada em anomalias estruturais de baixa granularidade, tipicamente associadas a problemas de coesão, acoplamento indevido ou excesso de responsabilidades. Esses *smells* são, em sua maioria, bem estabelecidos na literatura clássica de refatoração, o que reflete um recorte conservador do espaço de problemas explorado pelos

trabalhos analisados.

A Tabela A.7 sintetiza a ocorrência dos *CS* explicitamente investigados nos 8 (oito) estudos incluídos no EMS, permitindo visualizar a distribuição dos *smells* ao longo do conjunto analisado.

Tabela A.7: Matriz de ocorrência dos *CS* nos estudos selecionados (RQ2)

Code Smell	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**	**P8**
Data Clumps	X							
Feature Envy		X					X	X
Long Method					X	X		
God Class							X	
Refused Bequest							X	
GWCS (26)				X				
Nenhum CS definido			X					

Observa-se, na Tabela A.7, predominância de *Feature Envy* (P2, P7, P8) e de *Long Method* (P5, P6), além de ocorrências pontuais de *Data Clumps* (P1), *God Class* (P7) e *Refused Bequest* (P7). Um caso singular é o estudo P4, que investiga um conjunto extenso de 26 *Gas-Wasting Code Smells* (*GWCS*) específicos do domínio de contratos inteligentes, incluindo 13 *smells* inéditos nesse domínio, não contemplados por taxonomias tradicionais.

Um aspecto metodologicamente relevante diz respeito ao trabalho P3, que, embora trate da recomendação de *Extract Class*, não define explicitamente nenhum *CS*. Nesse caso, a refatoração é utilizada como objetivo analítico e não como mecanismo de correção de um *smell* formalmente caracterizado. Essa distinção evidencia uma tendência observada em parte da literatura recente: a condução de estudos orientados por operações de refatoração, em vez da identificação explícita de *CS*, o que pode gerar ambiguidades conceituais na delimitação do escopo das investigações.

De forma geral, constata-se escassez de estudos voltados a *smells* arquiteturais, distribuídos ou relacionados à evolução do sistema. Excetuando-se o domínio específico de contratos inteligentes em *blockchain* (P4), a aplicação de LLMs permanece restrita a *CS* localizados e estruturalmente bem definidos, o que a distancia da amplitude e da complexidade semântica contempladas em taxonomias clássicas. Esse cenário revela uma lacuna relevante na litera-

tura, especialmente no que diz respeito à detecção de anomalias sistêmicas e de maior nível de abstração.

RQ3 – Evidências Empíricas sobre o Desempenho dos LLMs

As evidências empíricas dos trabalhos analisados revelam um panorama marcadamente heterogêneo, tanto nas métricas adotadas quanto na estabilidade operacional das abordagens propostas. Os resultados incluem desde avaliações quantitativas baseadas em precisão, *recall* e F1 até análises qualitativas fundamentadas na percepção e na aceitação de desenvolvedores, refletindo a diversidade metodológica que caracteriza o campo.

Em soluções integradas a ambientes de desenvolvimento ou estruturadas em arquiteturas iterativas de refinamento, observam-se índices elevados de aceitação por parte dos desenvolvedores, variando entre 81,3% e 94,4% [62, 63]. De modo complementar, abordagens que associam LLMs a mecanismos estruturais — como filtros sintáticos, verificações determinísticas, hipergráficos ou especialistas heurísticos — apresentam ganhos mensuráveis em métricas clássicas de desempenho, sobretudo em tarefas de análise localizada, como *Move Method* [17, 84] e *Extract Class* [18].

Apesar desses resultados promissores, limitações relevantes persistem. Diversos estudos relatam instabilidade nas respostas dos LLMs, erros de raciocínio, inconsistências semânticas e elevada sensibilidade à formulação do *prompt*. Em artefatos extensos ou em cenários que demandam alto *recall*, o desempenho tende a se degradar significativamente, exigindo mecanismos adicionais de filtragem ou validação [81, 84]. Trabalhos conduzidos em domínios sensíveis, como contratos inteligentes em *blockchain*, evidenciam ainda a necessidade constante de supervisão humana para mitigar falhas críticas, mesmo quando os índices de aprovação subjetiva são elevados [41].

Outro aspecto crítico refere-se às limitações metodológicas da literatura atual. Observa-se escassez de experimentos em larga escala, ausência de comparações sistemáticas com detectores tradicionais de *CS* e carência de estudos de replicação independentes. Essas lacunas reduzem a capacidade de generalização dos achados e indicam que, embora os LLMs apresentem potencial relevante, seu desempenho empírico ainda depende fortemente de arquiteturas híbridas e de mecanismos complementares de controle.

A Tabela A.8 consolida as principais evidências empíricas reportadas pelos estudos ana-

lisados, distinguindo resultados quantitativos, avaliações humanas e observações críticas associadas a cada abordagem.

Tabela A.8: Evidências empíricas sobre o desempenho dos LLMs (RQ3)

ID	**Resultados Empíricos**	**Observações Críticas**
P1	Refatorações corretas em cenários controlados; degradação em projetos grandes.	Código não compilável em larga escala; dependência de validação humana.
P2	Melhorias relativas de 27,8% (precisão), 2,5% (*recall*) e 18,5% (F1).	LLM atua como filtro semântico; impacto indireto no desempenho final.
P3	Ganhos de 38,5% (precisão), 9,7% (*recall*) e 44,4% (F1).	Atributos semânticos do LLM aumentam expressividade do modelo.
P4	94,4% de aprovação pelos desenvolvedores.	Forte dependência de verificações estruturais e validação humana.
P5	81,3% de aceitação pelos usuários.	*Reprompting* e ranqueamento estabilizam parcialmente o LLM.
P6	Avaliação humana (escala 0–4): *Feature Envy* = 2,60; *Refused Bequest* = 2,36; *God Class* = 2,08.	Dificuldade acentuada em *God Class*; limitação de contexto.
P7	Economia de 10,53% (deploy) e 21,53% (message call).	Desempenho reduzido em funções extensas; validação humana necessária.
P8	Melhoria de até 15% em F1.	Queda significativa em cenários que exigem alto *recall*.

Sumariamente, as evidências empíricas indicam que os LLMs podem contribuir de forma relevante para tarefas específicas de detecção e refatoração de *CS*, desde que integrados a mecanismos estruturais e estratégias de controle. Contudo, as limitações observadas reforçam a ideia de que tais modelos ainda não operam de maneira confiável como detectores autônomos, o que motiva a análise sistemática das fragilidades e desafios associados, discutida a seguir na RQ4.

RQ4 – Fragilidades e Desafios Identificados

A análise dos estudos selecionados realça que as limitações associadas ao uso de LLMs na detecção e na refatoração de *CS* não se concentram em um único fator, todavia, distribuem-se por diferentes níveis de análise.

Parte da literatura discute limitações identificadas diretamente relacionadas às soluções propostas, abrangendo aspectos técnicos, metodológicos e de avaliação. Em paralelo, outros trabalhos enfatizam desafios estruturais inerentes ao próprio problema de detecção de *CS*, apontando lacunas do estado da arte que motivam novas abordagens. Essas duas perspectivas são complementares e, em conjunto, permitem uma compreensão mais abrangente dos obstáculos que ainda limitam o avanço da área.

No primeiro nível, destacam-se as limitações associadas às soluções efetivamente implementadas. Cabe ressaltar que nem todos os estudos analisados relatam explicitamente limitações associadas às soluções que propõem. Em particular, alguns trabalhos concentram-se predominantemente na caracterização das lacunas do estado da arte ou em desafios conceituais do próprio problema de detecção de *CS*, sem apresentar uma avaliação crítica detalhada das fragilidades operacionais, metodológicas ou avaliativas de seus métodos. Essa distinção é considerada na organização dos achados da RQ4.

A Tabela A.9 condensa as fragilidades explicitamente relatadas pelos estudos que avaliam seus próprios métodos.

Tabela A.9: Fragilidades explicitamente reportadas nas soluções propostas (RQ4)

Categoria de Fragilidade	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**	**P8**
Escala e contexto	X					X	X	
Fragilidades semânticas do LLM	X			X				
Alinhamento com a prática de desenvolvedores					X	X		
Generalização e escopo				X		X	X	X
Avaliação e *benchmarking*	X				X		X	

Conforme apresentado na Tabela A.9, observam-se restrições recorrentes relacionadas à escala e ao contexto de aplicação, como limitações da janela de contexto, custos computacio-

nais elevados e degradação de desempenho em artefatos extensos (P1, P6, P7). Também são frequentes fragilidades semânticas inerentes aos LLMs, como alucinações, erros de raciocínio e inconsistências entre execuções, especialmente em domínios sensíveis ou em cenários complexos (P1, P4).

A Tabela A.9 ainda mostra outros trabalhos que ressaltam dificuldades de alinhamento com a prática de desenvolvedores, indicando que recomendações automatizadas podem não refletir preferências reais, produzir refatorações pouco reutilizáveis ou falhar em capturar critérios subjetivos envolvidos no julgamento humano (P5, P6). Adicionalmente, limitações de generalização e de escopo são recorrentes, decorrentes do foco em linguagens específicas, em ecossistemas particulares (como IDEs) ou em conjuntos restritos de projetos de código aberto (P4, P6, P7, P8).

Por fim, limitações metodológicas relacionadas à avaliação e ao *benchmarking*, como a ausência de conjuntos de dados reconhecidos, o uso de corpora sintéticos e dificuldades de replicação, também são reportadas, o que reduz a robustez das conclusões apresentadas (P1, P5, P7).

Um segundo nível de análise refere-se aos desafios estruturais do próprio problema de detecção e de refatoração de *CS*. Diferentemente das fragilidades anteriores, esses desafios não caracterizam falhas operacionais das soluções propostas, mas evidenciam limitações conceituais do estado da arte que motivam o desenvolvimento de novas abordagens. Nesse contexto, alguns estudos concentram-se predominantemente na discussão dessas lacunas, sem apresentar uma avaliação crítica detalhada das restrições de seus próprios métodos.

A Tabela A.10 resume esses desafios estruturais.

Tabela A.10: Desafios estruturais do problema e lacunas do estado da arte (RQ4)

Categoria de Desafio Estrutural	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**	**P8**
Modelagem inadequada de dependências complexas		X	X					
Divergência entre recomendações e prática real		X	X		X		X	
Complexidade intrínseca de certos *CS*			X			X	X	

Conforme ilustrado na Tabela A.10, destaca-se, inicialmente, a modelagem inadequada

de dependências complexas, especialmente em abordagens baseadas em relações pareadas, que não capturam dependências *one-to-many* comuns em classes reais (P2, P3). Também é recorrente a divergência entre recomendações automatizadas e as necessidades reais dos desenvolvedores, o que indica que critérios puramente técnicos nem sempre refletem expectativas práticas ou julgamentos humanos (P2, P3, P5, P7). Além disso, alguns *CS* apresentam elevada complexidade intrínseca, como classes extensas e multifuncionais, cuja refatoração exige coordenação global e discernimento semântico, o que dificulta soluções totalmente automatizadas (P3, P6, P7).

Cabe ressaltar que a ausência de marcação de determinados estudos nas matrizes apresentadas não indica lacuna na extração dos dados, mas reflete diferenças no foco analítico adotado pelos próprios trabalhos. Em particular, os estudos P2 e P3 não foram demarcados na Tabela A.9 porque não reportaram limitações operacionais, metodológicas ou avaliativas associadas às soluções propostas. Esses trabalhos concentram-se predominantemente na identificação de lacunas do estado da arte e em desafios conceituais do problema de detecção de *CS*, os quais são apresentados na Tabela A.10.

De forma complementar, a ausência de marcações para P1, P4 e P8 na Tabela A.10 decorre do fato de que esses estudos enfatizam restrições relacionadas à implementação, à avaliação ou ao escopo de suas soluções, sem discutir explicitamente os desafios estruturais do problema ou as limitações conceituais do paradigma vigente. Assim, cada matriz reflete um nível distinto de análise, e a ausência em uma delas é, em si, um resultado relevante sobre o posicionamento metodológico e conceitual dos estudos analisados.

Em síntese, a RQ4 evidencia que as limitações identificadas na literatura assumem naturezas distintas. Enquanto parte dos estudos reporta fragilidades técnicas, metodológicas e de avaliação associadas às soluções propostas, outros enfatizam os desafios estruturais do próprio problema de detecção de *CS*. Esse panorama reforça que, no estado atual da pesquisa, os LLMs atuam predominantemente como componentes auxiliares em arquiteturas híbridas e que avanços significativos dependem tanto do amadurecimento das soluções técnicas quanto de uma melhor compreensão das dificuldades conceituais inerentes ao domínio.

RQ5 – Tendências e Oportunidades Futuras

Diferentemente das RQs anteriores, que se apoiam em evidências empíricas diretas, a RQ5 sintetiza tendências emergentes e oportunidades inferidas a partir da convergência e das lacunas observadas nos estudos analisados.

A literatura aponta para a evolução de arquiteturas híbridas cada vez mais sofisticadas, nas quais LLMs são combinados a mecanismos estruturais e heurísticos para mitigar limitações previamente identificadas e aumentar a estabilidade das recomendações. Entre as direções predominantes estão: (i) o uso de hipergrafos neurais para capturar dependências estruturais complexas [17, 18]; (ii) arquiteturas do tipo MoE, nas quais detectores especializados são articulados por um LLM orquestrador [81]; (iii) pipelines iterativos integrados a ambientes de desenvolvimento, reforçando aceitação e utilidade prática [63, 62]; (iv) modelos de aprendizado profundo enriquecidos por descrições ou atributos semânticos gerados por LLMs [84]; e (v) abordagens voltadas a domínios emergentes, como contratos inteligentes e *smells* associados a consumo de recursos [41].

Cabe observar que essas direções não apresentam o mesmo grau de recorrência na literatura. Enquanto abordagens baseadas em hipergrafos e enriquecimento semântico aparecem de forma consistente em múltiplos estudos, outras tendências — como arquiteturas do tipo MoE ou a combinação com modelos profundos — surgem de forma mais pontual, caracterizando direções emergentes ainda em estágio exploratório.

Além dessas tendências, permanecem abertas oportunidades relevantes para avanço do campo, incluindo: (i) mecanismos mais robustos de verificação automática para reduzir instabilidades e alucinações; (ii) ampliação do escopo semântico das análises, de modo a capturar dependências interclasse ou arquiteturais; (iii) validação das abordagens em bases de código reais, heterogêneas e de grande escala; e (iv) exploração sistemática de linguagens contemporâneas — como TypeScript — e de *smells* arquiteturais, ainda pouco contemplados nos estudos existentes. Essas lacunas evidenciam que o uso de LLMs para detecção de *CS* é um campo em consolidação, com amplo espaço para contribuições metodológicas e empíricas.

Com base nessa análise, a Tabela A.11 organiza as principais tendências e oportunidades futuras identificadas no EMS, estruturando os eixos de evolução recorrentes apontados pela literatura. Diferentemente das matrizes apresentadas em RQs anteriores, essa tabela possui

caráter interpretativo e não de ocorrência por estudo.

Tabela A.11: Tendências e oportunidades futuras identificadas na literatura (RQ5)

Categoria	**Tendências e Oportunidades**
Arquiteturas Híbridas	Consolidação de *pipelines* que combinam LLMs com modelos estruturais (hipergrafos, redes profundas especializadas) e estratégias MoE.
Verificação e Confiabilidade	Desenvolvimento de mecanismos automáticos de verificação e de filtros estruturais para mitigar inconsistências, alucinações e erros semânticos.
Enriquecimento Semântico	Uso de descrições textuais, atributos semânticos ou *embeddings* gerados por LLMs para reforçar modelos estruturais e aprimorar as recomendações.
Integração com IDEs	Tendência de acoplamento direto entre LLMs e ambientes de desenvolvimento, promovendo assistência contínua e maior aceitação por parte dos desenvolvedores.
Domínios Emergentes	Expansão para contextos específicos como contratos inteligentes, refatorações orientadas a alto custo computacional e análise de novos ecossistemas de software.
Benchmarking e Protocolos	Necessidade de datasets padronizados, tarefas reprodutíveis, métricas consistentes e *benchmarks* multilinguagem para comparação entre abordagens.
Automação Progressiva	Busca por reduzir a dependência de validação humana e aumentar a autonomia dos pipelines, sem comprometer a verificabilidade e a estabilidade.

Nesse sentido, essas oportunidades decorrem diretamente das limitações e dos desafios discutidos na RQ4, delineando caminhos promissores para a evolução da área. Elas também fundamentam as escolhas metodológicas desta tese, que busca contribuir para a consolidação empírica do campo ao explorar linguagens contemporâneas, cenários realistas e protocolos experimentais comparáveis, os quais são operacionalizados nos capítulos desta proposta de tese.

A.3.3 Síntese dos Resultados por Questão de Pesquisa

Com o objetivo de sintetizar, de forma objetiva, as respostas às questões de pesquisa investigadas neste EMS, a Tabela A.12 apresenta um sumário consolidado dos principais achados associados a cada RQ. Esse sumário funciona como uma visão de referência, permitindo relacionar diretamente cada questão de pesquisa às conclusões derivadas da análise sistemática dos estudos selecionados.

Tabela A.12: Sumário dos achados do EMS por RQ

RQ	**Síntese dos Achados**
RQ1	A literatura emprega predominantemente arquiteturas híbridas para aplicar LLMs à detecção e à refatoração de *CS*, nas quais esses modelos atuam como componentes semânticos auxiliares integrados a mecanismos estruturais, heurísticos ou estatísticos, e não como detectores autônomos.
RQ2	Os estudos concentram-se majoritariamente em *CS* estruturais de baixa granularidade, como *Feature Envy* e *Long Method*, com escassa investigação de *smells* arquiteturais, distribuídos ou relacionados à evolução do sistema, excetuando-se domínios específicos como contratos inteligentes.
RQ3	As evidências empíricas indicam ganhos relevantes em cenários controlados e em tarefas localizadas, porém, com forte dependência de validação humana, de mecanismos estruturais complementares e de estratégias de controle para mitigar instabilidades, limitações de contexto e erros semânticos.
RQ4	As limitações identificadas distribuem-se entre fragilidades técnicas, metodológicas e avaliativas das soluções propostas e desafios estruturais inerentes ao próprio problema de detecção de *CS*, evidenciando tanto restrições operacionais quanto lacunas conceituais do estado da arte.
RQ5	Emergem tendências voltadas ao fortalecimento de arquiteturas híbridas, ao uso de modelos estruturais avançados e à integração com ambientes de desenvolvimento, bem como oportunidades associadas à ampliação de escopo, validações em larga escala e exploração de linguagens e *CS* ainda pouco investigados.

A.3.4 Discussão Geral dos Achados

A análise integrada das questões de pesquisa evidencia que a aplicação de LLMs à detecção e à refatoração de *CS* constitui um campo em expansão recente, ainda em processo de consolidação científica. Embora os estudos analisados demonstrem avanços relevantes, especialmente no uso de arquiteturas híbridas e na incorporação de informações semânticas ao processo de análise de código, tais progressos permanecem fortemente condicionados a mecanismos complementares de controle, validação e filtragem.

Os achados das RQs revelam um padrão recorrente: os LLMs têm sido empregados predominantemente como componentes auxiliares, responsáveis por enriquecer representações, gerar descrições semânticas ou refinar recomendações, enquanto a precisão estrutural e a confiabilidade das decisões dependem de análises estáticas, de heurísticas especializadas ou de supervisão humana. Esse resultado transversal indica que, no estado atual da literatura, tais modelos ainda não apresentam robustez suficiente para operar como detectores autônomos de *CS*.

Do ponto de vista empírico, observa-se que os ganhos reportados tendem a se concentrar em tarefas de baixa granularidade e em cenários controlados, ao passo que artefatos extensos, dependências complexas e *CS* de maior complexidade semântica continuam a representar desafios substanciais. Essa limitação é reforçada pela heterogeneidade metodológica dos estudos, pela escassez de avaliações em larga escala e pela ausência de *benchmarks* padronizados que permitam comparações diretas entre abordagens.

Adicionalmente, a concentração da literatura em um conjunto restrito de *CS* e de linguagens evidencia um escopo investigativo ainda limitado, distante da diversidade de problemas observados em sistemas reais. A ausência de estudos sistemáticos sobre linguagens amplamente utilizadas na indústria, bem como a baixa atenção a *smells* arquiteturais e distribuídos, evidenciam lacunas relevantes a serem exploradas.

Em conjunto, esses achados sugerem que o uso de LLMs na detecção de *CS* deve ser compreendido, no momento, como parte de um ecossistema de análise híbrido, no qual tais modelos ampliam capacidades semânticas, mas não substituem mecanismos estruturais consolidados. Essa leitura integrada fundamenta a necessidade de investigações empíricas mais robustas, comparativas e reprodutíveis, capazes de avaliar, de forma sistemática, o papel dos LLMs em cenários realistas de desenvolvimento.

A.4 Ameaças à Validade

A condução de um EMS envolve decisões metodológicas e interpretativas que podem influenciar a abrangência e a precisão dos resultados. Nesta seção, são discutidas as principais ameaças à validade identificadas neste estudo, organizadas conforme as categorias clássicas propostas por Wohlin *et al.* [80]. O objetivo é explicitar os limites do EMS, bem como as estratégias adotadas para mitigar seus impactos.

A.4.1 Validade de Construto

A validade de construto refere-se à adequação entre os conceitos investigados e os mecanismos empregados para identificá-los na literatura. Uma ameaça relevante nesse contexto decorre da formulação da *string* de busca. Embora construída de forma sistemática, há risco de que terminologias emergentes ou variações semânticas não tenham sido plenamente capturadas. Em áreas de pesquisa em rápida evolução, como a aplicação de modelos de linguagem à ES, diferentes estudos podem empregar terminologias distintas para descrever abordagens conceitualmente semelhantes, o que dificulta a cobertura exaustiva da literatura por meio de uma única formulação de busca.

Para mitigar essa ameaça, adotou-se um processo iterativo de refinamento da *string*, que incluiu a análise de estudos de referência, a incorporação de sinônimos e termos correlatos, bem como a realização de buscas-piloto para validação preliminar. Ainda assim, reconhece-se que o campo evolui rapidamente e que novas nomenclaturas podem ter surgido após o período de coleta, o que caracteriza uma ameaça residual inerente a áreas emergentes.

A.4.2 Validade Interna

A validade interna está associada ao risco de vieses durante a seleção, a triagem e a categorização dos estudos. Uma ameaça específica deste EMS decorre da dependência inicial da base Scopus, o que pode ter excluído trabalhos relevantes não indexados nessa fonte. Essa limitação foi parcialmente mitigada por meio da aplicação de *forward snowballing*, estratégia que permitiu identificar estudos adicionais a partir das referências dos artigos selecionados.

Além disso, o processo de aplicação dos critérios de inclusão e exclusão envolve julgamentos interpretativos. Para reduzir vieses individuais, cada estudo foi avaliado por pelo me-

nos dois pesquisadores de forma independente, seguindo um procedimento de dupla revisão. As divergências foram discutidas até o consenso. Ainda assim, variações na interpretação de conceitos como "uso de LLMs"ou "detecção de *CS*"representam uma ameaça metodológica residual, comum em estudos de natureza qualitativa.

A.4.3 Validade Externa

A validade externa refere-se à capacidade de generalização dos achados. Uma limitação importante decorre do reduzido número de estudos identificados e da forte concentração temporal das publicações (2024–2025), o que indica que a aplicação de LLMs à detecção de *CS* constitui um campo ainda emergente.

Adicionalmente, muitos estudos foram conduzidos em ambientes controlados ou com *datasets* específicos e de pequena escala, frequentemente restritos a determinadas linguagens ou ecossistemas. Isso limita a extrapolação dos resultados para cenários industriais mais amplos. Em particular, a ausência de investigações sobre linguagens amplamente utilizadas na indústria, como TypeScript, restringe a generalização dos achados a esse contexto específico.

A.4.4 Validade de Conclusão

A validade de conclusão refere-se à consistência entre os dados analisados e as interpretações derivadas. Uma ameaça relevante nesse aspecto é a elevada heterogeneidade metodológica entre os estudos incluídos no EMS. Observam-se variações significativas nas métricas adotadas, nos protocolos experimentais, nos tipos de *CS* investigados e nos contextos de avaliação, o que dificulta comparações diretas e inviabiliza sínteses quantitativas mais robustas.

Diante desse cenário, a análise foi conduzida de forma predominantemente qualitativa, buscando identificar padrões recorrentes e tendências consistentes na literatura. No entanto, a ausência de *benchmarks* padronizados e de métricas uniformes constitui uma limitação estrutural do campo, o que impacta a força das conclusões agregadas.

A.4.5 Síntese das Ameaças

Em conjunto, as ameaças discutidas não invalidam os achados deste EMS, mas delimitam seu escopo interpretativo. Elas refletem tanto as limitações inerentes ao método de mapeamento

sistemático quanto as características de um campo de pesquisa em estágio inicial de consolidação. Ao tornar explícitas essas restrições, o estudo reforça sua transparência metodológica e estabelece bases sólidas para investigações empíricas mais controladas, comparativas e reprodutíveis.

A.5 Considerações Finais do EMS

Este estudo apresentou um mapeamento sistemático com o objetivo de caracterizar o estado da arte sobre o uso de LLMs na detecção de *CS* no contexto da ES. A análise sistemática da literatura evidenciou que se trata de um campo emergente, em rápida expansão, ainda em processo de consolidação científica.

Os resultados indicam que as abordagens existentes concentram-se predominantemente em arquiteturas híbridas, nas quais os LLMs atuam como componentes semânticos auxiliares integrados a mecanismos estruturais, heurísticos ou estatísticos. Observou-se também um escopo investigativo restrito, com forte ênfase em *CS* estruturais de baixa granularidade e em avaliações conduzidas majoritariamente em cenários controlados, frequentemente dependentes de validação humana e de mecanismos complementares de controle.

Apesar das evidências empíricas promissoras reportadas na literatura, persistem limitações técnicas, metodológicas e conceituais que dificultam a adoção dos LLMs como detectores autônomos de *CS*. Entre essas limitações, destacam-se a instabilidade das respostas, as restrições de contexto, as dificuldades de generalização e a ausência de *benchmarks* padronizados que permitam comparações sistemáticas com ferramentas tradicionais de análise estática.

Adicionalmente, o EMS revelou lacunas relevantes na literatura atual, como a ausência de estudos envolvendo linguagens amplamente utilizadas na indústria, como TypeScript, e a escassez de investigações voltadas a *CS* arquiteturais ou distribuídos. Essas lacunas reforçam a necessidade de estudos empíricos mais robustos, comparativos e reprodutíveis.

Portanto, este EMS estabeleceu uma base conceitual e empírica sólida para a continuidade da proposta de tese, ao evidenciar tanto o potencial quanto as limitações atuais do uso de LLMs na detecção de *CS*. Os achados apresentados fundamentaram o delineamento dos capítulos da proposta, que investigam, de forma sistemática e controlada, o desempenho

de LLMs em comparação com ferramentas tradicionais em projetos reais desenvolvidos em TypeScript.

Apêndice B

Diretrizes para Uso de IA em Atividades Acadêmicas

Este apêndice apresenta as diretrizes que fundamentaram o uso de ferramentas de IA ao longo do desenvolvimento deste trabalho, estabelecendo princípios, limites e boas práticas adotadas pelo autor no emprego dessas tecnologias nas atividades da pesquisa. O presente guia é adotado como referência pelo grupo de pesquisa *Software Practices Laboratory* (SPLab), vinculado à UFCG, com o objetivo de assegurar a transparência, a integridade científica e a responsabilidade autoral no uso de modelos de linguagem.

As orientações aqui reunidas estão alinhadas às recomendações contemporâneas sobre o uso ético da IA na pesquisa e delimitam explicitamente os tipos de uso permitidos, as restrições aplicáveis e os cuidados necessários para garantir que essas ferramentas atuem exclusivamente como apoio à escrita, à organização textual e à revisão linguística, sem substituir o julgamento crítico, a autoria intelectual ou o rigor metodológico exigido em trabalhos científicos.

Guidelines for the use of AI in academic activities

Context

This document aims to guide students on the use of AI, especially generative models, in their various academic activities. In addition to a general discussion of the ethical issues and implications of use not aligned with the guidelines presented here. This material is strongly based on the document "*Guidelines for the ethical and responsible use of Generative Artificial Intelligence: a practical guide for researchers (in portuguese)*"[1] and on guidelines extracted from scientific conferences in the field of Computer Science.

Guidelines

In general, we understand that:

> The model cannot replace the intellectual work that is your responsibility. It can be used, appropriately, as an assistant to accelerate mechanical tasks resulting from the directions you provide.

Transparency. Any use, no matter how minor, must be explicitly declared in your work. In articles or final undergraduate papers (TCCs), this is usually done in the *Acknowledgements* section or similar. Proposals, dissertations, qualifications, and theses must include this declaration in the introduction. See the format suggested by the document "Guidelines for the ethical and responsible use of Generative Artificial Intelligence: a practical guide for researchers"[1]:

"During the preparation of this work, the author(s) utilized [name of the tool/model or service] version [number and/or date] for [justify the reason]. After using this tool/model/service, the author(s) reviewed and edited the content in compliance with the scientific method and assume(s) full responsibility for the content of the publication."

Review and responsibility. The final content is your responsibility. You are the author. Therefore, carefully review everything generated through interaction with the models. Even in prompts that only require revisions, models may include, remove, or change the tone of important passages in the text.

Acceptable Use

As a general rule, the use of text generators or AI references is accepted in very specific cases, which may include activities such as:

- Summarizing articles to facilitate the understanding of complex ideas;
- Brainstorming ideas to assist in defining the problem, methodology, etc.;
- Assisting in learning programming, such as the use of libraries and frameworks, **without direct code generation for the article/document;**
- Reviewing writing style, with direct translation from Portuguese to English being prohibited. The text must be drafted in English and can be revised. **Do not directly paste the result of the revision**; rewrite the text based on the AI's result;
- Suggestions for related research, using specialized research tools, avoiding generalist models such as chatGPT, Copilot, or Gemini.

Even so, the following conditions must be observed:

- No generated text may be copied and pasted directly into the text of your documents, articles, and presentations;
- Use exclusively professional or paid versions, or open-source models, always avoiding models with limited capacity due to being a free version;
- In the case of using AI to search for bibliographic references, **each of the references must be manually checked**, and its summary must be read by the researcher. Under no circumstances may AI be used to automatically generate citations;
- Use is permitted, without citation, for:
    - Using AI or AI-assisted technologies to improve image quality in terms of contrast and clarity;
    - Using generative AI tools to edit and improve the quality of your existing text (similar to an assistant like Grammarly for improving spelling, grammar, punctuation, clarity, and engagement).

Implications of Inappropriate Use

Any failure to satisfy these conditions constitutes research misconduct and a breach of the researcher's trust with their research group, with the probable consequence being the termination of supervision or participation in projects.

References

[1] SAMPAIO, Rafael Cardoso; SABBATINI, Marcelo; LIMONGI, Ricardo. Diretrizes para o uso ético e responsável da Inteligência Artificial Generativa: um guia prático para pesquisadores [livro eletrônico]. São Paulo: Sociedade Brasileira de Estudos Interdisciplinares da Comunicação – Intercom, 2024. Disponível em: https://prpg.unicamp.br/wp-content/uploads/sites/10/2025/01/livro-diretrizes-ia-1.pdf

Apêndice C

Artefatos de Pesquisa e Reprodutibilidade

Este apêndice tem como objetivo disponibilizar os principais artefatos produzidos ao longo do desenvolvimento desta proposta de tese de doutorado, em consonância com as boas práticas de Ciência Aberta (*Open Science*), a transparência metodológica e a reprodutibilidade científica.

Ao longo da pesquisa, foram gerados diversos artefatos digitais, incluindo conjuntos de dados, *scripts* de extração e análise, *prompts* utilizados nos experimentos com modelos de linguagem, arquivos de configuração e resultados intermediários. Em razão de sua natureza, volume e heterogeneidade de formatos, esses materiais não se adequam plenamente à incorporação direta no corpo do texto da proposta, conforme as normas acadêmicas vigentes, podendo comprometer a legibilidade, a padronização e a preservação adequada do documento.

Dessa forma, optou-se por disponibilizar esses artefatos por meio de uma página web dedicada, acessível a partir do link a seguir:

- Link para Acessar Artefatos de Pesquisa.

A página web reúne os artefatos resultantes do desenvolvimento desta proposta de tese, organizados em diferentes formatos, bem como arquivos de documentação que descrevem detalhadamente o conteúdo disponibilizado, a finalidade de cada artefato e a estrutura de organização dos materiais, servindo como guia para pesquisadores interessados em com-

preender, auditar e apoiar a reprodutibilidade dos experimentos conduzidos. O repositório inclui documentação em inglês (`README.md`) e em português (`LEIAME.txt`), de modo a assegurar acessibilidade tanto ao público internacional quanto aos leitores desta proposta de tese.

A disponibilização desses artefatos visa fortalecer a confiabilidade dos resultados apresentados, facilitar a replicação dos estudos conduzidos e fomentar o avanço do conhecimento científico na área, permitindo que outros pesquisadores utilizem, avaliem e estendam os materiais produzidos no contexto desta proposta de tese.