Relatório Técnico: Análise de Governança via Padrões de Commit

Responsável: Irandi

Objeto de Análise: Histórico de versionamento do `google/langextract`

1. Objetivo

O objetivo desta etapa foi mensurar o nível de disciplina e governança do projeto através da análise semântica e sintática das mensagens de commit. A premissa da Engenharia de Software é que projetos maduros utilizam padrões consistentes (como *Conventional Commits* ou *Imperative Mood*) para facilitar a rastreabilidade e automação.

Para garantir a robustez dos dados, a análise não se limitou a um único algoritmo, mas realizou uma **validação cruzada utilizando três modelos distintos de Inteligência Artificial**.

2. Metodologia Implementada

Desenvolvemos scripts em Python que combinam Processamento de Linguagem Natural (IA) com regras de negócio baseadas no estilo de código do Google. Abaixo, detalhamos as funções principais e a lógica aplicada.

2.1. Coleta de Dados (Amostragem Estatística)

Para garantir relevância estatística, não analisamos commits isolados. Implementamos uma função para extrair os últimos 100 commits via API do GitHub, garantindo uma visão histórica recente do projeto:

```
Python
def get_commits(owner, repo, limit=100):
    # Conecta na API do GitHub para baixar o histórico recente
    url = f"https://api.github.com/repos/{owner}/{repo}/commits?per_page={limit}"
    response = requests.get(url)

    # Processa e limpa as mensagens recebidas (List Comprehension)
    if response.status_code == 200:
        return [item['commit']['message'].split('\n')[0] for item in response.json()]
    return []
```

Explicação do Código: Utilizamos a biblioteca `requests` para bater na API pública do GitHub. O comando `.split('\n')[0]` é crucial: ele descarta o corpo longo da mensagem e pega apenas o "título" do commit, que é onde a padronização de governança deve obrigatoriamente aparecer.

2.2. Análise Semântica com IA (Multi-Modelos)

Diferente de uma abordagem simples, utilizamos **três modelos** do Hugging Face para validar a semântica. Isso converte textos em vetores matemáticos (*embeddings*), permitindo calcular a similaridade entre o que os desenvolvedores escreveram e o "Gabarito de Qualidade".

Modelos utilizados na validação:

1. **all-MiniLM-L6-v2:** (Referência/Equilibrado)
2. **all-mpnet-base-v2:** (Alta Precisão Semântica)
3. **paraphrase-MiniLM-L3-v2:** (Alta Velocidade)

```
Python
# Inicialização do modelo (Exemplo com o modelo de Alta Precisão)
model = SentenceTransformer("sentence-transformers/all-mpnet-base-v2")

# Criação dos vetores de referência (Gabarito de Qualidade)
padrao_bom = [
    "feat: add functionality", "fix: resolve bug", # Conventional Commits
    "Add new feature support", "Update dependency" # Imperative Mood (Google Style)
]
# A IA aprende o que é "Bom" transformando essas frases em números
embeddings_bom = model.encode(padrao_bom)
```

Explicação do Código: O método `model.encode` transforma as frases de referência em coordenadas numéricas. Assim, a IA não compara palavras soltas, ela compara o **significado/contexto** matemático do que é um "commit bom".

2.3. Algoritmo de Classificação Híbrida

O diferencial da nossa abordagem foi o refinamento lógico. Inicialmente focado apenas em *Conventional Commits*, ajustamos a lógica para validar também o *Imperative Mood* (verbos de ação), padrão comum em projetos Open Source de alta maturidade (como os do Google).

```
Python
# Lógica de Classificação aplicada a cada commit
for commit in commits_reais:
    # 1. Análise Vetorial (IA): Calcula similaridade (Cosseno) com boas práticas
    emb_commit = model.encode(commit)
```

```
sim_bom = util.cos_sim(emb_commit, embeddings_bom).max().item()

# 2. Análise Sintática: Verifica se começa com Verbos de Ação
primeira_palavra = commit.split(' ')[0].replace(':', '') # Limpa pontuação
verbos_fortes = ["Add", "Fix", "Update", "Remove", "Refactor", "feat", "fix", "chore"]

# Decisão Final (Lógica OR): É padronizado se usar verbo forte OU tiver alta similaridade
if primeira_palavra in verbos_fortes or sim_bom > 0.30:
    commits_padronizados += 1
else:
    commits_fora_padrao += 1
```

Explicação do Código: Esta é uma estrutura condicional híbrida.

- **Sintaxe:** Se a primeira palavra for um verbo forte (ex: "Fix"), o commit passa direto.
- **Semântica:** Se não tiver verbo forte, a IA verifica o contexto (`sim_bom > 0.30`). Isso evita falsos negativos, aprovando commits que são tecnicamente corretos mas escritos de forma ligeiramente diferente.

3. Resultados Obtidos (Validação Cruzada)

A execução dos algoritmos sobre a amostra (n=100) apresentou dados consistentes entre as três IAs testadas, comprovando a confiabilidade da análise:

Modelo de IA	Foco	Aderência (Governança)	Detalhe
all-MiniLM-L6	Equilíbrio	**86%**	86 commits estruturados.
all-mpnet-base	Alta Precisão	**89%**	Identificou nuances em refatorações complexas.
paraphrase-L3	Alta Velocidade	**91%**	Validou a clareza textual imediata.

Média Consolidada de Governança: ~88.6% **Commits Genéricos / Ad-Hoc:** Média de apenas 11%.

Exemplos típicos identificados como **BONS**:

- *`chore: Bump version to 1.1.1`* (Padrão Conventional)
- *`Fix: Pass project parameter correctly`* (Padrão Imperativo Google)

Exemplos identificados como **FORA DO PADRÃO**:

- `Internal: Strengthen CI/CD` (Uso de prefixo não padrão)
- `Import modules not classes` (Frase descritiva sem verbo de ação inicial)

4. Conclusão

Com índices de padronização variando entre **86% e 91%** nas três validações por IA, a análise comprova que a governança do projeto `LangExtract` é **ALTA E ESTRUTURADA**.

A presença dominante de mensagens iniciando com verbos imperativos (*Add, Update, Fix*) ou prefixos semânticos (*feat:, chore:*) demonstra que a equipe segue rigorosamente ritos de engenharia de software. O fato de três modelos diferentes de redes neurais chegarem à mesma conclusão reforça que a documentação do projeto é clara o suficiente para ser entendida tanto por humanos quanto por máquinas, viabilizando a automação total de *releases*.