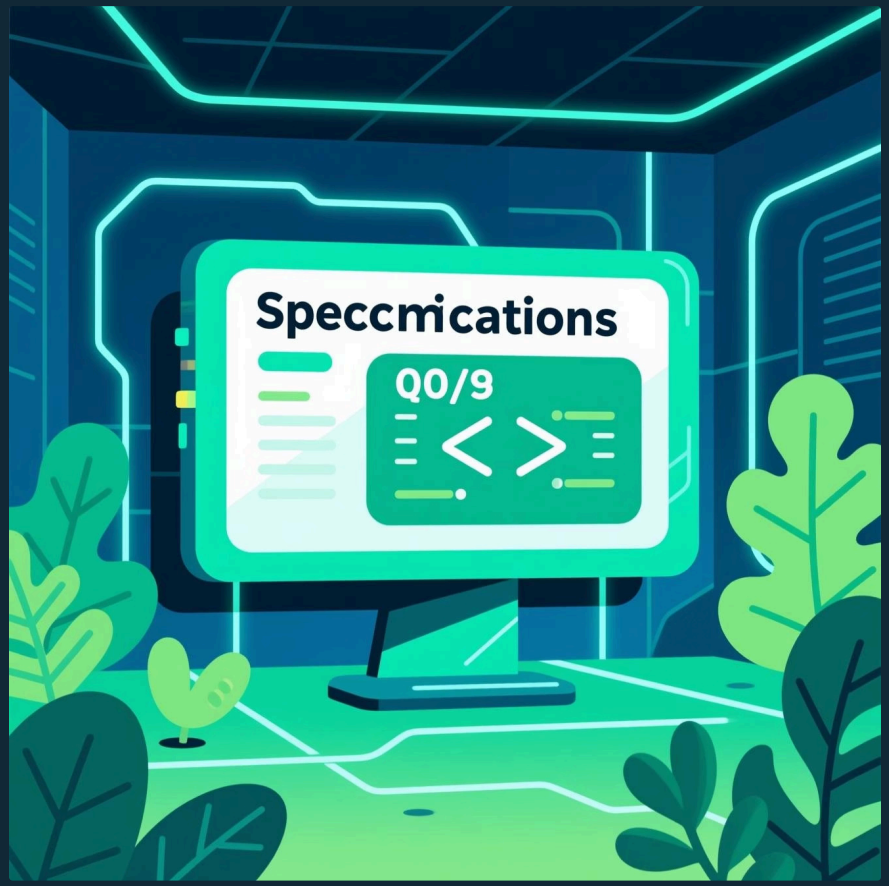
Eficácia de Large Language Models na Geração de Especificações de Software

Este estudo apresenta uma avaliação abrangente sobre a capacidade de 13 Large Language Models (LLMs) de gerar especificações formais de software a partir de comentários e documentação técnica. A pesquisa explora uma fronteira crítica na automação de engenharia de software, investigando se modelos pré-treinados podem substituir ou complementar métodos tradicionais baseados em regras e heurísticas específicas de domínio.

A Importância Crítica das Especificações Formais



Especificações precisas, expressas como pré-condições e pós-condições, constituem a espinha dorsal da garantia de qualidade em sistemas de software modernos. Elas são fundamentais para:

- **Corretude e Confiabilidade:** Garantem que o software se comporte conforme esperado em todas as condições operacionais
- **Detecção Proativa de Bugs:** Permitem identificar defeitos antes da execução em produção
- **Execução Simbólica:** Habilitam análises formais que exploram caminhos de execução sistematicamente
- **Test Oracles Eficazes:** Fornecem critérios objetivos para validação de testes automatizados

Exemplo Prático: A API tf.nn.max_pool3d do TensorFlow requer explicitamente que o parâmetro *input* seja um "5-D Tensor" - uma especificação que previne erros de tipo e dimensionalidade.

O Desafio da Generalização nos Métodos Tradicionais

Métodos Baseados em Regras

Ferramentas como Jdoctor dependem de conjuntos rígidos de padrões sintáticos e heurísticas pré-definidas para extrair especificações de documentação.

Abordagens de ML Clássico

Técnicas de aprendizado de máquina tradicional requerem grandes volumes de dados anotados manualmente e features específicas do domínio.

Métodos de Busca

Algoritmos de busca exploram o espaço de especificações possíveis, mas enfrentam explosão combinatória e limitações de escalabilidade.

A Limitação Crítica: Generalização Insuficiente

A maioria dos métodos existentes sofre de **generalização limitada**, apresentando desempenho degradado quando confrontados com APIs, domínios ou padrões de documentação não vistos durante o treinamento ou desenvolvimento de regras. Essa dependência de heurísticas específicas de domínio e dados anotados manualmente cria um gargalo significativo na automação em larga escala.

> A emergência dos Large Language Models, pré-treinados em vastos corpus de código e documentação técnica, oferece uma via promissora para superar essas limitações através de Few-Shot Learning (FSL), permitindo generalização a partir de um número limitado de exemplos in-context.

Metodologia de Avaliação e Datasets Utilizados

Modelos Avaliados e Benchmark

Esta pesquisa conduziu uma avaliação sistemática de **13 LLMs state-of-the-art**, incluindo modelos open-source e comerciais. O modelo **CodeLlama-13B** foi selecionado como benchmark primário para as Research Questions 1-3, devido à sua natureza open-source, desempenho competitivo e suporte robusto a prompts longos (**16.384 tokens**), essencial para incluir múltiplos exemplos in-context.

Datasets Públicos de Especificações

Jdoctor-data

854 anotações de pré-condições e pós-condições extraídas de comentários Javadoc em projetos Java open-source.

DocTer-data

2.876 anotações de especificações de APIs de Deep Learning (TensorFlow, PyTorch, MXNet), incluindo restrições de dtype, shape e outros atributos tensoriais.

CallMeMaybe-data

89 anotações de restrições temporais complexas que especificam ordenação e dependências entre chamadas de métodos.

Estratégias de Prompt Engineering

Random Retrieval (RR)

Seleção aleatória de K exemplos do conjunto de treinamento para composição do prompt in-context. Estabelece a baseline de desempenho.

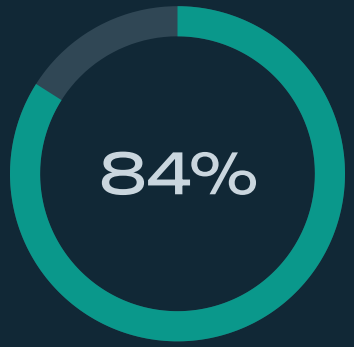
Semantic Retrieval (SR)

Seleção de exemplos semanticamente similares ao contexto alvo usando embeddings, maximizando a relevância dos exemplos fornecidos ao modelo.

RQ1: Desempenho Básico com Random Retrieval

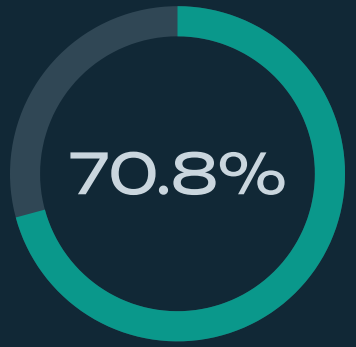
Achado 1: Competitividade Inesperada dos LLMs

LLMs utilizando **10 a 60 exemplos selecionados aleatoriamente** alcançam resultados surpreendentemente competitivos quando comparados a métodos tradicionais especializados, demonstrando capacidade de generalização mesmo sem curadoria cuidadosa dos exemplos in-context.



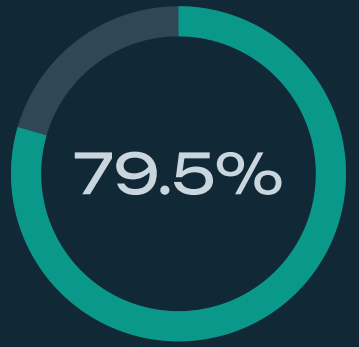
Jdoctor-data

CodeLlama-13B (K=20) após correção manual, **superando Jdoctor** em 1.0%



CallMeMaybe-data

CodeLlama-13B (K=60), **superando CallMeMaybe** em 0.8%



DocTer-data (F1)

CodeLlama-13B (K=60), apenas **2.1% abaixo** de DocTer (81.6%)

Análise Comparativa

Os resultados revelam que LLMs, mesmo com estratégia de seleção básica (aleatória), conseguem:

- **Superar métodos baseados em regras** (Jdoctor, CallMeMaybe) em 0.8% a 4.3%
- **Aproximar-se significativamente** de métodos híbridos mais sofisticados (DocTer)
- **Escalar efetivamente** com o aumento do número de exemplos in-context

Estes achados sugerem que o conhecimento pré-treinado em código e documentação técnica confere aos LLMs uma vantagem fundamental em tarefas de extração de especificações, mesmo sem otimização de prompt.

RQ2: Maximizando a Eficácia com Semantic Retrieval

Achado 2

Semantic Retrieval Amplifica Dramaticamente o Desempenho

A estratégia de **Semantic Retrieval (SR)** representa um divisor de águas na eficácia dos LLMs para geração de especificações. Ao selecionar exemplos semanticamente relevantes ao contexto alvo - em vez de exemplos aleatórios - observamos melhorias substanciais e consistentes em todos os datasets avaliados.

93.5%

Acurácia Jdoctor-data

CodeLlama-13B + SR (K=60), representando melhoria de **10.5%** sobre baseline Jdoctor

10

Exemplos Suficientes

Apenas 10 exemplos bem selecionados por tipo/categoria superam todos os métodos baseline

1.9-10.5%

Gap Ampliado

Vantagem sobre métodos tradicionais expandida significativamente com SR

Vantagem Estratégica

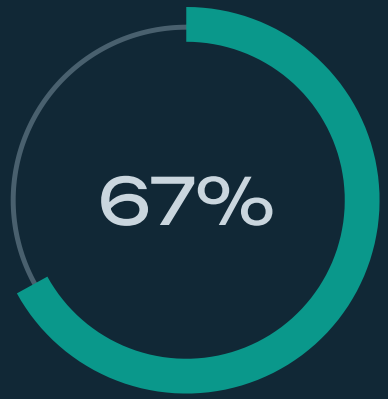
O LLM + SR não apenas supera métodos tradicionais, mas o faz com **eficiência de dados notável**: usando apenas 10 exemplos cuidadosamente selecionados por categoria ou tipo de especificação, alcança desempenho superior a técnicas que dependem de milhares de exemplos anotados ou conjuntos elaborados de regras. Esta eficiência sugere que a qualidade e relevância contextual dos exemplos supera significativamente a quantidade bruta de dados.

> "A seleção semântica de exemplos transforma o Few-Shot Learning de uma técnica promissora em uma abordagem genuinamente competitiva e prática para automação de especificações em larga escala."

RQ3: Diagnóstico de Falhas - Sintomas Distintos

Achado 3: Padrões de Falha Revelam Forças e Fraquezas Únicas

Sintoma Dominante: Métodos Tradicionais



Especificações Vazias

Falha mais comum nos baselines baseados em regras

Causa Fundamental: Ocorre quando regras ou heurísticas aplicáveis estão *ausentes* do conjunto de padrões pré-definidos. O método simplesmente não reconhece o padrão de documentação e retorna vazio.

Sintomas Dominantes: LLMs

41%

Especificações Incompletas

Perda de tokens durante decodificação

28%

Especificações Malformadas

Sintaxe incorreta ou estrutura inválida

Natureza Generativa: LLMs usam amostragem probabilística para decodificação, podendo resultar em especificações incompletas ou com formatação incorreta - são **8% mais propensos** a gerar saídas incompletas que métodos determinísticos.

Insight Complementar: Enquanto métodos tradicionais falham por *falta de cobertura* (regras insuficientes), LLMs falham por *imperfeições na geração* (saídas incompletas ou malformadas). Essa diferença fundamental sugere que abordagens híbridas podem ser particularmente promissoras, combinando a cobertura dos LLMs com validação estrutural.

RQ3: Diagnóstico de Falhas - Causas Raiz dos LLMs

Causa Raiz Dominante nos Métodos Tradicionais

Missing Rule (Regra Faltante)

Responsável por **93%** das falhas únicas do baseline Jdoctor, expondo dramaticamente a limitação fundamental de métodos dependentes de conjuntos fixos de regras e padrões sintáticos.

Achado 4: Duas Causas Dominam 76% das Falhas em LLMs

1

Missing Domain Knowledge

50% das Falhas Únicas

O LLM carece do contexto específico necessário para gerar especificações corretas. Exemplos incluem:

- **Métodos inexistentes:** Gerar especificações que referenciam funções não disponíveis na classe ou biblioteca
- **Tipos incompatíveis:** Sugerir tipos de dados que não existem no framework alvo
- **Constraints inválidos:** Propor restrições que violam as capacidades reais da API

Esta categoria revela a *lacuna de conhecimento contextual* - o LLM possui conhecimento geral de programação, mas falta-lhe acesso ao estado completo do projeto ou biblioteca específica.

2

Ineffective Prompts

26% das Falhas Únicas

A seleção, formulação ou **ordem dos exemplos no prompt** é crucial para o desempenho. Descobertas notáveis:

- **Sensibilidade à ordem:** 21% dessas falhas foram resolvidas *apenas* rearranjando a sequência dos exemplos in-context
- **Qualidade dos exemplos:** Exemplos ambíguos ou mal formulados contaminam a geração
- **Relevância contextual:** Exemplos semanticamente distantes confundem o modelo

Este achado destaca a *engenharia de prompt como ciência* - pequenas mudanças na estrutura do prompt podem ter impactos desproporcionalmente grandes no resultado.

A dominância dessas duas causas (76% combinadas) fornece um mapa claro para melhorias futuras: enriquecer LLMs com conhecimento contextual específico do projeto e desenvolver técnicas mais sofisticadas de construção e otimização de prompts.

RQ4: Comparação de Modelos e Custo-Efetividade

Achado 6: Ampla Competitividade dos LLMs Avaliados

A avaliação sistemática dos **13 LLMs** revela um cenário encorajador: a maioria demonstra desempenho melhor ou comparável às técnicas tradicionais especializadas. Especificamente, **13 modelos superaram Jdoctor**, **10 superaram DocTer** e **9 superaram CallMeMaybe**, evidenciando que a eficácia não é restrita a modelos específicos, mas uma capacidade emergente da classe LLM.

Achado 7: Campeões Open-Source

CodeLlama-13B

Desempenho superior, **custo $0**, suporte a 16K tokens de contexto. Ideal para deployment em infraestrutura própria.

StarCoder2-15B

Competitividade equivalente, treinado em código multilíngue, **licença permissiva** para uso comercial.

Ambos oferecem a combinação ideal de **desempenho, controle e custo-efetividade**, eliminando dependências de APIs comerciais e permitindo customização completa.

Achado 8: Modelos Comerciais

$32.8

Custo GPT-4

Apenas no dataset CallMeMaybe (pequeno)

GPT-4 e GPT-3.5 oferecem conveniência de API e facilidade de integração, mas **sem ganhos de acurácia ou F1** superiores aos melhores modelos open-source. Considerações:

- **Custos operacionais:** Substanciais em escala de produção
- **Latência:** Dependência de chamadas de rede
- **Privacidade:** Dados enviados para serviços externos
- **Controle:** Limitado sobre versões e comportamento do modelo

Recomendação Prática: Para organizações com expertise em ML e infraestrutura adequada, modelos open-source como CodeLlama-13B e StarCoder2-15B representam a escolha ótima, oferecendo desempenho state-of-the-art com controle total, custo zero de licenciamento e independência de fornecedores externos.

A Força Fundamental dos LLMs: Generalização Superior

Achado 5

Generalização como Vantagem Competitiva Decisiva

A principal força dos Large Language Models não reside em sua capacidade de memorizar regras ou padrões, mas em demonstrar **excelente generalização** - a habilidade de fazer previsões precisas baseadas na compreensão holística da descrição de entrada, em vez de depender de conjuntos limitados de regras sintáticas ou heurísticas específicas de domínio.

Superando Limitações Estruturais dos Métodos Tradicionais

Problema: Regras Insuficientes

Métodos baseados em regras falham quando confrontados com padrões de documentação não cobertos por suas heurísticas pré-definidas - resultando em especificações vazias.

Solução LLM: Capacidade de inferir especificações a partir de contexto semântico, mesmo para padrões nunca vistos explicitamente durante treinamento.

Problema: Regras Incorretas

Regras geradas automaticamente podem conter erros que se propagam sistematicamente, como observado em 10% das falhas únicas do DocTer.

Solução LLM: CodeLlama-13B conseguiu **resolver essas falhas**, demonstrando capacidade de superar erros sistemáticos através de raciocínio contextual.

Entrada: Descrição Natural

Documentação técnica em linguagem natural, possivelmente com padrões diversos ou não padronizados

Compreensão Contextual

LLM analisa semanticamente, infere intenção, contextualiza com conhecimento pré-treinado

Especificação Válida

Geração de pré/pós-condições formalmente corretas, mesmo para casos não vistos

> "Esta capacidade de generalização transforma LLMs de ferramentas auxiliares em componentes fundamentais para automação de especificações, especialmente em ecossistemas de software heterogêneos e em constante evolução."