Análise Comparativa entre Estratégias Baseadas em LLMs para Detecção de Ambiguidades Vagas e Não Mensuráveis em Requisitos de Software escritos em Português

Francisco Leonel Ferreira dos Santos
Instituto Federal do Maranhão (IFMA), Campus Caxias
Caxias, Maranhão, Brazil
leonel.s@acad.ifma.edu.br

Antonio Wills da Silva Santos
Instituto Federal do Maranhão (IFMA), Campus Caxias
Caxias, Maranhão, Brazil
antonio.w@acad.ifma.edu.br

Evelly Victory Vieira Pinto
Instituto Federal do Maranhão (IFMA), Campus Caxias
Caxias, Maranhão, Brazil
victoryvieira@acad.ifma.edu.br

Rafael Torres Anchiêta
Instituto Federal do Maranhão (IFMA), Campus Caxias
Caxias, Maranhão, Brazil
rafael.torres@ifma.edu.br

Resumo

A detecção automática de ambiguidades em requisitos de software é um desafio recorrente na Engenharia de Requisitos, especialmente em português, língua para a qual há escassez de conjuntos de dados rotulados. Este trabalho compara quatro estratégias aplicadas ao modelo Gemma 3 1B Instruct: zero-shot, few-shot, chain-of-thought controlado e fine-tuning supervisionado com LoRA. Os experimentos utilizaram 7.000 requisitos sintéticos, distribuídos entre o desenvolvimento e o teste. No conjunto de teste com 2.100 exemplos, o fine-tuning alcançou acurácia e F1-macro de 1,000; entre as estratégias de prompting, o few-shot obteve os melhores resultados, com acurácia de 0,538 e F1-macro de 0,500. Entretanto, uma auditoria mostrou que 79,24% dos exemplos de teste possuíam padrões estruturais já observados no treinamento. Assim, o desempenho do fine-tuning evidencia aprendizado consistente do padrão do conjunto sintético, mas não comprova generalização semântica para requisitos reais. Desse modo, os resultados reforçam a necessidade de avaliações futuras com dados reais.

Palavras-chave

Engenharia de Requisitos, Detecção de Ambiguidades, Processamento de Linguagem Natural

1 Introdução

A Engenharia de Requisitos consiste no processo de elicitar, analisar, especificar e gerenciar os requisitos de um sistema ao longo de seu ciclo de vida, sendo considerada uma das áreas fundamentais da Engenharia de Software [16]. Durante a etapa de especificação, espera-se que os requisitos sejam claros, completos, consistentes e de fácil compreensão. Entretanto, na prática, alcançar tais características é uma tarefa complexa, uma vez que requisitos escritos em linguagem natural frequentemente apresentam ambiguidades, termos vagos e múltiplas interpretações. Essas inconsistências podem gerar conflitos entre clientes, analistas e desenvolvedores, ocasionando retrabalho, aumento de custos e falhas no desenvolvimento do software.

Dentre os diferentes problemas presentes na especificação de requisitos, destacam-se as ambiguidades semânticas e subjetivas. Segundo Ferrari et al. [6], uma expressão é considerada ambígua quando pode ser interpretada de diferentes maneiras pelos envolvidos no processo de desenvolvimento. Nesse contexto, termos vagos, subjetivos ou não mensuráveis, como "rápido", "intuitivo" ou "seguro", podem gerar múltiplas interpretações entre clientes, analistas e desenvolvedores, comprometendo a comunicação e a consistência dos requisitos. Um exemplo de requisito ambíguo é: "A nova interface do painel de controle deve ser intuitiva e fácil de usar, minimizando os erros cometidos pelos usuários", pois termos como "intuitivo" e "fácil" dependem inteiramente do histórico e da familiaridade técnica de quem usa.

A redução de ambiguidades em requisitos de software tem sido investigada por meio de diferentes abordagens ao longo do tempo. As primeiras propostas apoiavam-se em regras linguísticas e em métodos clássicos de Processamento de Linguagem Natural (PLN) [8, 12]. Mais recentemente, técnicas de Deep Learning passaram a explorar representações semânticas contextualizadas para a detecção automática de ambiguidades [4, 5]. Diferentemente de representações estáticas, que associam uma palavra a um mesmo vetor independentemente do uso, os *contextual embeddings* representam cada ocorrência de acordo com as palavras ao seu redor [1]. Por fim, os Grandes Modelos de Língua (*Large Language Models* — LLMs) têm demonstrado capacidade de seguir instruções, realizar classificação e produzir respostas textuais estruturadas [2, 17].

Devido aos avanços dos modelos de língua, o presente artigo propõe uma análise comparativa entre estratégias baseadas em LLMs para detecção automática de ambiguidades em requisitos de software escritos em português, com foco em requisitos vagos e não mensuráveis. A abordagem adotada envolve o uso de técnicas de prompting, incluindo zero-shot, few-shot e chain-of-thought, além da comparação com um modelo ajustado por fine-tuning, com o objetivo de analisar o desempenho, os limites de generalização e a qualidade das respostas geradas.

O modelo utilizado foi o Gemma 3 1B Instruct do Google, que oferece um equilíbrio entre capacidade de compreensão semântica, geração textual e viabilidade computacional, sendo adequado para tarefas de análise de requisitos em linguagem natural. Por se tratar de um modelo de porte reduzido, ele permite realizar ajustes finos e experimentos sem exigir infraestrutura extremamente robusta.

O restante do artigo está organizado da seguinte forma: a Seção 2 apresenta a fundamentação teórica; a Seção 3 descreve a metodologia adotada; a Seção 4 discute os resultados obtidos; a Seção 5 apresenta as considerações finais; e a Seção 5.1 discute as limitações do estudo.

2 Fundamentação Teórica

Para melhor compreensão do trabalho, esta seção está organizada nas seguintes subseções: Qualidade de Requisitos e Ambiguidades (Seção 2.1), LLMs e Estratégias de Prompting (Seção 2.2) e Fine-tuning Supervisionado e LoRA (Seção 2.3).

2.1 Qualidade de Requisitos e Ambiguidades

A Engenharia de Requisitos é responsável por identificar, documentar e validar as necessidades que orientam o desenvolvimento de um sistema. Como grande parte dos requisitos é escrita em língua natural, os documentos produzidos nessa etapa podem apresentar imprecisões, ambiguidades e diferentes possibilidades de interpretação. Sommerville [16] destaca que a língua natural, embora acessível aos envolvidos no projeto, pode gerar problemas de clareza e precisão na especificação de requisitos.

A qualidade de um requisito está associada a características como clareza, consistência, completude, verificabilidade e ausência de ambiguidades. Nesse contexto, a ambiguidade representa um problema relevante, pois ocorre quando a mesma declaração pode ser interpretada de mais de uma forma pelos stakeholders. Gervasi et al. [8] discutem que as ambiguidades nos requisitos têm sido amplamente estudadas na Engenharia de Requisitos, especialmente devido à sua influência na comunicação entre clientes, analistas e desenvolvedores.

Este trabalho concentra-se em dois problemas específicos de especificação: a vagueza e a ausência de mensurabilidade. Requisitos vagos são aqueles que apresentam termos subjetivos ou dependentes de interpretação, como "fácil", "intuitivo", "amigável", "adequado" ou "robusto". Esses termos não definem critérios claros de aceitação, o que dificulta a validação do requisito e pode gerar desalinhamento entre a expectativa do cliente e a implementação realizada.

A ausência de mensurabilidade, por sua vez, ocorre quando o requisito menciona uma propriedade desejada do sistema sem apresentar um critério objetivo de verificação. Esse problema é comum em requisitos relacionados a desempenho, segurança, disponibilidade, eficiência ou escalabilidade. Por exemplo, a sentença "o sistema deve responder rapidamente" indica uma expectativa de desempenho, mas não define o tempo máximo de resposta, o percentual de requisições atendidas ou a condição de teste. Assim, neste trabalho, requisitos não mensuráveis são tratados como aqueles que precisam de métricas, limites, percentuais, prazos ou critérios objetivos para se tornarem verificáveis.

2.2 LLMs e Estratégias de Prompting

Os Grandes Modelos de Língua, ou *Large Language Models* (LLMs), têm sido utilizados em tarefas de Processamento de Linguagem Natural por sua capacidade de interpretar instruções, classificar textos e gerar respostas estruturadas [2, 17]. No contexto deste trabalho, a detecção de ambiguidades vai além de uma classificação textual simples: o modelo recebe um requisito, atribui uma das classes `vago`, `não_mensurável` ou `nenhuma` e produz um diagnóstico estruturado. Esse diagnóstico inclui a justificativa da decisão e uma sugestão de melhoria; nas estratégias de prompting, também é solicitado o trecho problemático. Assim, avalia-se tanto a categoria prevista quanto a capacidade de explicar e de apoiar a reformulação do requisito.

Uma das formas de utilizar LLMs é por meio de *prompting*, isto é, pela elaboração de instruções textuais que orientam o comportamento do modelo. Brown et al. [2] mostram que modelos de língua podem executar tarefas em cenários *zero-shot* e *few-shot*, sem atualização dos pesos do modelo. No *zero-shot*, o modelo recebe apenas a descrição da tarefa, as classes possíveis e as regras de decisão. Essa abordagem tem baixo custo de configuração, mas pode ser limitada quando as fronteiras entre as classes são sutis.

No *few-shot*, o prompt inclui exemplos de entrada e de saída antes da classificação do novo requisito. Esses exemplos servem como demonstrações do comportamento esperado, ajudando o modelo a compreender melhor a diferença entre as classes. Neste trabalho, essa estratégia é relevante porque a distinção entre `vago`, `não_mensurável` e `nenhuma` exige interpretar a função da expressão no contexto da sentença. Em outras palavras, o modelo precisa interpretar o requisito para decidir se ele é vago, não mensurável ou já apresenta critérios objetivos. Assim, a classificação não pode se basear apenas na presença de palavras específicas.

Outra estratégia avaliada foi o *chain-of-thought* (CoT). Wei et al. [17] apresentam o CoT como uma técnica que induz o modelo a realizar etapas intermediárias de raciocínio antes da resposta final, especialmente em tarefas que exigem inferência. Neste estudo, adotou-se uma forma controlada dessa estratégia: o modelo é instruído a analisar internamente o requisito, mas deve retornar apenas uma justificativa resumida em formato estruturado. Essa decisão busca evitar respostas longas ou difíceis de processar automaticamente, preservando a saída em um formato compatível com a extração da classe prevista.

2.3 Fine-tuning Supervisionado e LoRA

Embora as estratégias baseadas em prompts sejam flexíveis e não exijam treinamento adicional, seu desempenho pode variar conforme a formulação e a ordem das demonstrações, especialmente em tarefas com fronteiras de decisão sutis [13]. Na classificação de requisitos, uma pequena mudança pode alterar a classe correta: "O sistema deve responder rapidamente" é `não_mensurável`, pois não define um limite de tempo, enquanto "O sistema deve responder em até 2 segundos para 95% das requisições" é `nenhuma`, por apresentar critérios verificáveis. Por isso, além das estratégias zero-shot, few-shot e CoT controlado, este trabalho também avalia o fine-tuning supervisionado, no qual o modelo é ajustado a partir de exemplos rotulados no formato instrução-resposta.

O fine-tuning supervisionado permite adaptar o comportamento do modelo ao domínio específico da tarefa. Neste trabalho, cada exemplo de treinamento contém um requisito textual e uma resposta esperada em formato JSON, composta pelos campos `tipo`, `justificativa` e `sugestão_melhoria`. Dessa forma, o treinamento busca ensinar ao modelo não apenas a classe correta, mas também o padrão de saída necessário para uso em um pipeline automático.

Para tornar o ajuste computacionalmente viável, utilizou-se a técnica LoRA (*Low-Rank Adaptation*). Hu et al. [11] propõem o LoRA como uma estratégia de adaptação eficiente de parâmetros, na qual os pesos originais do modelo são mantidos congelados e apenas matrizes adicionais de baixo posto são treinadas. Essa abordagem reduz os custos de memória e de processamento em comparação ao fine-tuning completo, tornando possível adaptar modelos de linguagem em ambientes com recursos computacionais limitados.

Neste trabalho, o modelo adotado para o fine-tuning foi o *Gemma 3 1B Instruct*. A escolha desse modelo está relacionada ao equilíbrio entre a capacidade linguística e o custo computacional. O relatório técnico do Gemma 3 descreve a família como composta por modelos leves em diferentes escalas, incluindo versões de 1B, 4B, 12B e 27B de parâmetros [7]. Assim, o uso da versão 1B torna o experimento mais viável em ambientes como o Google Colab, mantendo a compatibilidade com a proposta de avaliar soluções baseadas em LLMs para apoio à Engenharia de Requisitos.

3 Metodologia

A metodologia adotada neste trabalho é de natureza experimental e comparativa. Conforme Gil [9], a pesquisa experimental consiste na definição de um objeto de estudo, na seleção de variáveis capazes de influenciá-lo e na observação dos efeitos produzidos por essas variáveis. Neste estudo, tal característica se manifesta na avaliação de diferentes estratégias baseadas em LLMs para a classificação automática de requisitos de software. Além disso, a pesquisa assume caráter comparativo, pois, segundo Marconi e Lakatos [14], o método comparativo permite examinar semelhanças e diferenças entre os fenômenos analisados. Assim, comparam-se os desempenhos de diferentes abordagens de prompting na identificação de requisitos vagos e não mensuráveis.

O objetivo metodológico foi verificar a capacidade do modelo Gemma 3 1B Instruct de classificar requisitos escritos em linguagem natural em três classes: `vago`, `não_mensurável` e `nenhuma`. A classe `vago` foi utilizada para requisitos que apresentam termos subjetivos ou dependentes de interpretação, como "fácil", "intuitivo", "adequado" ou "amigável". A classe `não_mensurável` foi atribuída a requisitos que indicam propriedades como desempenho, segurança, disponibilidade ou eficiência sem apresentar métrica objetiva. Por fim, a classe `nenhuma` foi usada para requisitos claros, verificáveis e objetivos no escopo deste trabalho.

A utilização de dados sintéticos gerados por IA para o treinamento e a validação de modelos tem recebido atenção crescente na literatura recente sobre Aprendizado Profundo. Conforme discutido por de Melo et al. [3], o desenvolvimento de modelos preditivos frequentemente é limitado pela disponibilidade de dados de treinamento em quantidade e qualidade adequadas, especialmente em cenários com restrições de acesso, privacidade ou segurança. Nesse contexto, os dados sintéticos surgem como uma alternativa metodológica relevante, pois podem ampliar a diversidade dos exemplos disponíveis e apoiar a construção de modelos mais robustos, desde que sejam incorporados de forma controlada aos pipelines de aprendizagem.

Embora o uso de dados sintéticos seja amplamente discutido em domínios críticos e regulados, como a área da saúde, onde pode contribuir para contornar a escassez de dados, apoiar simulações e reduzir o risco de exposição de informações sensíveis [15], seus benefícios também podem ser explorados no contexto da Engenharia de Software. Nesta pesquisa, a geração sintética de requisitos permite construir cenários variados de falhas de especificação, incluindo casos de ambiguidade, vagueza e ausência de mensurabilidade, que poderiam ser difíceis, custosos ou demorados de coletar em bases reais de projetos corporativos.

Apesar dessas vantagens, o uso de dados sintéticos exige cautela metodológica. Conforme apontam Hao et al. [10], conjuntos de dados artificiais podem introduzir riscos estatísticos, como mudanças na distribuição dos dados em relação ao mundo real, redução da variabilidade linguística e reprodução de padrões artificiais gerados pelo próprio modelo. Tais limitações podem comprometer a capacidade de generalização dos modelos treinados, especialmente quando a avaliação é realizada apenas em dados com características semelhantes às utilizadas no treinamento.

Para mitigar esses riscos, a construção do conjunto de dados desta pesquisa foi orientada por critérios de diversidade, balanceamento das classes e revisão dos exemplos gerados. O modelo Claude Sonnet 4.6, da Anthropic, foi utilizado para gerar os 7.000 requisitos sintéticos iniciais, com diferentes formulações textuais e níveis de ambiguidade. Entretanto, reconhece-se que a geração automática, por si só, não garante realismo nem representatividade plena. Por esse motivo, os dados sintéticos foram tratados como uma base experimental inicial, adequada para avaliar o comportamento comparativo das estratégias propostas, mas ainda dependente de validação futura com requisitos reais e de anotação especializada.

3.1 Conjunto de Dados

O conjunto de dados foi estruturado no formato instrução-resposta. Cada instância contém um requisito textual como entrada e uma saída esperada em formato JSON, composta pelos campos `tipo`, `justificativa` e `sugestão_melhoria`. Essa estrutura foi adotada para permitir tanto a classificação automática do requisito quanto a geração de uma explicação resumida e de uma possível reformulação. Nas estratégias de prompting, o campo adicional `trecho_problemático` foi solicitado para localizar a expressão associada ao diagnóstico.

Utilizaram-se dados sintéticos devido à escassez de bases públicas em português voltadas especificamente a ambiguidades em requisitos de software. Essa escolha permite construir exemplos controlados e variados, embora apresente limitações quanto à representatividade de requisitos reais.

O conjunto de desenvolvimento continha 4.900 exemplos e foi dividido, de forma estratificada, em 4.165 para treinamento e 735 para validação. O teste foi mantido separado, com 2.100 exemplos. A Tabela 1 apresenta a distribuição por classe. `ERRO_PARSE` não constitui uma classe do conjunto de dados: trata-se apenas de um estado registrado durante a avaliação, quando a resposta gerada não pôde ser interpretada conforme o esquema JSON esperado.

3.2 Estratégias Avaliadas

Foram avaliadas quatro estratégias. A primeira foi o *zero-shot prompting*, em que o modelo recebe apenas a instrução da tarefa, as classes possíveis, as regras de classificação e o formato de saída obrigatório.

Tabela 1: Distribuição dos conjuntos utilizados nos experimentos.

Classe	Treino	Validação	Teste	Total
vago	1.389	245	700	2.334
não_mensurável	1.388	245	700	2.333
nenhuma	1.388	245	700	2.333
Total	4.165	735	2.100	7.000

A segunda foi o *few-shot prompting*, que acrescenta quatro exemplos fixos de entrada e saída ao prompt, com o objetivo de orientar melhor o comportamento do modelo. Esses exemplos foram definidos manualmente pelos autores como demonstrações representativas das três classes, permaneceram inalterados durante toda a execução e não foram extraídos do conjunto de teste. A terceira foi o *chain-of-thought controlado*, no qual o modelo é instruído a analisar o requisito internamente, mas deve retornar apenas uma justificativa resumida, evitando respostas extensas e difíceis de processar. A quarta estratégia foi o fine-tuning supervisionado do modelo Gemma 3 1B Instruct, com adaptação via LoRA em FP16.

A escolha do Gemma 3 1B foi motivada pelo menor custo computacional em comparação com modelos maiores, permitindo o treinamento no Google Colab com GPU T4. A técnica LoRA foi adotada por permitir a adaptação eficiente do modelo sem atualizar todos os parâmetros. O treinamento foi configurado para até três épocas, taxa de aprendizado de 1×10^{-4}, lote de duas amostras por dispositivo e acumulação de gradiente de oito passos, o que equivale a um lote efetivo de 16 amostras por GPU. O comprimento máximo foi de 512 tokens. Na adaptação LoRA, utilizaram-se $r = 8$, $\alpha = 16$ e dropout de 0,05; o treinamento foi realizado em FP16, com parada antecipada por paciência de 2.

3.3 Prompts Utilizados

Para reduzir repetição, as três estratégias de prompting compartilharam as definições de classe e o esquema de saída. A Figura 1 apresenta a instrução-base de forma condensada. As diferenças entre as estratégias consistiram na presença das quatro demonstrações do few-shot (Figura 2) e na instrução de raciocínio interno do CoT controlado.

No zero-shot, apenas a instrução-base foi utilizada. No chain-of-thought controlado, acrescentou-se a orientação: "Analise internamente o requisito em etapas, mas não mostre o raciocínio completo; apresente somente uma justificativa resumida". Os prompts canônicos completos estão disponíveis no artefato do estudo.

3.4 Procedimento Experimental

O procedimento experimental foi aplicado igualmente a todas as estratégias avaliadas. Para cada requisito do conjunto de teste, foi construído o prompt correspondente, enviado à entrada do modelo e, posteriormente, foi registrada a resposta gerada. Em seguida, uma rotina de extração identificou o valor do campo `tipo` e o comparou à classe esperada. Quando a resposta não apresentava JSON válido, não continha o campo `tipo` ou retornava uma classe inexistente, o caso era registrado como `ERRO_PARSE`.

```
Você é um especialista em Engenharia de Requisitos.
Classifique em: vago, nao_mensuravel ou nenhuma.

Regras:
- vago: termos subjetivos ou dependentes de interpretação;
- nao_mensuravel: propriedade sem métrica, limite, prazo,
  percentual, quantidade ou critério verificável;
- nenhuma: requisito claro, objetivo e verificável;
- na dúvida, priorize nao_mensuravel quando uma métrica
  puder resolver o problema; caso contrário, use vago.

Retorne somente JSON puro com:
{
  "tipo": "...",
  "trecho_problematico": "...",
  "justificativa": "...",
  "sugestao_melhoria": "..."
}
```

Figura 1: Estrutura condensada da instrução-base utilizada no zero-shot, no few-shot e no chain-of-thought.

```
Exemplo 1: "O sistema deve ser intuitivo."
{
"tipo":"vago",
"trecho_problematico":"intuitivo",
"justificativa":"O termo é subjetivo e pode ser interpretado de formas
diferentes pelos usuários e desenvolvedores.",
"sugestao_melhoria":"O sistema deve permitir que usuários
iniciantes concluam o cadastro em até 3 minutos sem treinamento prévio."
}

Exemplo 2: "O sistema deve responder em até 2 segundos
para 95% das requisições."
{
"tipo":"nenhuma",
"trecho_problematico":"",
"justificativa":"O requisito apresenta um critério mensurável de
tempo de resposta e percentual de requisições.",
"sugestao_melhoria":""
}
```

Figura 2: Duas das quatro demonstrações fixas utilizadas na estratégia few-shot.

As saídas também foram armazenadas para análise qualitativa, permitindo verificar se a justificativa e a sugestão de melhoria eram coerentes com a classe prevista. Esse procedimento foi importante porque o modelo poderia acertar a classe, mas produzir uma justificativa genérica ou uma sugestão pouco verificável.

3.5 Métricas de Avaliação

A avaliação quantitativa considerou acurácia, acurácia balanceada, precisão, revocação, F1-score macro, F1-score ponderado, Matthews Correlation Coefficient (MCC), Cohen's Kappa, taxa de erro de parse e matriz de confusão. Essas métricas foram utilizadas para analisar tanto o desempenho geral quanto o comportamento por classe.

A matriz de confusão foi empregada para identificar os principais padrões de erro, especialmente as confusões entre `vago` e `não_mensurável`. Essa análise é relevante porque alguns requisitos podem apresentar, simultaneamente, termos subjetivos e a ausência de critérios objetivos. Como o experimento adota uma formulação de rótulo único, priorizou-se `não_mensurável` quando o problema pudesse ser resolvido pela inclusão de métrica, limite, prazo, percentual, quantidade ou outro critério verificável; quando uma métrica não fosse suficiente e a dificuldade decorresse da interpretação subjetiva, utilizou-se `vago`.

4 Resultados

Esta seção apresenta os resultados da classificação dos requisitos nas classes vago, não_mensurável e nenhuma. A avaliação considerou quatro estratégias: zero-shot, few-shot, chain-of-thought controlado e fine-tuning. O estado ERRO_PARSE foi empregado apenas nas matrizes e nos arquivos de avaliação para indicar falhas de extração ou de formatação da resposta.

Tabela 2: Métricas obtidas pelas estratégias avaliadas no teste.

Estratégia	Acurácia	Acc. Bal.	F1-macro	F1-pond.	MCC	Kappa
Fine-tuning	1,000	1,000	1,000	1,000	1,000	1,000
Few-shot	0,538	0,538	0,500	0,500	0,369	0,307
Zero-shot	0,338	0,338	0,177	0,177	0,039	0,008
CoT controlado	0,329	0,329	0,204	0,204	0,016	0,009

A Tabela 2 mostra que o modelo ajustado obteve os maiores valores em todas as métricas. Entre as abordagens sem treinamento adicional, o few-shot apresentou o melhor desempenho. Comparações pareadas pelo teste exato de McNemar indicaram vantagem significativa do few-shot sobre o zero-shot ($p < 10^{-35}$) e sobre o CoT controlado ($p < 10^{-29}$). A diferença entre zero-shot e CoT controlado não foi significativa ($p = 0{,}636$).

Na estratégia zero-shot, a Figura 3 mostra uma forte concentração das previsões na classe não_mensurável: 2.070 dos 2.100 exemplos receberam esse rótulo. A estratégia obteve acurácia de 0,338 e F1-macro de 0,177, além de sete erros de parse. O modelo reconheceu todos os exemplos realmente não mensuráveis, mas praticamente não distinguiu requisitos vagos e não identificou corretamente nenhum requisito da classe nenhuma.

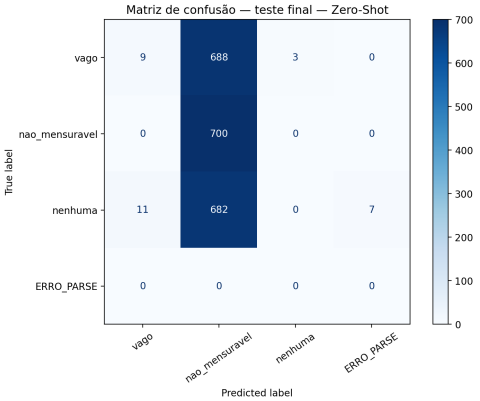

Figura 3: Matriz de confusão da estratégia zero-shot.

O few-shot alcançou acurácia de 0,538 e F1-macro de 0,500, conforme a Figura 4. As quatro demonstrações elevaram principalmente a identificação da classe nenhuma, cuja revocação foi 0,933. Entretanto, a revocação foi de 0,206 para vago e de 0,476 para não_mensurável, evidenciando que os exemplos também induziram uma tendência a classificar requisitos como objetivos. Houve um erro de parse.

O chain-of-thought controlado apresentou acurácia de 0,329 e F1-macro de 0,204. A matriz da Figura 5 evidencia concentração

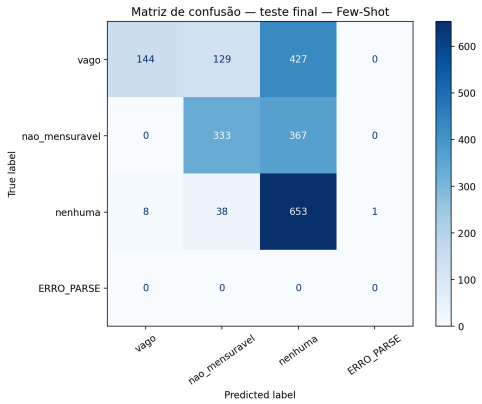

Figura 4: Matriz de confusão da estratégia few-shot.

na classe vago, prevista em 1.803 exemplos, e ausência de acertos na classe nenhuma. Além disso, foram registrados 67 erros de parse, a maior taxa entre as estratégias. Nesse cenário, a instrução de análise interna não melhorou a separação das classes e tornou a saída estruturada menos estável.

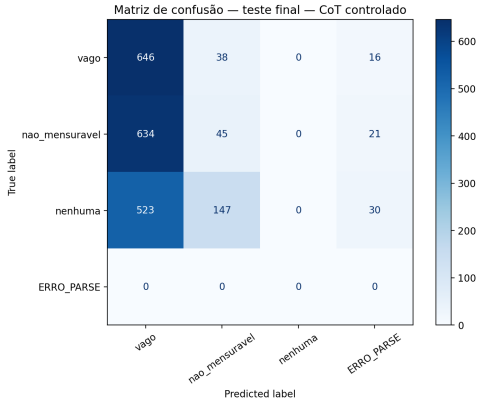

Figura 5: Matriz de confusão da estratégia de chain-of-thought controlada.

O fine-tuning apresentou os melhores resultados no conjunto avaliado, classificando corretamente os 2.100 exemplos e produzindo JSON válido em todos os casos, como ilustra a Figura 6. O resultado mostra que o ajuste supervisionado aprendeu de forma consistente a associação entre as formulações presentes no conjunto sintético, as três classes e o padrão de resposta esperado.

Entretanto, a auditoria estrutural altera a interpretação desse resultado. Não foram encontradas duplicatas exatas entre treino e teste, mas foram identificadas 13 quase duplicatas com similaridade acima de 0,92. Mais importante, 1.664 dos 2.100 exemplos de teste (79,24%) possuíam um padrão estrutural já observado no treinamento. Por exemplo, sentenças que mantêm a mesma estrutura e apenas substituem valores, módulos, papéis ou códigos permanecem lexicalmente distintas, mas oferecem ao modelo um padrão recorrente de classificação e geração. Portanto, a acurácia de 1,000

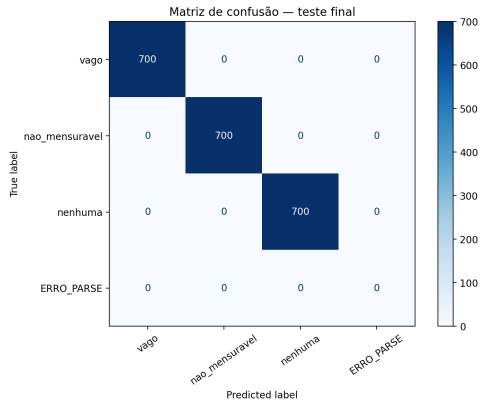

Figura 6: Matriz de confusão da estratégia com fine-tuning.

demonstra aprendizado dos padrões estruturais do conjunto sintético, porém não permite concluir que o modelo tenha generalizado semanticamente para requisitos redigidos de forma independente ou provenientes de projetos reais.

De modo geral, o few-shot foi a estratégia de prompting mais eficaz, enquanto zero-shot e CoT controlado apresentaram desempenhos próximos ao do acaso balanceado e fortes vieses de classe. O fine-tuning dominou o cenário experimental, mas sua vantagem deve ser interpretada no contexto da distribuição sintética avaliada. Uma avaliação mais rigorosa requer dados reais, exemplos gerados por fontes independentes e uma separação entre treino e teste que também impeça a recorrência dos mesmos padrões estruturais entre os conjuntos.

5 Considerações Finais

Este trabalho apresentou uma análise comparativa entre zero-shot, few-shot, chain-of-thought controlado e fine-tuning com LoRA, todas aplicadas ao modelo Gemma 3 1B Instruct para diagnosticar requisitos vagos, não mensuráveis ou sem os problemas considerados.

No conjunto sintético de teste, o fine-tuning obteve acurácia e F1-macro de 1,000, sem erros de parse. Esse resultado, contudo, não deve ser interpretado isoladamente como uma generalização semântica. A auditoria estrutural mostrou que 79,24% dos exemplos de teste compartilhavam um padrão com o treinamento. Assim, a conclusão sustentada pelos experimentos é que o modelo ajustado aprendeu de forma muito eficaz a estrutura linguística e o formato de resposta do conjunto sintético. A capacidade de analisar formulações realmente novas e requisitos provenientes de projetos reais permanece em aberto.

Entre as abordagens baseadas exclusivamente em prompting, o few-shot apresentou o melhor desempenho, com acurácia de 0,538 e F1-macro de 0,500. O zero-shot concentrou as previsões em `não_mensurável`, enquanto o CoT controlado concentrou as previsões em `vago` e produziu 67 erros de parse. Esses resultados mostram que poucas demonstrações podem melhorar um modelo pequeno, mas também evidenciam sensibilidade ao desenho dos exemplos e às regras de decisão.

Do ponto de vista metodológico, os dados sintéticos permitiram uma avaliação controlada diante da escassez de bases em português. Entretanto, os resultados também demonstram que uma divisão aleatória, mesmo sem duplicatas exatas, pode preservar estruturas recorrentes entre o treinamento e o teste e, assim, superestimar o desempenho do fine-tuning. Como trabalhos futuros, propõe-se: (i) avaliar requisitos reais anotados por especialistas; (ii) dividir os dados de modo que frases com a mesma estrutura, o mesmo assunto ou a mesma origem não apareçam ao mesmo tempo no treino e no teste; (iii) realizar testes cruzados com dados produzidos por outros modelos ou por autores humanos; (iv) avaliar modelos de maior porte; e (v) investigar uma formulação multi-label para requisitos simultaneamente vagos e não mensuráveis.

A extensão da abordagem a outros defeitos de especificação não é imediata, pois cada tipo de defeito exige uma formulação diferente da tarefa. Para detectar inconsistências, é preciso comparar requisitos entre si, e não classificar uma sentença isolada. Já a incompletude não pode ser identificada apenas pelo texto do requisito, pois depende de saber quais informações deveriam estar presentes no domínio. Ambiguidades sintáticas ou anafóricas, por sua vez, precisam ser marcadas em trechos específicos da sentença, e não na sentença como um todo. Assim, o pipeline avaliado neste trabalho é um ponto de partida, mas cobrir outros defeitos exige redefinir a tarefa e construir conjuntos de dados anotados para cada caso.

5.1 Limitações

A principal limitação deste estudo é o uso exclusivo de dados sintéticos gerados por um único modelo, o que pode reduzir a diversidade linguística e introduzir padrões artificiais. Embora o teste contenha 2.100 exemplos balanceados e não haja duplicatas exatas entre treino e teste, 79,24% dos exemplos de teste reutilizam padrões estruturais observados no treinamento e 13 pares foram identificados como quase duplicatas. Essa característica limita a validade externa do resultado perfeito do fine-tuning. Além disso, não foi realizada uma avaliação humana sistemática da qualidade das justificativas e sugestões geradas, e o problema foi formulado com um único rótulo, priorizando `não_mensurável` em casos de sobreposição. Por fim, o estudo considera apenas a vagueza e a ausência de mensurabilidade, sem abranger outras formas de ambiguidade, inconsistência ou incompletude.

Disponibilidade de Artefatos

Os notebooks das técnicas de ***prompt***, ***fine tuning*** e o arquivo do ***dataset*** utilizados estão disponíveis em: *https://github.com/Leonel-Santos1/gemma3-requirements-ambiguity-pt.git*.

Referências

[1] Simran Arora, Avner May, Jian Zhang, and Christopher Ré. 2020. Contextual Embeddings: When Are They Worth It?. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2650–2663. doi:10.18653/v1/2020.acl-main.236

[2] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Proceedings of the 34th International Conference on Neural Information Processing*

Systems. Curran Associates Inc., Vancouver, BC, Canada, 877–1901. https://arxiv.org/abs/2005.14165

[3] Celso M. de Melo, Antonio Torralba, Leonidas Guibas, James DiCarlo, Rama Chellappa, and Jessica Hodgins. 2022. Next-generation deep learning based on simulators and synthetic data. *Trends in Cognitive Sciences* 26, 2 (2022), 174–187. doi:10.1016/j.tics.2021.11.008

[4] Saad Ezzini, Sallam Abualhaija, Chetan Arora, Mehrdad Sabetzadeh, and Lionel C. Briand. 2021. Using Domain-specific Corpora for Improved Handling of Ambiguity in Requirements. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 1485–1497. doi:10.1109/ICSE43902.2021.00133

[5] Alessio Ferrari, Andrea Esuli, and Stefania Gnesi. 2018. Identification of Cross-Domain Ambiguity with Language Models. In *2018 5th International Workshop on Artificial Intelligence for Requirements Engineering (AIRE)*. IEEE, 31–38. doi:10.1109/AIRE.2018.00011

[6] Alessio Ferrari, Paola Spoletini, and Stefania Gnesi. 2016. Ambiguity and Tacit Knowledge in Requirements Elicitation Interviews. *Requirements Engineering* 21, 3 (2016), 333–355. doi:10.1007/s00766-016-0249-3

[7] Gemma Team. 2025. Gemma 3 Technical Report. *arXiv preprint arXiv:2503.19786* (2025).

[8] Vincenzo Gervasi, Alessio Ferrari, Didar Zowghi, and Paola Spoletini. 2019. Ambiguity in Requirements Engineering: Towards a Unifying Framework. In *From Software Engineering to Formal Methods and Tools, and Back*. Lecture Notes in Computer Science, Vol. 11865. Springer International Publishing, 191–210. doi:10.1007/978-3-030-30985-5_12

[9] Antonio Carlos Gil. 2002. *Como elaborar projetos de pesquisa* (4 ed.). Atlas, São Paulo.

[10] Shuang Hao, Wenfeng Han, Tao Jiang, Yiping Li, Haonan Wu, Chunlin Zhong, Zhangjun Zhou, and He Tang. 2024. Synthetic Data in AI: Challenges, Applications, and Ethical Implications. *arXiv preprint arXiv:2401.01629* (2024).

[11] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations (ICLR 2022)*. https://arxiv.org/abs/2106.09685

[12] Giuseppe Lami. 2005. *QuARS: A Tool for Analyzing Requirements*. Technical Report CMU/SEI-2005-TR-014. Software Engineering Institute, Carnegie Mellon University. doi:10.1184/R1/6582770.v1

[13] Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. 2022. Fantastically Ordered Prompts and Where to Find Them: Overcoming Few-Shot Prompt Order Sensitivity. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 8086–8098. doi:10.18653/v1/2022.acl-long.556

[14] Marina de Andrade Marconi and Eva Maria Lakatos. 2003. *Fundamentos de metodologia científica* (5 ed.). Atlas, São Paulo.

[15] Vasileios C. Pezoulas, Dimitrios I. Zaridis, Eugenia Mylona, Christos Androutsos, Kosmas Apostolidis, Nikolaos S. Tachos, and Dimitrios I. Fotiadis. 2024. Synthetic data generation methods in healthcare: A review on open-source tools and methods. *Computational and Structural Biotechnology Journal* 23 (2024), 2892–2910. doi:10.1016/j.csbj.2024.07.005

[16] Ian Sommerville. 2016. *Software Engineering* (10 ed.). Pearson.

[17] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing Systems 35 (NeurIPS 2022)*. 24824–24837. https://arxiv.org/abs/2201.11903