Avaliação de Técnicas de Engenharia de Prompt para Detecção de Dívidas Técnicas em Artefatos Textuais de Software

Ricardo Yamasaki Sassagawa
Universidade Federal do Acre
Rio Branco, AC, Brasil
ricardo.sassagawa@sou.ufac.br

Raoni Simões Ferreira
Universidade Federal do Amazonas
Manaus, AM, Brasil
raoni@icomp.ufam.edu.br

Catarina de Souza Costa
Universidade Federal do Acre
Rio Branco, Acre, Brasil
catarina.costa@ufac.br

Resumo

A identificação automática de Dívida Técnica Auto-Admitida (SATD, do inglês *Self-Admitted Technical Debt*) em artefatos textuais de software, como *issues* e mensagens de *commit*, permanece um desafio devido à linguagem informal, à ambiguidade semântica e à forte dependência de contexto desses documentos. Embora modelos baseados em *Transformers* ajustados por *fine-tuning* representem o estado da arte para essa tarefa, sua aplicação depende de grandes volumes de dados rotulados, elevado custo computacional e limitada capacidade de generalização entre projetos. Como alternativa, *Large Language Models* (LLMs) permitem explorar estratégias de engenharia de prompt para realizar a identificação de SATD sem a necessidade de retreinamento completo do modelo. Neste trabalho, realiza-se uma avaliação sistemática de diferentes LLMs empregando técnicas de engenharia de prompt, incluindo *Zero-shot*, *Few-shot*, *Chain-of-Thought* (CoT), *Tree-of-Thought* (ToT), *Prompt Tuning with Rules* (PTR) e *Soft Prompt Tuning*, comparando seus resultados com uma abordagem baseada em *fine-tuning* tradicional. Adicionalmente, propõe-se uma estratégia de *Prompt-Based Fine-Tuning* (PBFT) voltada à identificação de SATD, que adapta o modelo por meio de *prompts* especializados sem exigir o retreinamento completo de seus parâmetros. Os resultados mostram que, embora LLMs utilizadas exclusivamente com engenharia de prompt apresentem desempenho limitado na tarefa, a abordagem PBFT alcança desempenho estatisticamente equivalente ao *fine-tuning* tradicional, preservando a vantagem de reduzir o esforço de anotação de dados e o custo computacional de adaptação do modelo. Esses resultados fornecem evidências sobre as limitações das abordagens baseadas apenas em *prompt* para identificação de SATD e demonstram o potencial de estratégias híbridas para tarefas especializadas da Engenharia de Software.

Palavras-chave

Dívida Técnica, Modelos de Linguagem em Larga Escala, Engenharia de Prompt, Fine-tuning, Engenharia de Software, Classificação de Texto

1 Introdução

A Dívida Técnica (TD, do inglês *Technical Debt*) é um fator crítico na evolução de sistemas de software, estando associada ao aumento dos custos de manutenção, à degradação da qualidade e à redução da produtividade das equipes [15]. Entre suas diferentes manifestações, destaca-se a Dívida Técnica Auto-Admitida (SATD, do inglês *Self-Admitted Technical Debt*), caracterizada pelo reconhecimento explícito, por parte dos desenvolvedores, de limitações, soluções temporárias ou necessidades de melhoria registradas em artefatos textuais do processo de desenvolvimento, como *issues* e mensagens de *commit* [26].

A identificação automática de SATD permanece desafiadora devido à linguagem informal, à ambiguidade semântica e à forte dependência de contexto presentes nesses artefatos. Modelos baseados em *Transformers*, como BERT, RoBERTa e CodeBERT, alcançam resultados promissores quando submetidos ao *fine-tuning* com dados rotulados específicos do domínio [5, 6, 17]. Entretanto, essa estratégia depende de grandes volumes de dados anotados, requer elevado custo computacional para o retreinamento dos modelos e apresenta limitações de generalização entre diferentes projetos [6].

Como alternativa, *Large Language Models* (LLMs) têm demonstrado elevada capacidade de generalização por meio de *In-Context Learning*, especialmente com técnicas de engenharia de prompt, como *Zero-shot*, *Few-shot*, *Chain-of-Thought* (CoT), *Tree-of-Thought* (ToT) e *Prompt Tuning* [3, 34, 37]. Apesar desses avanços, ainda não está claro se essas abordagens são suficientes para resolver tarefas especializadas da Engenharia de Software, como a identificação de SATD, sem recorrer ao *fine-tuning* tradicional.

Este trabalho investiga essa lacuna por meio de um estudo empírico que compara LLMs sem *fine-tuning*, empregando diferentes estratégias de engenharia de prompt, com um modelo baseado em *fine-tuning* tradicional (DistilRoBERTa). Além disso, propõe uma abordagem híbrida baseada em *Prompt-Based Fine-Tuning* (PBFT), que utiliza *prompts* especializados para adaptar o modelo à identificação de SATD sem a necessidade de atualizar todos os seus parâmetros.

A avaliação experimental é guiada pelas seguintes questões de pesquisa:

- RQ1: Qual o desempenho de LLMs sem *fine-tuning* na identificação de SATD em *issues* e *commits*, em comparação com modelos pré-treinados ajustados por *fine-tuning*?
- RQ2: Diferentes técnicas de engenharia de prompt (*Zero-shot*, *Few-shot*, CoT, ToT, *Prompt Tuning with Rules* e *Soft Prompt Tuning*) melhoram significativamente o desempenho dessas LLMs?
- RQ3: Qual técnica de *prompt* apresenta melhor desempenho e em quais condições?
- RQ4: Uma abordagem baseada em PBFT reduz a lacuna entre LLMs sem *fine-tuning* e o *fine-tuning* tradicional na identificação de SATD?

Os resultados mostram que LLMs utilizadas apenas com engenharia de prompt apresentam desempenho limitado na identificação de SATD, frequentemente próximo ao acaso, independentemente da técnica empregada. Em contrapartida, a abordagem PBFT supera consistentemente essas estratégias e alcança desempenho estatisticamente equivalente ao obtido pelo DistilRoBERTa com *fine-tuning*

tradicional, preservando a vantagem de exigir menor esforço de anotação de dados e menor custo computacional.

As principais contribuições deste trabalho são: (i) uma avaliação sistemática de LLMs sem *fine-tuning* para detecção de SATD; (ii) uma comparação abrangente de diferentes estratégias de engenharia de prompt sob um protocolo experimental unificado; (iii) evidências empíricas sobre as limitações dessas abordagens em tarefas especializadas de Engenharia de Software; e (iv) a proposição de uma estratégia baseada em PBFT que mantém desempenho comparável ao *fine-tuning* tradicional, reduzindo significativamente o custo de adaptação do modelo.

2 Fundamentação Teórica

Este capítulo apresenta os conceitos e as tecnologias que fundamentam o protocolo experimental deste trabalho, estabelecendo a base teórica necessária para a análise dos resultados.

2.1 Dívida Técnica Auto Admitida (SATD)

A Dívida Técnica (TD) é o custo futuro decorrente da adoção deliberada de soluções subótimas para acelerar a entrega de software – um compromisso entre ganhos de curto prazo e custos de manutenção que tendem a crescer se a dívida não for quitada via refatoração. Essa interpretação deriva da metáfora de Cunningham [4] e foi consolidada por Brown et al. [2] e Kruchten, Nord e Ozkaya [10].

A Dívida Técnica Auto Admitida (SATD) é um subconjunto em que os próprios desenvolvedores reconhecem explicitamente limitações, soluções temporárias ou necessidades de melhoria [21]. Inicialmente estudada em comentários de código-fonte, foi posteriormente categorizada por Maldonado e Shihab [18] em dívidas de projeto, defeitos, documentação, requisitos e testes. Pesquisas recentes ampliaram o conceito ao demonstrar que a SATD também se manifesta em *issues*, mensagens de *commit* e *pull requests*, artefatos que registram decisões e restrições ao longo do ciclo de vida do software [14].

2.2 Modelos Baseados em Transformer e Fine-Tuning Tradicional

O processamento de artefatos textuais de software evoluiu significativamente com o surgimento da arquitetura *Transformer* [32]. Seu mecanismo de *self-attention* permite capturar dependências entre palavras independentemente da distância na sequência, favorecendo a construção de representações contextualizadas. Modelos baseados no codificador (*encoder*), como BERT [5] e RoBERTa [17], consolidaram-se em tarefas de classificação de texto por explorarem representações bidirecionais capazes de capturar o contexto semântico das palavras.

A adaptação desses modelos para tarefas específicas é tradicionalmente realizada por meio do *Fine-Tuning* [5], no qual um modelo pré-treinado é complementado por uma camada de classificação e treinamento supervisionado com dados rotulados do domínio de interesse. Durante esse processo, todos os parâmetros do modelo são atualizados por retropropagação, estratégia amplamente empregada na análise de artefatos de software [6, 27].

Embora essa abordagem apresente elevado desempenho por especializar o modelo às características do domínio, ela depende de grandes volumes de dados rotulados, demanda elevado poder computacional para atualização de milhões de parâmetros e tende a reduzir sua capacidade de generalização para projetos pertencentes a diferentes ecossistemas de software [6].

2.3 Large Language Models (LLMs) e Engenharia de Prompt

Diferentemente dos modelos baseados em codificadores, o cenário recente do Processamento de Linguagem Natural passou a ser dominado pelas LLMs, incluindo modelos proprietários e de pesos abertos, como LLaMA e Mistral. Esses modelos consolidaram o paradigma do *In-Context Learning* (ICL), no qual tarefas especializadas são realizadas por meio de instruções fornecidas no *prompt*, sem necessidade de atualização dos pesos do modelo [3].

No ICL, destacam-se as estratégias de *zero-shot prompting*, baseada apenas na descrição da tarefa, e *few-shot prompting*, que incorpora exemplos de entrada e saída para orientar a geração das respostas. Para tarefas mais complexas, foram propostas técnicas como CoT, que induz o modelo a explicitar etapas intermediárias de raciocínio [34], e ToT, que expande esse processo ao explorar múltiplos caminhos de raciocínio e mecanismos de autoavaliação [37].

Além dos *hard prompts*, representados por instruções em linguagem natural, a literatura introduziu os *soft prompts*, compostos por vetores contínuos aprendidos no espaço de *embeddings*. Sua otimização, denominada *Soft Prompt Tuning*, constitui uma abordagem de Ajuste Fino Eficiente em Parâmetros (PEFT), na qual os pesos da LLM permanecem congelados e apenas os parâmetros do *soft prompt* são atualizados durante o treinamento supervisionado [12].

2.4 Prompt-Based Fine-Tuning (PBFT)

O PBFT, também conhecido na literatura como aprendizado baseado em padrões (*pattern-based learning*), surgiu como um paradigma híbrido que une a semântica rica dos *prompts* discretos ao poder de otimização do *fine-tuning* clássico [8, 24]. Em vez de tratar a classificação de texto como a previsão de um rótulo abstrato (como uma classe numérica 0 ou 1) por meio de uma camada linear adicionada ao topo do modelo, o PBFT reformula o problema de classificação em uma tarefa de preenchimento de lacunas (*cloze task*) ou de geração de texto. Essa estratégia alinha a tarefa-alvo ao formato exato no qual o modelo de linguagem foi originalmente pré-treinado, reduzindo drasticamente o esforço necessário para a adaptação do modelo.

A mecânica do PBFT é estruturada sob dois componentes fundamentais: (i) template (padrão): uma função que envolve a instância de texto original e introduz uma instrução em linguagem natural contendo uma lacuna a ser preenchida (e.g., “Commit: [Texto do Commit]. Pergunta: este artefato indica uma dívida técnica? [MASK]”). (ii) verbalizer (verbalizador): uma função bijetora que mapeia as palavras de alta probabilidade geradas pelo modelo para preencher a lacuna (conhecidas como *label words*, tais como “sim”, “não”, “FIX” ou “TODO”) diretamente para os rótulos lógicos da tarefa de classificação do mundo real (SATD vs. Não-SATD).

Durante o treinamento, todos os pesos do modelo são atualizados, mas a otimização ocorre calculando a perda da entropia

cruzada sobre a probabilidade do *token* correto aparecer na posição da [MASK].

A principal vantagem teórica do PBFT reside na sua eficiência amostral (*sample efficiency*). Como o modelo resolve a tarefa especializada utilizando a mesma cabeça de predição e o mesmo comportamento linguístico do seu pré-treinamento massivo, o fenômeno do esquecimento catastrófico é mitigado. Consequentemente, o modelo requer uma quantidade substancialmente menor de dados rotulados para convergir para um desempenho competitivo, tornando-se ideal para cenários onde a escassez de dados ou o custo de anotação é um gargalo.

3 Trabalhos Relacionados

A detecção automática de SATD evoluiu de abordagens baseadas em padrões linguísticos e aprendizado supervisionado tradicional [13, 14, 26] para o uso de representações contextuais ricas com *Transformers* como BERT e RoBERTa [23, 27]. Embora estes modelos, bem como suas versões derivadas e de menor dimensões em termos de parâmetros, como o DistilRoBERTa, apresentem alto desempenho, todos enfrentam limitações de generalização e exigem numerosos conjuntos de dados rotulados.

Alternativamente, LLMs pré-treinadas e generalistas têm sido aplicados em diversas tarefas de software [20, 31], mas a identificação de SATD permanece um desafio devido à necessidade de conhecimento do desenvolvedor e do contexto do projeto embutidos nessas LLMs [6, 7]. Por outro lado, técnicas de *fine-tuning* se mostram custosas quando aplicadas neste cenário em particular. Assim, a Engenharia de Prompt surgiu como alternativa ao *fine-tuning* [16], ao deixar de lado o retreinamento, focando apenas na criação de instruções textuais inteligentes, que induza o comportamento correto do modelo na tarefa, com técnicas que incluam a instrução acompanhada por exemplos rotulados, dentre as quais destacam-se *Zero-shot*, *Few-shot*. Além das abordagens convencionais, técnicas avançadas de engenharia de prompt fundamentam-se na premissa de que a elicitação do raciocínio estruturado potencializa o desempenho das LLMs. O método CoT [34], introduz uma série de etapas intermediárias de raciocínio antes da entrega da resposta final. Estudos como os de Lambert *et al.* (2024) [11] e Sheikhaei (2025) [25] exploram essas técnicas de forma fragmentada, focando em modelos ou artefatos específicos, como comentários de código ou *issues*. sem apresentar um estudo sistemático das diferentes técnicas de *prompts* e fontes textuais com sinas de dívidas técnicas. Portanto, este trabalho diferencia-se por aplicar um protocolo unificado a múltiplos artefatos (*issues* e *commits*) e quatro famílias de LLMs, permitindo uma análise sistêmica da eficácia dessas técnicas.

Adicionalmente, este trabalho se diferencia ao investigar como mitigar as limitações das LLMs generalistas sem retreinar seus bilhões de parâmetros. A abordagem foca em: (i) identificar as estratégias de *prompt* mais eficazes para a classificação de dívida técnica; e (ii) propor uma estratégia que combine a engenharia de prompt – para a criação de instruções que induzam o comportamento desejado – com os benefícios do *fine-tuning*, permitindo que o modelo incorpore padrões específicos e discriminativos da dívida técnica ao seu conhecimento generalista pré-existente.

4 Metodologia

Esta seção descreve as abordagens empregadas, incluindo o modelo transformers pré-treinado e os modelos LLMs selecionados. Adicionalmente, é apresentado o papel e a descrição textual das instruções fornecidas para essas LLMs para cada uma das técnicas de prompts utilizadas. Todas as abordagens foram avaliadas sob os mesmos conjuntos de dados e métricas.

4.1 Conjuntos de Dados

4.1.1 Dataset de Issues. O primeiro conjunto de dados deste estudo deriva do *benchmark* proposto por Li et al. [13], que compreende originalmente 4.200 *issues* extraídas de sete projetos de código aberto. Para assegurar a validade interna dos experimentos e a compatibilidade com arquiteturas de modelos de linguagem, os dados foram submetidos a um rigoroso processo de curadoria e reestruturação, detalhado a seguir:

- **Agregação e Rotulação Binária:** as diferentes tipologias originais de dívida (e.g., *design_debt*, *requirement_debt*) foram consolidadas sob o rótulo unificado **TD (1)**. Registros classificados como *non_debt* foram mapeados para o rótulo **NTD (0)**, simplificando a tarefa para uma classificação binária de presença ou ausência de dívida técnica;
- **Consolidação de Contexto e Filtragem de Ruído:** a estrutura original fragmentava cada *issue* em seções distintas (*Summary*, *Description* e *Comments*). Realizou-se a concatenação dos campos *Summary* e *Description* em uma entrada textual única, preservando a hierarquia da informação por meio de delimitadores estruturais. Os campos de comentários foram deliberadamente descartados; a análise qualitativa revelou que tais registros frequentemente carecem de independência semântica, apresentando dependências referenciais ambíguas e discussões interpessoais que introduzem ruído estatístico, prejudicando a capacidade preditiva do modelo;
- **Amostragem Estratificada por Projeto:** visando a generalização dos resultados e a mitigação de vieses associados a padrões de escrita específicos de uma equipe, a extração das amostras foi estratificada. Isso garantiu que todos os sete projetos originais estivessem representados de forma proporcional no conjunto final de dados;
- **Balanceamento do Dataset:** dado que a distribuição original apresentava um desbalanceamento natural, aplicou-se a técnica de *random under-sampling* na classe majoritária. O resultado é um *corpus* perfeitamente balanceado (50% TD e 50% NTD), essencial para a confiabilidade das métricas de Acurácia e MCC durante a validação cruzada.

4.1.2 Dataset de Commits. O segundo conjunto de dados é derivado do *benchmark* SATDAUG [29]. Este *dataset* destaca-se por utilizar técnicas de aumento de dados sintéticos via AugGPT, expandindo amostras reais de *Self-Admitted Technical Debt* (SATD) para capturar uma maior diversidade linguística em mensagens de *commit*. O processo de adaptação para esta pesquisa seguiu o seguinte protocolo:

- **Consolidação e Padronização de Classes:** em conformidade com a metodologia aplicada ao conjunto de *issues*, as

subcategorias específicas de SATD (e.g., *requirement_debt*, *test_debt*) foram convergidas para a classe unificada **TD (1)**. Registros originais identificados como *non_debt* foram mapeados para a classe **NTD (0)**. Essa padronização permite uma análise comparativa direta entre diferentes tipos de artefatos de software sob a mesma taxonomia binária;
- **Balanceamento e Integridade dos Dados:** o conjunto de dados original do SATDAUG apresentava uma distribuição desbalanceada entre as classes. Para mitigar o risco de viés de sobreajuste (*overfitting*) à classe majoritária e assegurar a validade das métricas de correlação (MCC), aplicou-se um procedimento de *random under-sampling*. O resultado final é um *corpus* equilibrado de mensagens de *commit*, proporcionando uma base estatística robusta para os experimentos de *fine-tuning* e *prompting*.

Tabela 1: Distribuição e Composição dos Conjuntos de Dados

Artefato	Referência	# de instâncias TD	# de instâncias NTD	Total
Issues	Li *et al.* [13]	1.750	1.750	3.500
Commits	Sutoyo e Capiluppi [29]	2.047	2.047	4.094

[1] https://github.com/yikun-li/satd-issue-tracker-data
[2] https://zenodo.org/records/10521909

4.2 Abordagem LLMs Sem Fine-Tuning

Nesta abordagem, foram selecionados LLMs de pesos abertos – cujos parâmetros treinados (*weights*) são publicamente disponibilizados, permitindo download, execução local, inspeção e modificação [22, 28] – e diferentes arquiteturas, visando avaliar sua capacidade de generalização na detecção de SATD sem a necessidade de ajustes em seus parâmetros.

Os modelos foram executados localmente em modo de inferência estrita, carregados em 4-bit *NormalFloat* (NF4) com quantização dupla, visando a eficiência no uso de VRAM sem degradação severa da acurácia. As operações computacionais foram realizadas em *float16* para preservar a estabilidade numérica. A temperatura de inferência foi definida em 0, assegurando respostas determinísticas e a reprodutibilidade dos experimentos.

Os modelos selecionados abrangem diferentes paradigmas de projeto:

- Llama 3.1 8B[1]: utilizado como referência de arquitetura *Transformer* densa, treinado em um corpus massivo de *tokens*, apresentando forte capacidade de raciocínio geral [31];
- Mistral 7B (v0.3)[2]: selecionado pela sua eficiência e pelo uso de *Grouped-Query Attention* (GQA), sendo um dos modelos de 7 bilhões de parâmetros mais consolidados na literatura [9];
- Gemma 2 9B[3]: modelo desenvolvido pelo Google, que utiliza uma arquitetura com *Sliding Window Attention* e *Logit Softmax*, demonstrando alta performance em tarefas de compreensão de texto [30];
- Phi-4 Mini[4]: representante da categoria de modelos compactos (*Small Language Models*), focado em alta densidade de conhecimento e raciocínio lógico, apesar do menor número de parâmetros [1].

Para garantir uma comparação equivalente com as abordagens de *fine-tuning* e *Prompt-Based Fine-Tuning* (PBFT), os resultados das LLMs foram submetidos a um protocolo de validação cruzada (*cross-validation*) sintética.

O processo seguiu as etapas abaixo:

- Inferência Unificada: para cada combinação de modelo e técnica de *prompt* (e.g., *Zero-shot*, *Few-shot*, CoT), realizou-se uma única passagem de inferência sobre a totalidade do *dataset* de cada artefato (*issues* e *commits*);
- Fatiamento por *Folds*: o conjunto de predições gerado foi particionado em 5 *folds* estratificados, utilizando os mesmos índices de divisão empregados no treinamento dos modelos *fine-tuned* e PBFT;
- Cálculo de Métricas: para cada um dos 5 *folds*, calcularam-se de forma independente as métricas de Acurácia, Precisão, Recall, F1-Score e, primordialmente, o MCC (*Matthews Correlation Coefficient* [19]);
- Agregação: os resultados finais reportados consistem na média aritmética e no desvio padrão obtidos entre os *folds*.

Esta abordagem permite não apenas medir a performance global das LLMs, mas também validar a estabilidade das técnicas de *prompting* frente a diferentes subconjuntos de dados, permitindo a aplicação de testes de significância estatística na comparação direta com os modelos especializados.

4.3 Abordagem Transformers Pré-Treinado Com Fine-Tuning

Nesta abordagem, adotou-se o *fine-tuning* de parâmetros em um modelo de linguagem pré-treinado de arquitetura *Transformer* do tipo *Encoder*. Diferente da inferência direta, esta abordagem especializa os pesos do modelo para a tarefa alvo – a classificação de dívida técnica – permitindo que ele capture nuances semânticas e terminologias específicas do domínio de engenharia de software presentes em *issues* e *commits* [36].

Como *baseline* deste estudo, foi selecionado o modelo DistilRoBERTa. A escolha justifica-se por: (i) ser uma versão destilada do RoBERTa [17], mantendo cerca de 95% do desempenho em tarefas de compreensão de linguagem natural (GLUE) com apenas 82 milhões de parâmetros [23]; e (ii) apresentar um custo computacional reduzido de tempo e memória, facilitando execuções iterativas necessárias para validação estatística.

O processo de *fine-tuning* foi estruturado sob o protocolo de *Stratified 5-Fold Cross-Validation*, garantindo que cada exemplo dos *datasets* de *issues* e *commits* fosse utilizado tanto para treinamento quanto para avaliação. A injeção de marcadores semânticos (e.g.: [TD_INDICATOR]) não foi aplicada ao DistilRoBERTa por: (i) incompatibilidade: o *baseline* usa classificação linear via *token* [CLS], não se beneficiando de marcações locais; (ii) evitar ruído: injetar *tokens* sintéticos violaria o viés indutivo e o domínio de pré-treino do *baseline*, penalizando-o injustamente.

[1] https://llama.meta.com/
[2] https://mistral.ai/news/announcing-mistral-7b/
[3] https://ai.google.dev/gemma
[4] https://huggingface.co/microsoft/phi-4

O *dataset* original de cada artefato foi dividido em cinco subconjuntos (*folds*) mutuamente exclusivos, preservando a distribuição equilibrada das classes (50% TD e 50% Não-TD) em cada partição. O modelo foi treinado de forma independente para cada artefato. Em cada uma das cinco iterações, quatro *folds* foram utilizados para o ajuste dos pesos, enquanto o *fold* remanescente serviu como conjunto de teste cego.

Utilizou-se uma taxa de aprendizado de 2×10^{-5} com decaimento de peso (*weight decay*) de 0.01 e estratégia de parada precoce (*early stopping*) baseada no desempenho do conjunto de validação (MCC) para evitar *overfitting*. Para cada iteração do *cross-validation*, calcularam-se as métricas de Acurácia, F1-Score e o Coeficiente de Correlação de Matthews (MCC) [19]. Os resultados finais reportados na Seção 4 consistem na média aritmética e no desvio padrão dessas métricas sobre os cinco *folds*.

Como resultado, este procedimento gerou uma avaliação robusta de dois classificadores especializados (um de *issues* e outro de *commits*), permitindo quantificar não apenas a eficácia absoluta do DistilRoBERTa na detecção de TD, mas também a sua estabilidade estatística frente a diferentes amostras de dados.

4.4 Abordagem Híbrida

A abordagem híbrida proposta baseia-se no paradigma de *prompt-based fine-tuning* (PBFT), detalhado na Seção 2.4. Para sua implementação, utiliza-se o modelo ModernBERT [33] sob uma estratégia de treinamento inspirada no *framework* SciPrompt [38]. Conforme ilustrado na Figura 1, o processo mapeia a classificação binária tradicional (atribuir o rótulo TD ou NTD) para uma tarefa de *Masked Language Modeling* (MLM) [35], permitindo que o modelo capture o contexto bidirecional de termos ambíguos comuns ao domínio de software (e.g., *"hack"* ou *"workaround"*).

Diferente do modelo de *baseline* DistilRoBERTa [23], o ModernBERT destaca-se por ter sido pré-treinado nativamente para tarefas MLM em escalas massivas de código-fonte e documentação técnica, além de suportar janelas expandidas de contexto [33]. O pipeline operacional da abordagem híbrida é estruturado nos seguintes passos fundamentais:

- **Construção do *Prompt*:** cada instância de *issue* ou *commit* é inserida em um *template* discreto que introduz um *token* de mascaramento na estrutura: *"Does this issue contain technical debt? Answer: [MASK]"*;
- **Configuração do *Verbalizer*:** o mapeamento entre as predições do modelo na posição `[MASK]` e as classes lógicas do problema é realizado por um *verbalizer* customizado para o domínio de SATD. Os *tokens "yes"*, *"todo"*, *"fixme"*, *"workaround"* e *"hack"* são projetados para a classe de dívida técnica (rótulo TD), enquanto *"no"*, *"clean"* e *"done"* indicam sua ausência (rótulo NTD). A classificação final é determinada via *torch.argmax* sobre o espaço de *logits* dos *tokens* mapeados. Para conferir robustez contra variações morfológicas, integra-se um mecanismo de *fallback* baseado em similaridade de cosseno para *tokens* sintaticamente próximos não listados explicitamente;
- **Pré-processamento e Reforço Contextual:** marcadores explícitos de dívida técnica presentes no texto original (e.g., comentários textuais de *TODO*) recebem marcações especiais para enfatizar pistas contextuais. Este passo aproveita a capacidade do ModernBERT de processar contextos longos (até 8.192 *tokens*) para capturar dependências de longo alcance em discussões complexas.

O treinamento do ModernBERT foi conduzido com o otimizador AdamW, aplicando um cronograma de *warmup* linear (20% dos passos iniciais) para estabilização dos gradientes e *weight decay* para regularização. Para mitigar o excesso de confiança do modelo e melhorar a calibração das predições, utilizou-se *label smoothing*. O critério de interrupção adotado foi o *early stopping*, monitorando o Coeficiente de Correlação de Matthews (MCC) [19]. Os hiperparâmetros detalhados encontram-se na Tabela 2.

Tabela 2: Hiperparâmetros de Treinamento (Abordagem Híbrida)

Hiperparâmetro	Valor / Configuração
Otimizador	AdamW
Learning Rate	5×10^{-5}
Warmup Ratio	0.2
Weight Decay	0.01
Label Smoothing	0.1
Early Stopping Patience	3 épocas
Métrica de Monitoramento	MCC

Para validar a eficácia da abordagem e garantir a comparabilidade com as demais técnicas, o experimento utilizou o protocolo de *Stratified 5-Fold Cross-Validation*, mitigando vieses de seleção através dos seguintes critérios:

- **Particionamento Estratificado:** o *dataset* foi dividido em cinco partições independentes, preservando a proporção de 50% para cada classe em todos os *folds*. A cada iteração, quatro subconjuntos foram utilizados para o ajuste dos pesos e o restante atuou como teste;
- **Agregação de Resultados:** foram computadas a **média aritmética** e o **desvio padrão** das métricas de Acurácia, F1-Score e MCC [19] ao final do ciclo, permitindo avaliar a estabilidade do PBFT frente à variabilidade linguística dos artefatos;
- **Avaliação Baseada em Prompts:** as métricas de desempenho foram extraídas diretamente a partir da distribuição de probabilidade calculada pelo *verbalizer* na posição `[MASK]`. Considerou-se uma classificação correta quando o somatório ou o valor máximo de *logit* associado aos *tokens* da classe alvo superou o da classe oposta, conforme a lógica de inferência descrita no pipeline.

4.5 Protocolo de Validação Estatística

Para determinar se as diferenças de desempenho observadas entre as abordagens são estatisticamente significantes, aplicou-se o teste *t-Student* pareado bi-caudal com nível de significância de $\alpha = 0,05$ (95% de confiança). O teste foi conduzido em dois níveis de comparação utilizando os resultados de MCC obtidos em cada um dos 5 folds:

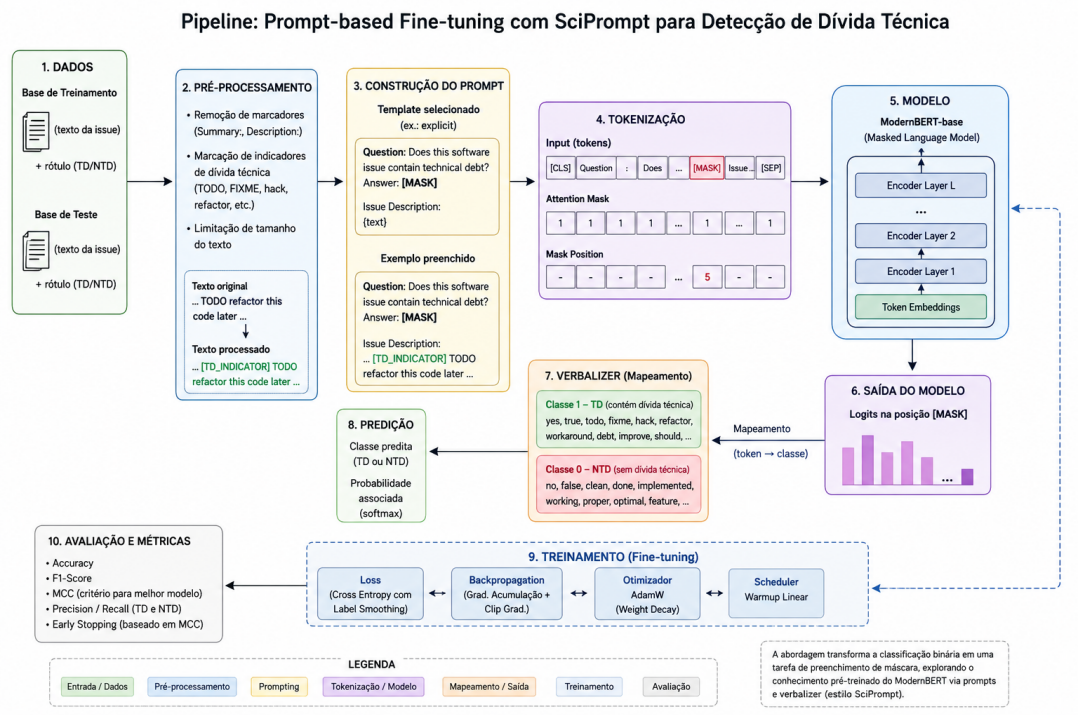

Figura 1: Diagrama do pipeline de treinamento e inferência da abordagem prompt-based fine-tuning. Fonte: Elaborado pelo autor com o auxílio de IA generativa ChatGPT (OpenAI).

- Comparação Inter-Modelos Especializados: confrontando a abordagem híbrida proposta (*PBFT*) contra o *baseline* (*DistilRoBERTa*);
- Comparação com LLMs generalistas: confrontando a abordagem híbrida proposta (*PBFT*) contra a melhor configuração identificada entre as LLMs sem *fine-tuning*.

A aplicação do teste pareado na comparação com as LLMs foi viabilizada pelo protocolo de validação cruzada sintética (Seção 4.2). Como as predições das LLMs foram fatiadas utilizando os mesmos índices de partição dos modelos treinados, os conjuntos de amostras por *fold* são diretamente dependentes, preenchendo os requisitos matemáticos para o emparelhamento dos dados.

4.6 Estratégias de Engenharia de Prompt

As estratégias de *prompt* seguiram um *template* base próprio contendo definições de TD/NTD e instruções da estrutura de saída das respostas. A Tabela 3 ilustra todas as seis estratégias avaliadas neste trabalho. Cada estratégia foi utilizada tanto para as inferências no *dataset* de *issues* quanto no de *commits*.

As Figuras 2, 4, 3, 5 e 6 apresentam a estrutura exata dos *prompts zero-shot*, *few-shot*, *chain-of-Thought* (CoT), *Tree-of-Thought* (ToT) e *Prompt tuning with Rules* (PTR) respectivamente, enviados às LLMs. Os termos entre chaves, como `{commit}` e `{issue}`, representam os espaços reservados para a inserção dinâmica dos artefatos dos *datasets*. Cabe ressaltar que a técnica de *Soft Prompt Tuning* não foi incluída nestas ilustrações por não se tratar de uma entrada textual discreta; em vez disso, ela utiliza vetores de *embeddings* aprendíveis que são injetados diretamente no espaço latente do modelo no momento da inferência.

```
You are an expert in software engineering and technical debt analysis. Your task is to classify whether a commit message indicates the presence of Technical Debt (TD) or Not Technical Debt (NTD).

Technical Debt (TD) refers to commits that: Fix bugs or errors; Refactor code; Address code smells; Improve test coverage; Update documentation; Perform code cleanup; Address performance issues; Fix security.

Not Technical Debt (NTD) includes: New features; Enhancements; Version updates; Build/config changes; Merge commits.

Analyze the following commit message {commit}

Answer with only one word: either TD or NTD.'
```

Figura 2: Template Zero-shot prompt para Commits.

```
Follow this chain-of-thought reasoning process step by step:

STEP 1: Extract and understand the issue information: {issue}

STEP 2: Identify the primary nature of the issue (Bug, Feature, Refactoring, etc.).
STEP 3: Analyze for technical debt indicators (hack, refactor, smell, workaround).
STEP 4: Distinguish from other issue types (Is it adding funcionality or improving structure?).
STEP 5: Apply the technical debt classification rules (TD if it improves maintainability; NTD if its a new feature).
STEP 6: Provide your final classification (TD or NTD).
```

Figura 3: Template Chain-of-thought (CoT) prompt para Issues.

Tabela 3: Resumo das técnicas de prompting aplicadas para avaliação das LLMs.

Técnica	Descrição	Objetivo
Zero-shot	Instrução direta sem exemplos prévios.	Avaliar o conhecimento intrínseco do modelo sobre Dívida Técnica (TD).
Few-shot	Inclusão de exemplos rotulados no *prompt*.	Fornecer um padrão de referência para prover contexto semântico ao modelo na classificação de casos ambíguos.
CoT	*Chain-of-Thought* (Cadeia de Pensamento).	Induzir o raciocínio lógico passo a passo antes da decisão final.
ToT	*Tree of Thoughts* (Árvore de Pensamentos).	Explorar múltiplos caminhos de raciocínio em paralelo para problemas complexos.
PTR	*Prompt Tuning with Rules*.	Aplicar regras determinísticas e restrições semânticas rígidas na instrução.
Soft Prompt	Otimização de vetores de *embeddings*.	Ajustar parâmetros contínuos do *prompt* via gradientes (abordagem não textual).

```
You are a software engineering expert.

Technical debt occurs when developers intentionally or unintentionally introduce suboptimal solutions that
should be improved later.

Example 1:
Issue:
"Summary: Beeline output doesn t align properly when accessing impala
Description: When beeline is accessing the impala, the output is misaligned."
Answer: TD

Example 2:
Issue:
"Summary: Speed up a couple of heavy-hitting expr-tests
Description: Two tests ( LongReverse and the base64 tests in run their tests over all lengths from 0.. some
length . Both take several minutes to complete. This adds a lot of runtime for not much more confidence. If
instead we pick a set of interesting (including powers-of-two, prime numbers, edge-cases) lengths, we can get a
similar amount of confidence while significantly reducing the runtime of expr-test."
Answer: TD

Example 3:
Issue:
"Summary: Restlet when used as client doesn t use the configured SSL properties
Description: We re missing configuration options needed for the Restlet client to use the configured SSL
properties."
Answer: NTD

Example 4:
Issue:
"Summary:
12690: Edit-button is enabled on submitted changes
Description: The buttons to publish the edit or to leave the edit mode in the change screen are not displayed. To
exit the edit-mode the URL has to be manually changed."
Answer: NTD

Classify the following software issue: {issue}
Answer with only one word: either TD or NTD.
```

Figura 4: Template Few-shot prompt para Issues.

```
Task: Software Technical Debt Classification
Analyze the following software commit and determine if it contains Technical Debt (TD) or No Technical Debt
(NTD).

COMMIT CONTENT: {commit}

Execute a multi-perspective analysis by exploring the following "thoughts":

Thought 1 (Maintainability & Complexity):
- Does this commit introduce code that is difficult to maintain?
- Does it add unnecessary complexity or "quick-and-dirty" temporary solutions?

Thought 2 (Explicit Technical Debt Indicators):
- Does the code or message contain keywords like "workaround", "hack", "temporary", "fixme", "todo", or
"shippable for now"?
- Is there evidence of intentional technical debt?

Thought 3 (Design Quality & Architecture):
- Does it improve the architecture (refactoring) or increase tight coupling?
- Are necessary tests included, or is stability being bypassed?

Thought 4 (Strategic Context):
- Does this solve an immediate problem while compromising long-term sustainability?
- Is this a robust feature implementation or a fragile patch?

Consolidate the findings from all perspectives above. Evaluate the trade-offs and reach a final conclusion.

Provide the classification strictly in the following format:
CLASSIFICATION: [TD or NTD]
```

Figura 5: Template Tree-of-thought (ToT) prompt para Commits.

```
You are an expert in detecting technical debt using strict rules.

Analyze the issue using these rules:

R1: describes refactoring or improving existing code structure
R2: addresses a known workaround or temporary solution
R3: aims to reduce code complexity or improve maintainability
R4: updates outdated patterns or dependencies for long-term health
R5: mentions paying down technical debt explicitly
R6: improves testability or adds missing tests for existing code
R7: purely adds new features without touching existing code (negative signal)
R8: pure bug without design issue (negative signal)

For each rule:
- 1 if present
- 0 if absent

Decision:
If any of R1–R6 = 1 → TD (override everything)
Else if any of R7–R8 = 1 → NTD
Else → NTD

Issue: {issue_text}

Return exactly ONE JSON.
Do not repeat answers.
Do not provide multiple alternatives.
```

Figura 6: Template Prompt Tuning with Rules (PTR) para Issues.

5 Resultados e Discussão

Nesta seção, são apresentados e discutidos os resultados experimentais obtidos, organizados pelas questões de pesquisa que norteiam este estudo. O desempenho global dos classificadores foi sumarizado na Tabela 4 com base nas métricas de Acurácia, F1-Score e o Coeficiente de Correlação de Matthews (MCC) [19]. As métricas de Precisão e Recall, embora omitidas da tabela por restrições de espaço, são integradas à discussão qualitativa dos resultados. Adotou-se o MCC como métrica primária de comparação devido à sua robustez em cenários de classificação binária sob complexidade semântica.

Para garantir o rigor científico e validar se as diferenças de desempenho observadas são estatisticamente significantes, os vetores de MCC obtidos ao longo dos cinco *folds* (tanto das abordagens treinadas quanto do fatiamento sintético das LLMs) foram submetidos ao teste *t-Student* pareado bi-caudal com nível de significância de $\alpha = 0,05$.

5.1 Desempenho de LLMs vs. Modelos Fine-tuned (RQ1)

RQ1: *Qual o desempenho de LLMs sem fine-tuning na identificação de dívida técnica em issues e commits, em comparação com modelos fine-tuned?*

A RQ1 investiga a eficácia de modelos generalistas em modo de inferência *zero-shot/few-shot* comparados a modelos especializados via ajuste de parâmetros. Como demonstrado na Tabela 4, as abordagens baseadas exclusivamente em engenharia de prompt apresentam desempenho sistematicamente inferior às supervisionadas (*DistilRoBERTa* e *ModernBERT*).

Os resultados evidenciam a insuficiência do *prompting* puro para a identificação de Dívida Técnica Auto-Admitida (SATD). Essa abordagem carece da especialização necessária para lidar com a ambiguidade terminológica inerente aos artefatos de software. Para diversos cenários de *prompting*, o MCC médio aproximou-se de zero (e.g., Llama e Mistral com *Few-Shot* em *Commits*), equiparando as predições a um classificador aleatório.

Em contraste, os modelos submetidos a ajuste de pesos demonstraram superioridade expressiva. No *dataset* de *commits*, o *baseline* *DistilRoBERTa* alcançou um MCC de 0, 847, enquanto a nossa abordagem híbrida baseada em *ModernBERT* (*PBFT*) atingiu a marca de 0, 859. O teste *t-Student* pareado confirmou que a superioridade das abordagens de *fine-tuning* sobre a melhor LLM sem ajuste (Gemma-CoT) é estatisticamente significante ($p < 0,001$) para ambos os artefatos. Esses achados comprovam que o conhecimento especializado incorporado via *fine-tuning* é indispensável para capturar as sutilezas semânticas do domínio.

5.2 Impacto das Técnicas de Engenharia de Prompt (RQ2)

RQ2: *Diferentes técnicas de engenharia de prompt (zero-shot, few-shot, Chain-of-Thought, Tree-of-Thought, Prompt Tuning with Rules e soft prompt tuning) melhoram significativamente o desempenho de LLMs sem fine-tuning na identificação de TD?*

A RQ2 avalia se o design estruturado de *prompts* pode mitigar a ausência de *fine-tuning*. A Tabela 4 detalha as métricas alcançadas. Observa-se que, embora técnicas avançadas como *Chain-of-Thought* (CoT) e *Prompt Tuning with Rules* (PTR) melhorem o desempenho basal dos LLMs se comparadas ao *Zero-Shot*, esse ganho não é robusto o suficiente para competir com os modelos especializados.

Mesmo sob estratégias que induzem cadeias de raciocínio lógico (CoT), as LLMs de pesos abertos permaneceram limitadas a uma zona de predição de baixa qualidade (MCC $\leq 0,261$). Ademais, técnicas como *Tree-of-Thought* (ToT) e *Soft Prompt Tuning* falharam em convergir para predições úteis em diversos modelos, resultando em MCCs nulos.

O teste *t-Student* pareado revelou que as flutuações de desempenho entre as diferentes técnicas de *prompting*, em sua maioria, não apresentam diferença estatisticamente significante entre si, evidenciando que a engenharia de prompt, isoladamente, não viabiliza o uso de LLMs generalistas para esta tarefa. O ajuste de parâmetros orientado ao domínio (*fine-tuning*) permanece como o único caminho estatisticamente comprovado para uma classificação de alta confiabilidade.

5.3 Comparação entre Técnicas de Prompt (RQ3)

RQ3: *Entre as técnicas de prompt avaliadas, qual apresenta melhor desempenho geral e sob quais condições?*

A RQ3 busca identificar qual estratégia de interação via ICL é mais promissora para o domínio de SATD. A Tabela 5 sintetiza a melhor configuração de *prompt* identificada para cada modelo nos dois conjuntos de dados.

- **Predomínio do Chain-of-Thought (CoT):** o CoT apresentou a maior consistência e robustez geral. Ele alcançou os melhores resultados com o modelo Gemma 2 em ambos os cenários (MCC de 0, 214 em *issues* e 0, 261 em *commits*), além de liderar no Llama e Mistral para *commits*.;
- **Degradação por Excesso de Complexidade:** abordagens multi-passo extremamente complexas, como o *Tree-of-Thought* (ToT), resultaram em severa degradação de performance, chegando a MCC nulo ou negativo. Esse fenômeno decorre do fato de o ToT induzir múltiplas ramificações de decisão que, em tarefas de classificação binária de software, introduzem ruído interpretativo e inconsistência lógica no *verbalizer*.

Além do predomínio do CoT, os resultados revelam uma estreita relação entre a arquitetura do modelo, a técnica de *prompt* e a natureza do artefato. Enquanto o Gemma demonstra maior capacidade de generalização puramente baseada em raciocínio (CoT) nos dois contextos, modelos como Llama e Mistral performaram melhor em *issues* quando apoiados por exemplos ou regras explícitas (*Few-shot* e PTR). Isso sugere que o ruído linguístico e a informalidade típicos de discussões em *issues* exigem referências externas e ancoragem sintática que o raciocínio endógeno do CoT, isoladamente, não consegue suprir sob condições de inferência quantizada (4-bit).

Por outro lado, observa-se uma superioridade sistemática no *dataset* de *Commits*. Por possuírem uma estrutura inerentemente mais técnica, concisa e padronizada, os *commits* facilitam a decomposição lógica e a extração de fatos pelo CoT. Isso resulta em maior estabilidade (menor desvio padrão) e correlação (maiores valores de MCC) em comparação às *issues*.

Em suma, não há uma técnica universal, mas o CoT oferece o melhor equilíbrio para a tarefa. O fato de o aumento de complexidade do *prompt* (como ToT) degradar a performance indica que o gargalo do ICL reside na falta de conhecimento intrínseco do modelo sobre o domínio de engenharia de software, e não na sofisticação da instrução.

5.4 Eficácia da Abordagem Híbrida (RQ4)

RQ4: *Uma abordagem híbrida baseada em prompt-based fine-tuning reduz a lacuna entre LLMs sem fine-tuning e o fine-tuning tradicional?*

Por fim, a RQ4 analisa se a abordagem híbrida, que combina os benefícios conceituais de *prompting* e *fine-tuning* por meio da integração SciPrompt + ModernBERT, é capaz de mitigar as limitações das abordagens isoladas. Os resultados comparativos gerais estão consolidados na Tabela 6.

Os resultados demonstram que a abordagem híbrida (PBFT) alcançou os maiores valores nominais de MCC em ambos os conjuntos de dados (0,397 em *Issues* e 0,859 em *Commits*). Para avaliar a significância estatística das diferenças observadas em relação à *baseline* de *fine-tuning* tradicional (DistilRoBERTa), foi conduzido um Teste *t*

Tabela 4: Comparação Completa de Abordagens Prompting e Fine-Tuning: Datasets de Issues vs. Commits (RQ1 e RQ2). Os valores representam a média ± desvio padrão obtidos via validação cruzada em 5 folds (sintética para LLMs).

MODELO / LLM	TÉCNICA DE ANÁLISE	DATASET DE ISSUES			DATASET DE COMMITS		
		ACUR.	F1	MCC	ACUR.	F1	MCC
Gemma	PROMPT - ZERO-SHOT	0.55 ± 0.02	0.59 ± 0.02	0.109 ± 0.041	0.50 ± 0.00	0.67 ± 0.00	0.052 ± 0.014
	PROMPT - FEW-SHOT	0.55 ± 0.01	0.24 ± 0.03	0.171 ± 0.045	0.51 ± 0.00	0.67 ± 0.00	0.076 ± 0.018
	PROMPT - CoT	**0.61 ± 0.02**	**0.63 ± 0.02**	**0.214 ± 0.040**	**0.62 ± 0.02**	**0.61 ± 0.03**	**0.261 ± 0.011**
	PROMPT - ToT	0.50 ± 0.01	0.51 ± 0.00	0.000 ± 0.000	0.41 ± 0.03	0.29 ± 0.03	0.000 ± 0.000
	PROMPT - PTR	0.54 ± 0.02	0.48 ± 0.01	0.128 ± 0.011	0.44 ± 0.01	0.44 ± 0.01	−0.120 ± 0.002
	PROMPT - SOFT PROMPT	0.51 ± 0.01	0.67 ± 0.02	0.059 ± 0.002	0.50 ± 0.00	0.35 ± 0.00	0.000 ± 0.000
Llama	PROMPT - ZERO-SHOT	0.51 ± 0.01	0.51 ± 0.01	0.020 ± 0.001	0.50 ± 0.00	0.33 ± 0.00	0.000 ± 0.000
	PROMPT - FEW-SHOT	0.51 ± 0.01	0.51 ± 0.01	0.025 ± 0.002	0.50 ± 0.00	0.33 ± 0.00	0.000 ± 0.000
	PROMPT - CoT	0.50 ± 0.00	0.33 ± 0.00	−0.035 ± 0.003	**0.61 ± 0.01**	**0.59 ± 0.02**	**0.250 ± 0.010**
	PROMPT - ToT	0.50 ± 0.00	0.33 ± 0.00	0.000 ± 0.000	0.46 ± 0.00	0.45 ± 0.01	−0.081 ± 0.020
	PROMPT - PTR	**0.57 ± 0.02**	**0.56 ± 0.03**	**0.142 ± 0.024**	0.52 ± 0.01	0.52 ± 0.02	0.033 ± 0.002
	PROMPT - SOFT PROMPT	0.50 ± 0.00	0.67 ± 0.01	0.000 ± 0.000	0.50 ± 0.00	0.35 ± 0.00	0.000 ± 0.000
Mistral	PROMPT - ZERO-SHOT	0.58 ± 0.01	0.58 ± 0.01	0.162 ± 0.022	0.50 ± 0.00	0.34 ± 0.00	0.016 ± 0.001
	PROMPT - FEW-SHOT	**0.58 ± 0.01**	**0.56 ± 0.02**	**0.171 ± 0.021**	0.50 ± 0.00	0.33 ± 0.00	0.000 ± 0.000
	PROMPT - CoT	0.50 ± 0.00	0.39 ± 0.01	0.080 ± 0.002	**0.61 ± 0.01**	**0.61 ± 0.02**	**0.228 ± 0.002**
	PROMPT - ToT	0.50 ± 0.00	0.33 ± 0.00	0.000 ± 0.000	0.45 ± 0.02	0.44 ± 0.03	−0.105 ± 0.010
	PROMPT - PTR	0.50 ± 0.00	0.33 ± 0.00	0.000 ± 0.000	0.47 ± 0.00	0.47 ± 0.00	−0.060 ± 0.002
	PROMPT - SOFT PROMPT	0.48 ± 0.00	0.48 ± 0.00	−0.035 ± 0.011	0.50 ± 0.00	0.35 ± 0.00	0.000 ± 0.001
Phi	PROMPT - ZERO-SHOT	0.49 ± 0.00	0.49 ± 0.00	0.000 ± 0.000	0.50 ± 0.00	0.33 ± 0.00	0.000 ± 0.000
	PROMPT - FEW-SHOT	0.51 ± 0.01	0.50 ± 0.01	0.010 ± 0.001	0.50 ± 0.00	0.33 ± 0.00	0.000 ± 0.000
	PROMPT - CoT	**0.56 ± 0.01**	**0.54 ± 0.02**	**0.129 ± 0.042**	0.50 ± 0.00	0.33 ± 0.00	0.000 ± 0.000
	PROMPT - ToT	0.50 ± 0.00	0.33 ± 0.00	0.000 ± 0.000	0.49 ± 0.00	0.39 ± 0.00	−0.003 ± 0.001
	PROMPT - PTR	0.52 ± 0.01	0.41 ± 0.02	0.082 ± 0.010	**0.56 ± 0.02**	**0.41 ± 0.01**	**0.087 ± 0.012**
	PROMPT - SOFT PROMPT	0.50 ± 0.01	0.67 ± 0.02	0.000 ± 0.002	0.50 ± 0.0	0.35 ± 0.00	0.000 ± 0.000
DistilRoBERTa	FINE-TUNING	0.69 ± 0.02†	0.68 ± 0.03†	0.376 ± 0.033†	0.92 ± 0.00†	0.92 ± 0.01†	0.847 ± 0.020†
ModernBERT (Ours)	**PBFT (SciPrompt)**	**0.71 ± 0.01**†	**0.71 ± 0.01**†	**0.417 ± 0.013**†	**0.93 ± 0.01**†	**0.93 ± 0.01**†	**0.859 ± 0.018**†
	PBFT (Ablation)	**0.70 ± 0.01**†	**0.70 ± 0.01**†	**0.397 ± 0.018**†	**0.93 ± 0.01**†	**0.93 ± 0.01**†	**0.859 ± 0.022**†

† Indica diferença estatisticamente significante ($p < 0,05$) em relação ao melhor resultado obtido pelas LLMs sem *fine-tuning* (Gemma-CoT).

Tabela 5: Melhor técnica de prompt por modelo nos conjuntos de dados de Issues e Commits (RQ3)

	Issues				Commits			
Modelo	**Melhor Prompt**	**Acc**	**F1**	**MCC**	**Melhor Prompt**	**Acc**	**F1**	**MCC**
Gemma	CoT	0.61 ± 0.02	0.63 ± 0.02	0.214 ± 0.040	CoT	0.62 ± 0.02	0.61 ± 0.03	0.261 ± 0.011
Llama	PTR	0.57 ± 0.02	0.56 ± 0.03	0.142 ± 0.024	CoT	0.61 ± 0.01	0.59 ± 0.02	0.250 ± 0.010
Mistral	Few-shot	0.58 ± 0.01	0.56 ± 0.02	0.171 ± 0.021	CoT	0.61 ± 0.01	0.61 ± 0.02	0.228 ± 0.002
Phi	CoT	0.56 ± 0.01	0.54 ± 0.02	0.129 ± 0.042	PTR	0.56 ± 0.02	0.41 ± 0.01	0.087 ± 0.012

Tabela 6: Comparação de performance das abordagens nos conjuntos de dados de Issues e Commits (RQ4)

	Issues			Commits		
Abordagem	**Acc**	**F1**	**MCC**	**Acc**	**F1**	**MCC**
LLM + Prompt (Gemma + CoT)	0.61 ± 0.02	0.63 ± 0.02	0.214 ± 0.040	0.62 ± 0.02	0.61 ± 0.03	0.261 ± 0.011
DistilRoBERTa (Fine-tuning)	0.69 ± 0.02	0.68 ± 0.03	0.376 ± 0.033	0.92 ± 0.00	0.92 ± 0.01	0.847 ± 0.020
Híbrido (ModernBERT c/ PBFT)	**0.69 ± 0.01**	**0.69 ± 0.01**	**0.397 ± 0.030**	**0.93 ± 0.01**	**0.93 ± 0.01**	**0.859 ± 0.023**

de *Student* Pareado bicaudal ($\alpha = 0,05$) com Intervalo de Confiança de 95% (IC95%) sobre os valores de MCC obtidos ao longo dos 5 *folds*. O teste indicou que a sutil superioridade nominal do modelo híbrido não possui significância estatística, configurando um cenário de

equivalência quantitativa entre o PBFT e o *fine-tuning* tradicional. No *dataset* de *Issues*, obteve-se $t = 1,9607$ ($p = 0,1214$), enquanto no *dataset* de *Commits* o resultado foi $t = 1,6426$ ($p = 0,1758$). Como $p \geq 0,05$ em ambos os cenários, confirma-se que a estruturação do conhecimento via *prompt* durante o ajuste dos pesos atinge o mesmo teto preditivo da especialização por camada linear clássica.

Apesar da equivalência estatística nos indicadores de performance, a abordagem híbrida proposta (PBFT) sustenta sua superioridade através de critérios práticos de viabilidade e engenharia de software, divididos em dois pilares essenciais:

- **Flexibilidade e Manutenibilidade em Produção:** o *fine-tuning* tradicional acopla uma camada de classificação rígida ao *Transformer*, exigindo retreinamento ao mudar rótulos. Já o PBFT utiliza um *Verbalizer Enhanced* com similaridade de cosseno nos *logits* do vocabulário, permitindo incorporar novos termos de dívida técnica (ex.: *"workaround"* e *"hack"*) via *fallback*, sem necessidade de coletar novos dados ou reexecutar o retreinamento;
- **Infraestrutura e Escalabilidade Industrial:** apesar do desempenho estatístico similar em *Commits* (MCC de $\sim 0,86$ vs. $\sim 0,84$), a abordagem híbrida adota o ModernBERT, garantindo vantagens para pipelines de CI/CD. O suporte nativo a *Flash Attention* reduz o custo de inferência, enquanto a janela de até 8.192 *tokens* permite analisar históricos longos de *issues* sem o truncamento severo do DistilRoBERTa.

Em suma, as evidências empíricas revelam que a detecção de SATD permanece uma tarefa complexa e com desafios de generalização entre ecossistemas. Contudo, a abordagem híbrida baseada em PBFT consolida-se como a alternativa mais promissora para o ambiente corporativo. Ela entrega o rigor e a assertividade de um modelo especializado de Engenharia de Software, mas mitiga de forma drástica o ônus operacional de anotação massiva de dados e a rigidez arquitetural dos classificadores lineares tradicionais.

6 Ameaças à Validade

Apesar do rigor metodológico, este estudo apresenta limitações classificadas em quatro dimensões:

- **Validade Interna:** a sensibilidade das LLMs à formulação sintática das instruções foi mitigada pela padronização rigorosa do protocolo de *prompts* e pela configuração da temperatura em 0, garantindo o determinismo e a reprodutibilidade dos experimentos;
- **Validade de Construção:** a modelagem da detecção de SATD como uma classificação binária simplifica um fenômeno inerentemente multidimensional. Adicionalmente, há o risco de subjetividade herdado dos processos de anotação manual realizados por terceiros nas bases originais;
- **Validade Externa:** o uso de *datasets* balanceados de *issues* e *commits* pode não representar a distribuição real e altamente desbalanceada de dívida técnica na indústria, o que pode influenciar a capacidade de generalização do modelo em diferentes linguagens e ecossistemas;
- **Validade de Conclusão:** para mitigar o impacto do comportamento estocástico das LLMs e blindar as conclusões, adotou-se o protocolo de *Stratified 5-Fold Cross-Validation*. Isso permitiu calcular a média e o desvio padrão das métricas de Acurácia, F1-Score e MCC, assegurando uma base estatística robusta para as comparações.

7 Conclusão

Este trabalho investigou a eficácia de LLMs sem *fine-tuning* na identificação de Dívida Técnica Auto-Admitida (SATD) em *issues* e *commits*, comparando diferentes estratégias de engenharia de *prompt* com abordagens baseadas em *fine-tuning* tradicional e uma estratégia híbrida de *Prompt-Based Fine-Tuning* (PBFT).

Os resultados mostram que, embora técnicas como *Chain-of-Thought* promovam melhorias em relação a outras estratégias de *prompt*, LLMs utilizadas exclusivamente por meio de engenharia de *prompt* apresentam desempenho limitado para a identificação de SATD. Em contrapartida, a abordagem híbrida proposta reduz essa limitação ao alcançar desempenho estatisticamente equivalente ao *fine-tuning* tradicional, mantendo a vantagem de exigir menor esforço de adaptação do modelo.

Esses resultados fornecem evidências de que estratégias baseadas apenas em engenharia de *prompt* ainda não são suficientes para tarefas especializadas da Engenharia de Software, enquanto abordagens híbridas representam uma alternativa promissora para cenários com escassez de dados anotados e restrições de custo computacional.

Como trabalhos futuros, pretende-se incorporar informações estruturais dos projetos, como grafos de dependência e fluxos de chamada, avaliar a abordagem proposta em bases de dados mais diversas e desbalanceadas, investigar cenários de classificação multiclasse de SATD e analisar sua capacidade de generalização em diferentes ecossistemas de software e linguagens de programação.

Disponibilidade de Artefatos

Para fomentar a transparência, a reprodutibilidade e a auditabilidade dos resultados apresentados neste trabalho, todos os artefatos da pesquisa — incluindo os conjuntos de dados curados (*issues* e *commits*), os códigos-fonte de treinamento e inferência, os hiperparâmetros de configuração e as instruções de execução — foram disponibilizados publicamente no repositório Zenodo sob o DOI permanente: 10.5281/zenodo.20030758.

Vale ressaltar que a técnica de *Soft Prompt Tuning* foi omitida do conjunto de arquivos de *prompt* textuais por utilizar parâmetros contínuos (*embeddings*) em vez de *tokens* discretos de texto, o que impossibilita uma representação direta em linguagem natural. No entanto, a sua lógica de otimização e o fluxo de inferência matemática podem ser integralmente verificados e replicados por meio dos *scripts* de implementação em Python contidos no próprio repositório.

Agradecimentos

A ilustração técnica apresentada na Figura 1 foi gerada utilizando-se a ferramenta de IA generativa ChatGPT (OpenAI), baseada em esquemas conceituais fornecidos pelo autor, visando a clareza visual e a padronização terminológica do estudo, conforme as diretrizes de submissão desta trilha.

Referências

[1] Abdelrahman Abouelenin, Atabak Ashfaq, Adam Atkinson, Hany Awadalla, Nguyen Bach, Jianmin Bao, Alon Benhaim, Martin Cai, Vishrav Chaudhary, Congcong Chen, et al. 2025. Phi-4-mini technical report: Compact yet powerful multimodal language models via mixture-of-loras. *arXiv preprint arXiv:2503.01743* (2025), 39 pages.

[2] Nicolas Brown, Yuanfang Cai, Yuepu Guo, Rick Kazman, Miryung Kim, Philippe Kruchten, Erin Lim, Alan MacCormack, Robert Nord, Ipek Ozkaya, et al. 2010. Managing technical debt in software-reliant systems. *Proceedings of the FSE/SDP workshop on Future of software engineering research* (2010).

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[4] Ward Cunningham. 1992. The WyCash portfolio management system. In *Addendum to the proceedings on Object-oriented programming systems, languages, and applications (OOPSLA)*. Addendum to the Proceedings of OOPSLA '92.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*. 4171–4186.

[6] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. Codebert: A pre-trained model for programming and natural languages. In *Findings of the association for computational linguistics: EMNLP 2020*. 1536–1547.

[7] Cuiyun Gao, Xing Hu, Shan Gao, Xin Xia, and Zhi Jin. 2025. The current challenges of software engineering in the era of large language models. *ACM Transactions on Software Engineering and Methodology* 34, 5 (2025), 1–30.

[8] Tianyu Gao, Adam Fisch, and Danqi Chen. 2021. Making Pre-trained Language Models Better Few-shot Learners. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (Eds.). Association for Computational Linguistics, Online, 3816–3830. doi:10.18653/v1/2021.acl-long.295

[9] Albert Q. Jiang et al. 2023. Mistral 7B. *arXiv preprint arXiv:2310.06825* (2023).

[10] Philippe Kruchten, Robert L Nord, and Ipek Ozkaya. 2012. Technical debt: From metaphor to theory and practice. *IEEE software* 29, 6 (2012), 18–21.

[11] Pedro Lambert, Lucila Ishitani, and Laerte Xavier. 2024. On the identification of self-admitted technical debt with large language models. In *Simpósio Brasileiro de Engenharia de Software (SBES)*. SBC, 651–657.

[12] Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691* (2021).

[13] Yikun Li, Mohamed Soliman, and Paris Avgeriou. 2022. Identifying self-admitted technical debt in issue tracking systems using machine learning. *Empirical Software Engineering* 27, 6 (2022), 131.

[14] Yikun Li, Mohamed Soliman, and Paris Avgeriou. 2023. Automatic identification of self-admitted technical debt from four different sources. *Empirical Software Engineering* 28, 3 (2023), 65.

[15] Zengyang Li, Paris Avgeriou, and Peng Liang. 2015. A systematic mapping study on technical debt and its management. *Journal of systems and software* 101 (2015), 193–220.

[16] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM computing surveys* 55, 9 (2023), 1–35.

[17] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

[18] Everton da S Maldonado and Emad Shihab. 2015. Detecting and quantifying different types of self-admitted technical debt. In *2015 IEEE 7Th international workshop on managing technical debt (MTD)*. IEEE, 9–15.

[19] Brian W Matthews. 1975. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure* 405, 2 (1975), 442–451.

[20] OpenAI. 2023. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774* (2023).

[21] Aniket Potdar and Emad Shihab. 2014. An exploratory study on self-admitted technical debt. In *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 91–100.

[22] Ananda Rimal and Adarsha Rimal. 2026. Benchmarking Linguistic Adaptation in Comparable-Sized LLMs: A Study of Llama-3.1-8B, Mistral-7B-v0. 1, and Qwen3-8B on Romanized Nepali. *arXiv preprint arXiv:2604.14171* (2026).

[23] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108* (2019).

[24] Timo Schick and Hinrich Schütze. 2021. Exploiting cloze-questions for few-shot text classification and natural language inference. In *Proceedings of the 16th conference of the European chapter of the association for computational linguistics: main volume*. 255–269.

[25] Mohammad Sadegh Sheikhaei. 2025. *Automated Self-Admitted Technical Debt Tracking, Classification, and Repayment for Sustainable Software Development*. Queen's University (Canada).

[26] Mohammad Sadegh Sheikhaei and Yuan Tian. 2023. Automated Self-Admitted Technical Debt Tracking at Commit-Level: A Language-independent Approach. In *2023 ACM/IEEE International Conference on Technical Debt (TechDebt)*. IEEE, 22–26.

[27] Karthik Shivashankar, Mili Orucevic, Maren Maritsdatter Kruke, and Antonio Martini. 2025. BEACon-TD: Classifying Technical Debt and its types across diverse software projects issues using transformers. *Journal of Systems and Software* 226 (2025), 112435.

[28] Stanford Human-Centered AI Institute. 2023. What is an Open-Weight Model? https://hai.stanford.edu/ai-definitions/what-is-an-open-weight-model. Accessed: 2026-04-03.

[29] Edi Sutoyo and Andrea Capiluppi. 2024. Satdaug-a balanced and augmented dataset for detecting self-admitted technical debt. In *Proceedings of the 21st International Conference on Mining Software Repositories*. 289–293.

[30] Gemma Team, Morgane Rivière, Shreya Pathak, Robert Dadaneh, et al. 2024. Gemma 2: Improving Open Language Models via Information Density. arXiv:2408.00118 [cs.LG]

[31] Hugo Touvron, Thibaut Lavril, Gautier Izacard, et al. 2023. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[33] Benjamin Warner, Antoine Chaffin, Benjamin Clavié, Orion Weller, Oskar Hallström, Said Taghadouini, Alexis Gallagher, Raja Biswas, Faisal Ladhak, Tom Aarsen, Griffin Thomas Adams, Jeremy Howard, and Iacopo Poli. 2025. Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (Eds.). Association for Computational Linguistics, Vienna, Austria, 2526–2547. doi:10.18653/v1/2025.acl-long.127

[34] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903* (2022).

[35] Alexander Wettig, Tianyu Gao, Zexuan Zhong, and Danqi Chen. 2023. Should you mask 15% in masked language modeling?. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*. 2985–3000.

[36] Xiao-Kun Wu, Min Chen, Wanyi Li, Rui Wang, Limeng Lu, Jia Liu, Kai Hwang, Yixue Hao, Yanru Pan, Qingguo Meng, et al. 2025. Llm fine-tuning: Concepts, opportunities, and challenges. *Big Data and Cognitive Computing* 9, 4 (2025), 87.

[37] Shunyu Yao et al. 2023. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601* (2023).

[38] Zhiwen You, Kanyao Han, Haotian Zhu, Bertram Ludaescher, and Jana Diesner. 2024. SciPrompt: Knowledge-augmented Prompting for Fine-grained Categorization of Scientific Topics. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 6087–6104. doi:10.18653/v1/2024.emnlp-main.350