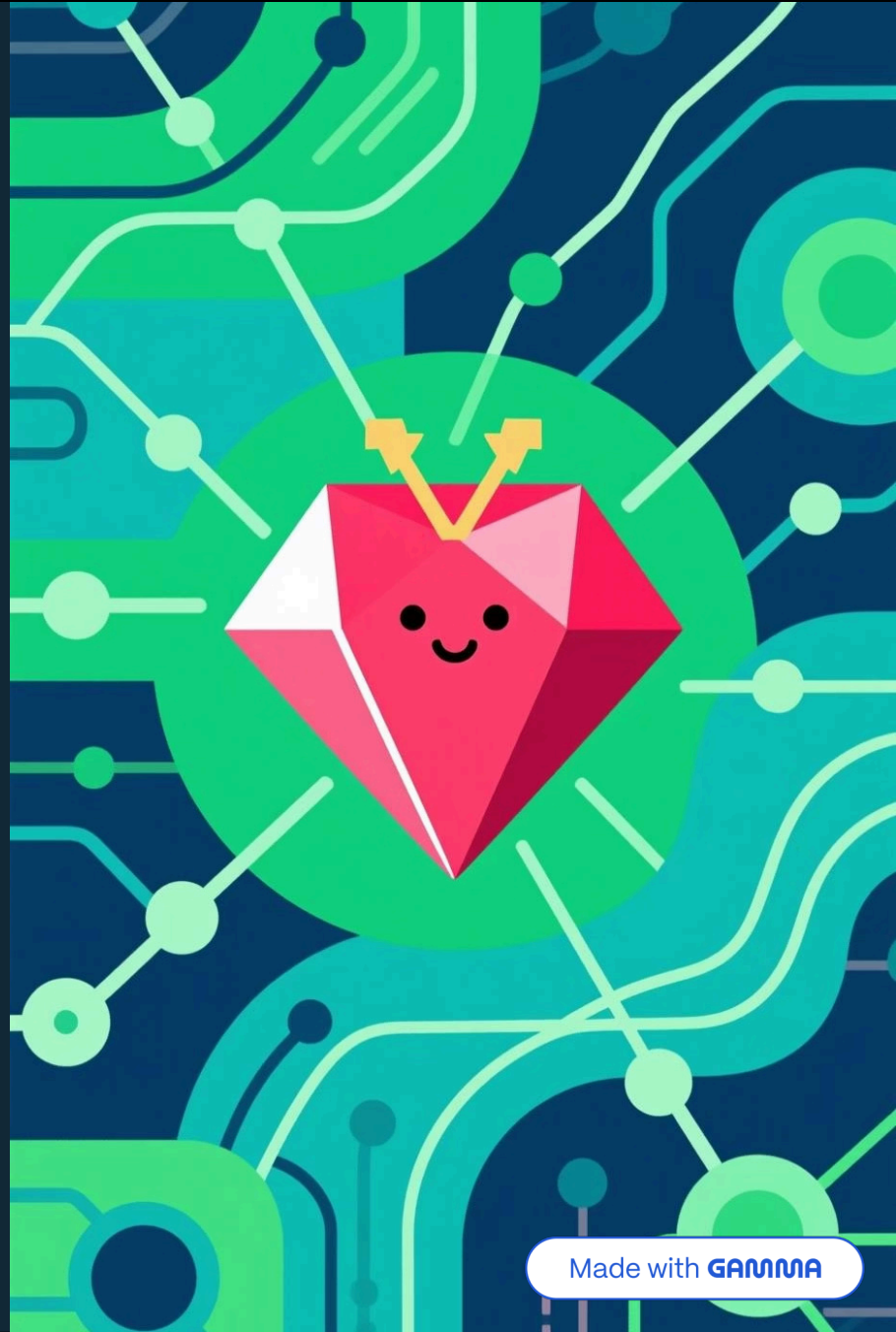
Extração Automatizada de Restrições de Parâmetros em Ruby com Suporte de LLMs e MNLI

Estudo inspirado no artigo *How Effective Are Large Language Models in Generating Software Specifications?*, aplicado em código Ruby real de projetos profissionais. Dataset com 286 métodos documentados via YARD.



Motivação

O Problema

Ruby não possui ferramentas equivalentes ao JDoctor para validar documentação de forma automatizada. Existe uma lacuna significativa entre a documentação escrita e a implementação real do código, que pode levar a inconsistências e bugs difíceis de detectar.

A alta variabilidade dos estilos de código Ruby e a natureza dinâmica da linguagem exigem uma abordagem mais contextual e inteligente para análise de documentação.

Nossa Solução

Objetivo: gerar condições Ruby automaticamente para validar tags `@param`, `@return` e `@raise` presentes na documentação YARD.

Aplicações práticas:

- Validar consistência entre documentação e código
- Criar contratos dinâmicos em tempo de execução
- Automatizar análises estáticas em bases de código legadas
- Detectar divergências antes da produção

Construção do Dataset

01

Seleção de Projetos

Dataset criado a partir de projetos Ruby reais nos quais trabalho profissionalmente. Critério principal: possuir ao menos documentação mínima usando YARD.

02

Extração de Métodos

Total coletado: **286 métodos** documentados, incluindo assinatura completa, tipos de parâmetros, tags YARD e comentários descritivos.

03

Categorização

Métodos organizados por classes representativas: ListUtils (operações numéricas), JSONParser (parse/stringify) e ImageProcessor (operações em imagens).

Cada método inclui informações completas: assinatura, tipos de parâmetros e retorno, tags YARD detalhadas e comentários contextuais que explicam o comportamento esperado.

Exemplo Real de Classes do Dataset

ListUtils

Operações numéricas em arrays

```
class ListUtils
  # @param numbers [Array]
  #   must be an array
  # @return [Numeric]
  #   sum of all numbers
  def sum(numbers)
  end
end
```

JSONParser

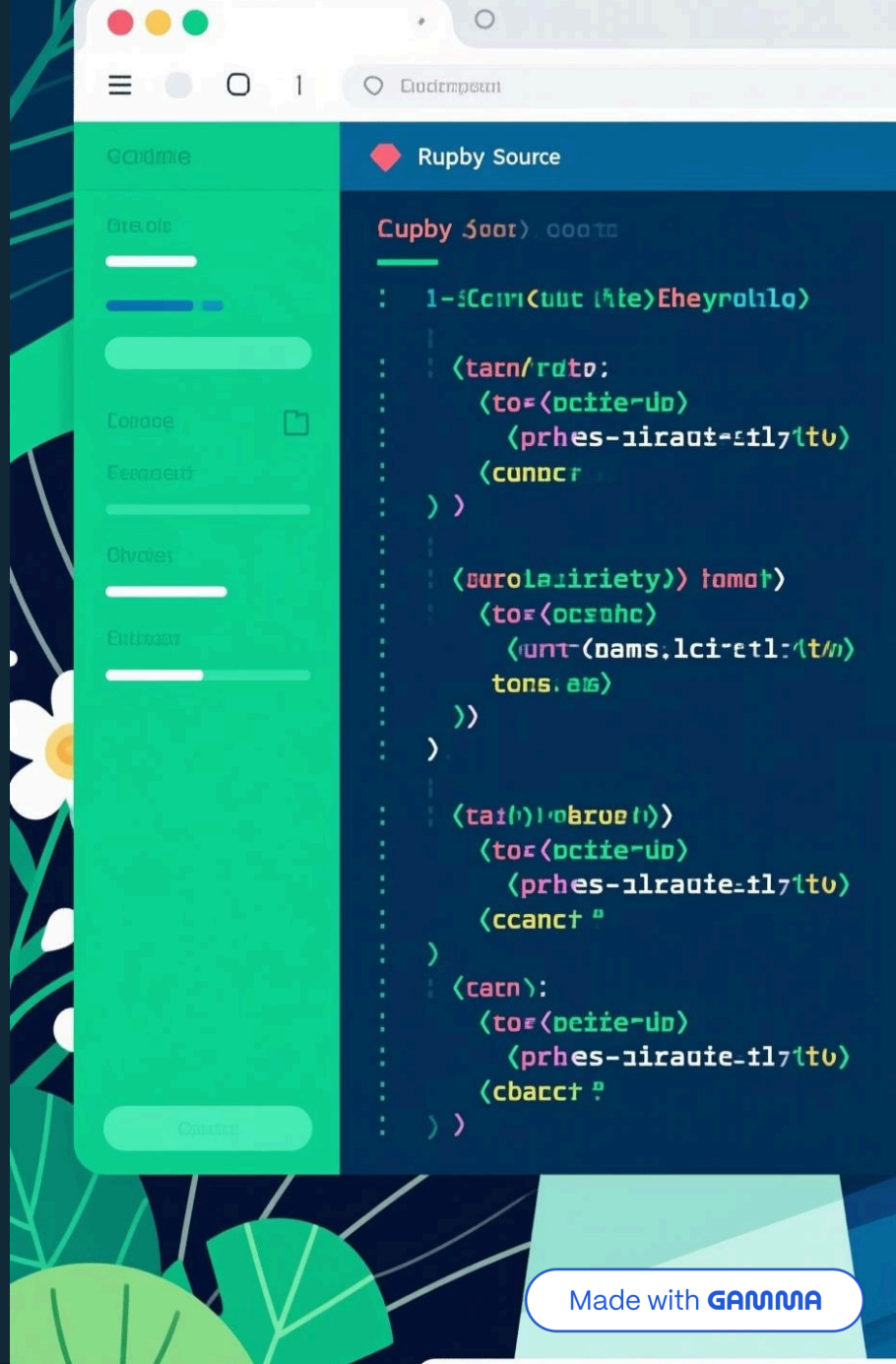
Parsing e serialização JSON

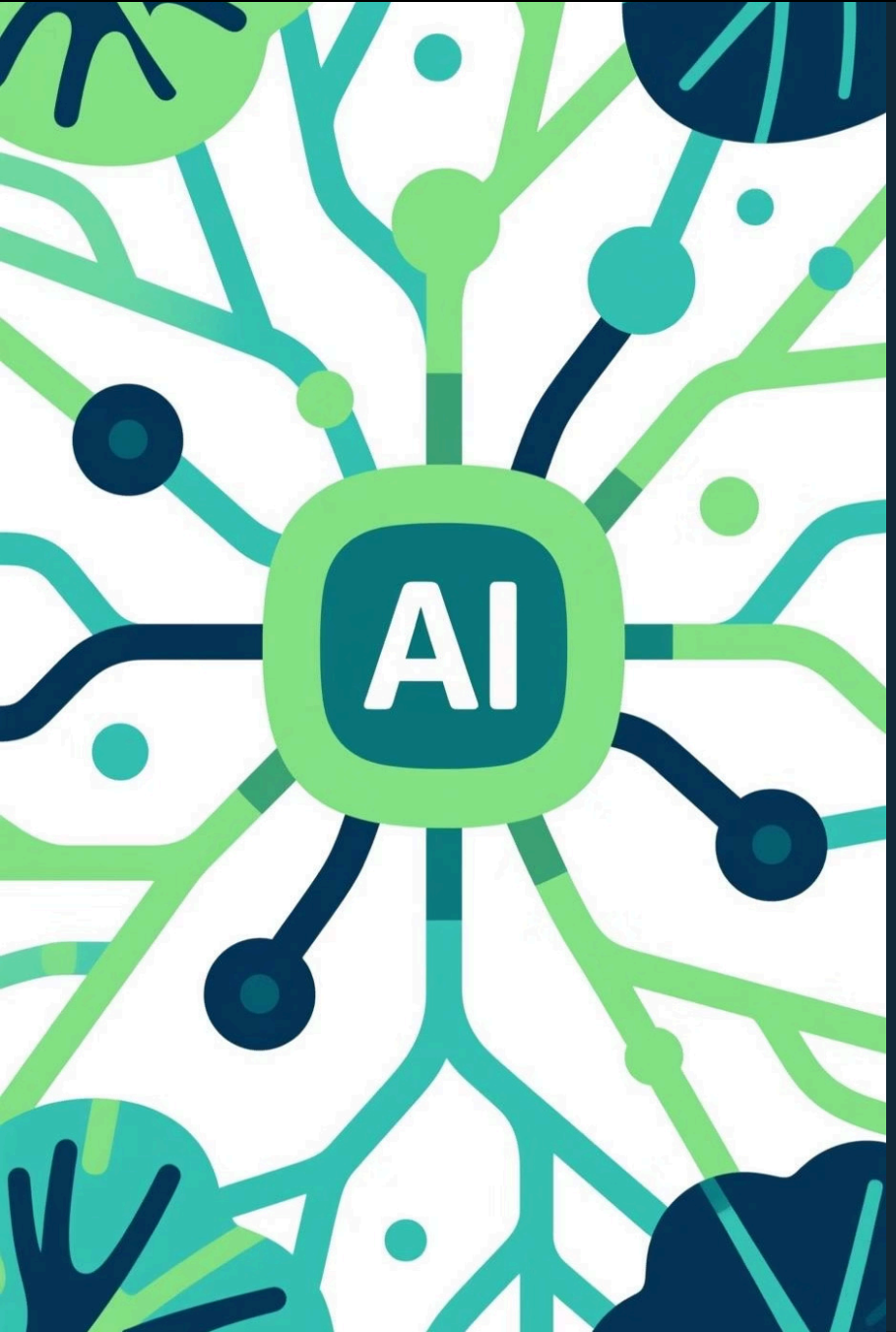
```
class JSONParser
  # @param json [String]
  # @return [Object]
  def parse(json)
  end
end
```

ImageProcessor

Processamento de imagens

```
class ImageProcessor
  # @param width [Integer]
  # must be > 0
  # @return [Boolean]
  def resize(width, height)
  end
end
```





Estratégia da Solução

Few-Shot Learning

Abordagem inspirada no paper original, mas adaptada especificamente para Ruby. Utilizamos exemplos mais semanticamente semelhantes ao método-alvo para guiar o modelo.

Seleção Inteligente de Exemplos

Seleção de exemplos via **SentenceTransformer** com similaridade de cosseno, garantindo que os exemplos fornecidos ao LLM sejam os mais relevantes para o contexto.

Prompt Enriquecido

Prompt completo incluindo: classe, método, assinatura completa, tipo de retorno, lista de parâmetros com tipos, e comentário original YARD.

LLM Utilizado

Modelo: **Ministral-3-14B** via Ollama Cloud, escolhido pelo equilíbrio entre performance e custo computacional.

Inovação: Avaliação com RoBERTa-MNLI

Limitação do Método Original

O paper original usa apenas **igualdade exata** entre expressões para avaliar resultados. Para Ruby, isso causa muitos **falsos negativos**, pois expressões semanticamente equivalentes podem diferir sintaticamente devido à natureza expressiva da linguagem.

Nossa Solução

Introduzimos classificação NLI (Natural Language Inference) usando o modelo **roberta-large-mnli** para avaliar três categorias:

- **Entailment** — expressões equivalentes
- **Neutral** — expressões mais restritivas mas válidas
- **Contradiction** — expressões contraditórias (rejeitadas)

Esta abordagem aceita expressões equivalentes e mais restritivas, reduzindo significativamente a necessidade de análise manual e aumentando a precisão da avaliação.

Exemplos de Falhas Corrigidas pelo MNLI

Caso 1 — Validação de Código MFA

Ground Truth

```
mfa_code.match?(/^[0-9]{6}$/)
```

Predição do LLM

```
mfa_code.is_a?(String) &&
mfa_code.length == 6 &&
mfa_code.match?(/^\d{6}$/)
```

- **Avaliação por Igualdade Literal:** Marcaria como **False** (expressões diferentes)
- **Avaliação por MNLI:** Identifica como **entailment** — semanticamente equivalente e até mais restritiva

A predição do modelo é não apenas correta, mas **mais robusta** que o ground truth, verificando também o tipo e o comprimento da string antes de aplicar a regex.

Caso 2 — Validação de Endereço Blockchain

Ground Truth

```
from_address.match?(/^0x[a-fA-F0-9]{40}$/)
```

Valida apenas endereços Ethereum no formato hexadecimal com prefixo 0x.

Predição do LLM

```
from_address.match?(/^(0x)?[a-fA-F0-9]{40}$/) ||
from_address.match?(/^[13][a-km-zA-HJ-NP-Z1-9]{25,34}$/)
```

Aceita endereços Ethereum *ou* Bitcoin, tornando a validação mais permissiva.

Classificação MNLI: O LLM expandiu a validação para incluir Bitcoin addresses. Embora não seja exatamente igual ao ground truth, não é contraditório — é apenas mais permissivo. MNLI marca como "não contraditório / permissive".

Sem MNLI, este caso seria automaticamente considerado incorreto, mesmo sendo uma interpretação válida e potencialmente mais útil do requisito.

Exemplos de Sucessos Diretos

Caso 1 — Parâmetros Hash

Ground truth:

```
params.is_a?(Hash)
```

Predição: Idêntica

Avaliação correta em ambos os métodos (exact match e NLI). Validação simples e direta de tipo Hash.

Caso 2 — Lista de Transformações

Ground truth:

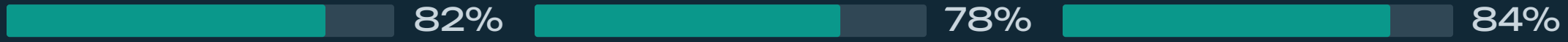
```
transformations.is_a?(Array) &&
transformations.length > 0
```

Predição: Idêntica

Sucesso imediato na validação de arrays não-vazios. Demonstra que o modelo captura bem condições compostas.

Estes casos demonstram que quando a documentação é clara e a validação é direta, o modelo LLM consegue gerar expressões booleanas perfeitamente equivalentes, sem necessidade de inferência complexa.

Resultados Quantitativos



PARAM

Acurácia na validação de parâmetros

RETURN

Acurácia na validação de retornos

RAISES

Acurácia na validação de exceções

Acurácia Geral

80%

Observações

Resultados inferiores ao artigo original (Java), devido a:

- **Sintaxe Ruby:** mais expressiva e livre
- **Dataset limitado:** apenas 286 métodos
- **Documentação inconsistente:** variação na qualidade dos comentários
- **Modelos não otimizados:** especificamente para Ruby

Apesar dos desafios, os resultados demonstram viabilidade da abordagem. A introdução do MNLI reduziu significativamente falsos negativos, tornando a avaliação mais precisa e reduzindo trabalho manual.