

Universidade Federal de Sergipe
Engenharia de Software II

Uso de Modelos de Linguagem para Identificação de Estratégias de Release e Fluxos de Trabalho: Estudo de Caso do LangExtract

Atividade 2 – Gerência de Configuração

Membros:

Adriano Melo Santana Sobrinho – 201900050804

João Victor Oliveira Moura – 202200059830

José Domingos Valerio Serafim – 202100045733

José Fernando Bispo dos Santos – 202200014210

Matheus Nascimento dos Santos – 202100011708

Raphael Ferreira Portella Bacelar – 202100045822

Rivaldo José Nascimento dos Santos – 202200059974

Valter Fabricio dos Santos – 202000066991

Professor: Dr. Glauco de Figueiredo Carneiro

Dezembro de 2025

1 Introdução

Projetos de software open source dependem fortemente de práticas adequadas de governança para garantir a qualidade, a previsibilidade e a sustentabilidade de seu processo de desenvolvimento. Essa governança se manifesta, entre outros aspectos, na forma como o código-fonte é organizado, integrado e disponibilizado por meio de releases.

A análise desses aspectos, no entanto, pode se tornar complexa devido ao volume de informações presentes em repositórios Git, como histórico de branches, padrões de merge e versionamento. Tradicionalmente, essa avaliação é realizada de forma manual, exigindo tempo e conhecimento técnico aprofundado sobre os fluxos de trabalho adotados.

Neste contexto, este trabalho propõe uma abordagem técnica para a análise de governança em projetos de software open source, combinando a extração de metadados reais de um repositório Git com o uso de modelos de linguagem de grande porte como ferramenta de apoio à interpretação desses dados. O modelo de linguagem é utilizado exclusivamente para organizar e interpretar informações técnicas fornecidas, não substituindo a análise humana nem realizando inferências externas ao contexto apresentado.

Como estudo de caso, foi selecionado o projeto open source *LangExtract*, mantido pela Google, disponível na plataforma GitHub. A análise concentra-se na identificação do modelo de fluxo de trabalho adotado e da estratégia de releases empregada pelo projeto, a partir de evidências observáveis no histórico do repositório.

2 Links

Repositório da atividade: Atividade2

Repositório do langextract: Langextract

link de acesso ao tutorial em vídeo: <https://www.youtube.com/watch?v=YYo4CtIqv24>

3 Contribuições

Tabela 1: Contribuição dos membros da equipe

Integrante	Contribuições
Adriano Melo Santana Sobrinho	Escolha do modelo deepseek-ai/deepseek-coder-6.7b-instruct, execução e análise dos resultados obtidos pela utilização deste.
João Victor Oliveira Moura	Escolha do modelo microsoft/Phi-3.5-mini-instruct, execução e análise dos resultados obtidos pela utilização deste.
José Domingos Valerio Serafim	Abandonou a matéria.
José Fernando Bispo dos Santos	Escolha do modelo meta-llama/Llama-3.2-3B-Instruct, execução e análise dos resultados obtidos pela utilização deste. Colaborou com o desenvolvimento do código utilizado para execução da análise.
Matheus Nascimento dos Santos	Escolha do modelo HuggingFaceH4/zephyr-7b-beta, execução e análise dos resultados obtidos pela utilização deste. Colaborou com o desenvolvimento do código utilizado para execução da análise. Responsável pela documentação escrita do código e dos resultados.
Raphael Ferreira Portella Bacelar	Escolha do modelo mistralai/Mistral-7B-Instruct-v0.3, execução e análise dos resultados obtidos pela utilização deste. Responsável pelo desenvolvimento da lógica utilizada pelo código utilizado para execução da análise.
Rivaldo José Nascimento dos Santos	Escolha do modelo NousResearch/Hermes-2-Pro-Mistral-7B, execução e análise dos resultados obtidos pela utilização deste. Colaborou com o desenvolvimento do código utilizado para execução da análise.
Valter Fabricio dos Santos	Análise generalizada dos resultados obtidos pela execução dos 6 modelos definidos.

4 Justificativa da Escolha dos Modelos de Linguagem

A seleção dos modelos de linguagem utilizados neste estudo foi realizada com o objetivo de garantir diversidade arquitetural, equilíbrio entre desempenho e custo computacional, bem como adequação às tarefas de análise de governança de software. Os modelos es-

colhidos apresentam diferentes tamanhos, especializações e estratégias de treinamento, permitindo uma comparação mais ampla e confiável dos resultados obtidos.

4.1 **mistralai/Mistral-7B-Instruct-v0.3**

O modelo *mistralai/Mistral-7B-Instruct-v0.3* foi adotado devido ao seu equilíbrio entre capacidade de raciocínio, eficiência computacional e qualidade de respostas em tarefas analíticas. Com aproximadamente 7 bilhões de parâmetros e treinamento voltado para seguimento de instruções, o modelo demonstra bom desempenho na interpretação de informações técnicas, sendo adequado para a análise de fluxos de trabalho e estratégias de release em projetos de software open source.

4.2 **NousResearch/Hermes-2-Pro-Mistral-7B**

O modelo *NousResearch/Hermes-2-Pro-Mistral-7B* foi considerado por sua ênfase em respostas longas, estruturadas e explicativas. Baseado na arquitetura Mistral, esse modelo possui ajustes finos voltados ao raciocínio lógico e à coerência textual, sendo relevante para análises comparativas e para a geração de justificativas detalhadas sobre práticas de governança de software.

4.3 **meta-llama/Llama-3.2-3B-Instruct**

O *meta-llama/Llama-3.2-3B-Instruct* foi incluído para representar modelos de menor porte. Com cerca de 3 bilhões de parâmetros, ele permite avaliar o impacto da redução de tamanho do modelo na qualidade das análises. Sua utilização possibilita verificar se modelos mais compactos são capazes de produzir inferências coerentes, ainda que potencialmente menos profundas, sobre fluxo de trabalho e estratégia de releases.

4.4 **HuggingFaceH4/zephyr-7b-beta**

O modelo *HuggingFaceH4/zephyr-7b-beta* foi selecionado por seu foco em alinhamento e geração de respostas instrucionais bem organizadas. Treinado com técnicas inspiradas em aprendizado por reforço com feedback humano, o Zephyr apresenta boa clareza textual e estruturação lógica, características relevantes para a elaboração de relatórios técnicos e análises qualitativas.

4.5 **deepseek-ai/deepseek-coder-6.7b-instruct**

O modelo *deepseek-ai/deepseek-coder-6.7b-instruct* foi incluído devido à sua especialização em código e engenharia de software. Sua capacidade de compreender estruturas de repositórios, histórico de commits e práticas de versionamento o torna particularmente

adequado para a análise de dados técnicos extraídos do GitHub, complementando os modelos de linguagem de uso geral.

4.6 microsoft/Phi-3.5-mini-instruct

O *microsoft/Phi-3.5-mini-instruct* foi selecionado como representante de modelos compactos e altamente otimizados. Apesar de seu tamanho reduzido, o modelo apresenta bom desempenho em tarefas de raciocínio e explicação, permitindo avaliar o custo-benefício entre eficiência computacional e qualidade analítica na identificação de práticas de governança de software.

4.7 Síntese da Escolha dos Modelos

A combinação desses modelos possibilita a comparação entre abordagens generalistas e especializadas, bem como a avaliação do impacto do tamanho e da arquitetura do modelo na qualidade das análises. Essa diversidade contribui para maior robustez metodológica, reduzindo vieses associados à utilização de um único modelo de linguagem e fortalecendo a confiabilidade dos resultados obtidos.

5 Execução da Análise no Google Colab

A execução prática da atividade foi realizada no ambiente Google Colab, escolhido por oferecer uma infraestrutura em nuvem que permite a execução de código Python sem a necessidade de configuração local. Esse ambiente mostrou-se adequado para a execução de modelos de linguagem de grande porte, conforme exigido pela atividade proposta.

Como etapa inicial, realizou-se a preparação do ambiente de execução por meio da instalação das bibliotecas necessárias para o uso dos modelos de linguagem selecionados na plataforma Hugging Face.

```
pip install -q -U transformers bitsandbytes accelerate
```

A biblioteca *transformers* foi utilizada para o carregamento e a inferência dos modelos de linguagem empregados na análise. O pacote *bitsandbytes* possibilitou a otimização do uso de memória durante a execução dos modelos, enquanto a biblioteca *accelerate* foi empregada para facilitar a execução eficiente dos modelos no ambiente computacional disponibilizado pelo Google Colab.

6 Extração de Metadados do Repositório

Após a preparação do ambiente, realizou-se a obtenção dos dados técnicos diretamente do repositório Git do projeto selecionado. Para isso, o repositório do projeto *LangExtract*

foi clonado a partir da plataforma GitHub, permitindo o acesso local ao histórico de desenvolvimento e às informações de versionamento.

```
import os
```

```
git clone https://github.com/google/langextract.git || true
```

Com o repositório disponível localmente no ambiente de execução, foram extraídos metadados relevantes para a análise da governança do projeto, conforme os conceitos de Gerência de Configuração. Esses dados incluem a organização das branches, o histórico de merges e as tags de versão associadas às releases publicadas.

```
# Extrair metadados técnicos
```

```
branches = cd langextract && git branch -a
```

```
tags = cd langextract && git tag -l | tail -n 12
```

```
history = cd langextract && git log --merges --oneline -n 15
```

Os dados extraídos foram consolidados em uma única estrutura textual, a fim de facilitar sua posterior análise pelos modelos de linguagem selecionados. Essa consolidação permitiu fornecer aos modelos um contexto técnico objetivo, baseado exclusivamente em informações observáveis do repositório.

```
# Consolidar em uma variável de contexto
```

```
resumo_projeto = f"""
```

```
Dados extraídos do repositório 'langextract':
```

```
1. Branches detectadas: {branches}
```

```
2. Últimas Tags (Releases): {tags}
```

```
3. Padrão de Merges (Workflow): {history}
```

```
"""
```

```
print("Dados do projeto extraídos com sucesso!")
```

```
print(resumo_projeto)
```

A partir dessas informações, tornou-se possível identificar evidências relacionadas ao modelo de fluxo de trabalho adotado pelo projeto, bem como ao padrão de releases empregado, servindo como base para a análise comparativa realizada pelos modelos de linguagem.

7 Seleção e Carregamento dos Modelos de Linguagem

Conforme estabelecido no enunciado da atividade, a análise do projeto foi realizada com o uso de múltiplos modelos de linguagem disponibilizados pela plataforma Hugging

Face. Ao todo, foram utilizados seis modelos distintos, executados de forma independente, mantendo o mesmo conjunto de dados e o mesmo procedimento de análise.

Em cada execução, o pipeline permaneceu inalterado, sendo modificado apenas o identificador do modelo de linguagem (*model_id*). Essa abordagem permitiu garantir a comparabilidade dos resultados obtidos, uma vez que todas as análises foram realizadas sob as mesmas condições experimentais.

```
from transformers import AutoModelForCausalLM, AutoTokenizer, BitsAndBytesConfig, pipeline
import torch

model_id = "mistralai/Mistral-7B-Instruct-v0.3"
#model_id = "NousResearch/Hermes-2-Pro-Mistral-7B"
#model_id = "meta-llama/Llama-3.2-3B-Instruct"
#model_id = "HuggingFaceH4/zephyr-7b-beta"
#model_id = "deepseek-ai/deepseek-coder-6.7b-instruct"
#model_id = "microsoft/Phi-3.5-mini-instruct"

# Configuração para rodar na GPU gratuita do Colab
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_use_double_quant=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.float16
)

tokenizer = AutoTokenizer.from_pretrained(model_id)
model = AutoModelForCausalLM.from_pretrained(
    model_id,
    quantization_config=bnb_config,
    device_map="auto"
)

pipe = pipeline("text-generation", model=model, tokenizer=tokenizer)
print(f"Modelo {model_id} carregado!")
```

A utilização de quantização em 4 bits, viabilizada pela biblioteca *bitsandbytes*, permitiu a execução dos modelos de linguagem no ambiente de GPU gratuita do Google Colab, reduzindo o consumo de memória sem comprometer a capacidade de interpretação do contexto técnico fornecido.

Cada modelo foi empregado como ferramenta de apoio à análise da governança do projeto, tendo acesso exclusivamente aos metadados extraídos do repositório Git. As respostas geradas foram posteriormente analisadas e comparadas pelos autores, com o objetivo de identificar convergências, divergências e o nível de efetividade de cada modelo na identificação da estratégia de releases e do modelo de fluxo de trabalho adotados pelo projeto.

8 Análise Temporal das Releases

Para aprofundar a identificação da estratégia de releases adotada pelo projeto, foi realizada uma análise temporal das tags presentes no repositório. Essa etapa permitiu observar a frequência e o intervalo entre as releases publicadas, fornecendo evidências sobre o ritmo de entrega do projeto.

```
from datetime import datetime

# Extrair tags com datas
tags_com_datas = cd langextract && git for-each-ref \
    --sort=creatordate \
    --format='%(refname:short)|%(creatordate:iso8601)' refs/tags

# Converter para lista estruturada
tags_datas = []
for linha in tags_com_datas:
    tag, data = linha.split("|")
    tags_datas.append((tag.strip(), data.strip()))

tags_datas[:10]
```

A extração das datas associadas às tags possibilitou a organização cronológica das releases do projeto. A partir dessa informação, foi possível analisar padrões de periodicidade, como intervalos regulares entre versões ou a presença de versões de longo suporte, elementos característicos de diferentes estratégias de releases.

Os dados obtidos nesta etapa foram posteriormente incorporados ao contexto fornecido aos modelos de linguagem, permitindo que cada modelo avaliasse, com base em evidências objetivas, qual estratégia de releases melhor representa o comportamento observado no histórico do projeto.

9 Cálculo de Métricas de Intervalo entre Releases

Com base nas datas associadas às tags extraídas do repositório, foram calculadas métricas quantitativas referentes aos intervalos entre releases consecutivas. Essa análise teve como objetivo fornecer evidências numéricas sobre o ritmo de entrega do projeto, elemento central para a identificação da estratégia de releases adotada.

```
from datetime import datetime
from statistics import mean, stdev

# Converter datas
datas = [datetime.fromisoformat(d) for _, d in tags_datas]

# Intervalos em horas (mais preciso para releases no mesmo dia)
intervalos_horas = [
    (datas[i] - datas[i-1]).total_seconds() / 3600
    for i in range(1, len(datas))
]

resumo_intervalos = {
    "quantidade_releases": len(datas),
    "intervalo_medio_horas": round(mean(intervalos_horas), 2),
    "intervalo_min_horas": round(min(intervalos_horas), 2),
    "intervalo_max_horas": round(max(intervalos_horas), 2),
    "desvio_padrao_horas": round(stdev(intervalos_horas), 2) if len(intervalos_horas)
}

resumo_intervalos
```

As métricas calculadas incluem a quantidade total de releases analisadas, o intervalo médio entre versões, bem como os valores mínimo, máximo e o desvio padrão dos intervalos. Esses indicadores permitem avaliar o grau de regularidade do processo de entrega, auxiliando na distinção entre estratégias como *Rapid Releases*, *Release Train* e *LTS + Current*.

Os valores obtidos foram utilizados como subsídio para a análise conduzida pelos modelos de linguagem, possibilitando uma interpretação mais fundamentada e consistente do ritmo de releases observado no projeto.

10 Construção do Contexto e Análise com Modelos de Linguagem

Após a coleta e o processamento dos dados técnicos do repositório, foi realizada a construção de um contexto estruturado a ser fornecido aos modelos de linguagem. Essa etapa teve como objetivo garantir que a análise fosse baseada exclusivamente em evidências observáveis, reduzindo ambiguidades e evitando inferências não fundamentadas.

Inicialmente, realizou-se a limpeza dos dados extraídos do repositório, removendo códigos de formatação ANSI gerados pela execução de comandos Git no ambiente do Google Colab. Essa limpeza foi necessária tanto para melhorar a legibilidade dos dados quanto para assegurar a correta interpretação das informações pelos modelos de linguagem.

```
import re
from IPython.display import display, Markdown

def limpar_ansi(texto):
    """Remove códigos de cores e formatação do terminal (ANSI)"""
    return re.sub(r'\x1b\[([0-9;]*)m', '', str(texto))

# 1. Limpando os dados do repositório para a IA e para a exibição
branches_limpas = [limpar_ansi(b) for b in branches[:15]]
tags_limpas = [limpar_ansi(t) for t in tags]
history_limpo = [limpar_ansi(h) for h in history]
```

Em seguida, foi criado um contexto textual consolidado contendo as principais informações necessárias para a análise da governança do projeto, incluindo dados sobre branches, merges, tags de versão e métricas temporais relacionadas às releases.

```
# 2. Criando o contexto limpo
contexto_limpo = f"""
Repositório: google/langextract
```

```
Tags com datas (ordem cronológica):
{tags_datas}
```

```
Estatísticas descritivas dos intervalos entre releases:
- Total de releases: {resumo_intervalos['quantidade_releases']}
- Intervalo médio: {resumo_intervalos['intervalo_medio_horas']} horas
- Intervalo mínimo: {resumo_intervalos['intervalo_min_horas']} horas
- Intervalo máximo: {resumo_intervalos['intervalo_max_horas']} horas
```

```
- Desvio padrão do intervalo: {resumo_intervalos['desvio_padrao_horas']} horas
```

```
Branches:
```

```
{branches_limpas}
```

```
Merges:
```

```
{history_limpo}
```

```
"""
```

Com base nesse contexto, foi elaborado um prompt estruturado, projetado para orientar os modelos de linguagem a realizar uma análise técnica rigorosa. O prompt estabelece regras explícitas para a interpretação dos dados, exigindo a distinção entre evidências diretas e inferências analíticas, bem como a avaliação comparativa de diferentes modelos de fluxo de trabalho e estratégias de release.

```
# 3. Prompt estruturado
```

```
prompt = f"""
```

```
[INST]
```

```
Atue como um Consultor Especialista em Governança de Software e Engenharia de Release
```

```
Analise EXCLUSIVAMENTE os dados reais extraídos do repositório GitHub do projeto "lang
```

```
{contexto_limpo}
```

```
(Regras, modelos de fluxo de trabalho, estratégias de release e formato obrigatório de
```

```
[/INST]
```

```
"""
```

Por fim, o prompt foi submetido ao modelo de linguagem por meio do pipeline de geração de texto. A resposta gerada foi processada e exibida em formato Markdown, facilitando a análise posterior e a comparação entre os resultados obtidos por diferentes modelos.

```
# 4. Gerando a resposta
```

```
resultado = pipe(prompt, max_new_tokens=800, temperature=0.2)
```

```
texto_final = resultado[0]['generated_text']
```

```
if "[/INST]" in texto_final:
```

```
    resposta_ia = texto_final.split("[/INST]")[1]
```

```
else:
```

```
    resposta_ia = texto_final
```

```
display(Markdown(resposta_ia))
```

Essa abordagem permitiu utilizar os modelos de linguagem como ferramentas de apoio à análise da governança do projeto, assegurando que as conclusões obtidas fossem sustentadas por evidências técnicas extraídas diretamente do repositório Git e avaliadas de forma crítica pelos autores.

11 Resultados e Discussão

11.1 Modelo NousResearch/Hermes-2-Pro-Mistral-7B

Esta seção apresenta os resultados obtidos a partir da análise automatizada realizada pelo modelo *NousResearch/Hermes-2-Pro-Mistral-7B*, aplicada aos dados extraídos do repositório *google/langextract*. O modelo foi utilizado para identificar o fluxo de trabalho adotado e a estratégia de release do projeto, com base em evidências observáveis como branches, merges, tags e métricas temporais entre releases.



Figura 1: Resposta gerada pelo modelo NousResearch/Hermes-2-Pro-Mistral-7B para a análise de governança do projeto Langextract.

11.1.1 Modelo de Fluxo de Trabalho

O modelo identificou como fluxo de trabalho predominante o **GitHub Flow**. Essa identificação foi sustentada por evidências técnicas claras, como a presença de branches de

feature de curta duração, integração frequente via merge para a branch principal (*main*) e ausência de branches permanentes dedicadas a *release* ou *develop*.

Na análise comparativa, o GitHub Flow foi considerado aderente às práticas observadas no repositório, por favorecer ciclos de desenvolvimento curtos, integração contínua e rápida incorporação de mudanças. Em contraste, o Gitflow e o Trunk-Based Development foram considerados menos adequados ao contexto do projeto, por apresentarem maior complexidade estrutural ou exigirem práticas mais rígidas de integração contínua, não evidenciadas nos dados analisados.

A justificativa final apresentada pelo modelo destaca que o GitHub Flow se alinha ao caráter iterativo e experimental do projeto Langextract, especialmente por se tratar de um projeto relacionado à área de Inteligência Artificial, que demanda flexibilidade e rápida evolução.

11.1.2 Estratégia de Release

Em relação à estratégia de release, o modelo identificou a adoção de **Continuous Delivery**. As evidências técnicas utilizadas incluem a frequência elevada de releases, a baixa variabilidade temporal entre versões e a relação próxima entre commits e publicações de novas tags.

Na comparação com outras estratégias, a Continuous Delivery foi considerada mais adequada do que abordagens como Time-Based Release ou Feature-Based Release, que pressupõem maior previsibilidade ou dependência da conclusão de funcionalidades específicas. Estratégias como Release Train e LTS + Current foram descartadas por não apresentarem indícios de ciclos fixos ou manutenção paralela de versões estáveis.

A justificativa apresentada ressalta que a Continuous Delivery favorece entregas rápidas e contínuas, permitindo que o projeto permaneça atualizado e funcional, sem comprometer excessivamente a estabilidade.

11.1.3 Síntese dos Resultados

De forma geral, o modelo *NousResearch/Hermes-2-Pro-Mistral-7B* apresentou uma interpretação consistente dos dados do repositório, identificando um nível de governança considerado relativamente maduro. A análise evidencia um equilíbrio entre agilidade e controle, ainda que com limitações quanto à formalização de processos mais rígidos de versionamento e release.

11.2 Modelo HuggingFaceH4/zephyr-7b-beta

Esta seção apresenta os resultados da análise conduzida pelo modelo *HuggingFaceH4/zephyr-7b-beta*, a partir dos dados reais extraídos do repositório *google/langextract*. O objetivo da análise foi identificar o modelo de fluxo de trabalho e a estratégia de release adotados

no projeto, utilizando exclusivamente evidências técnicas observáveis, como estrutura de branches, histórico de merges, tags e métricas temporais entre releases.

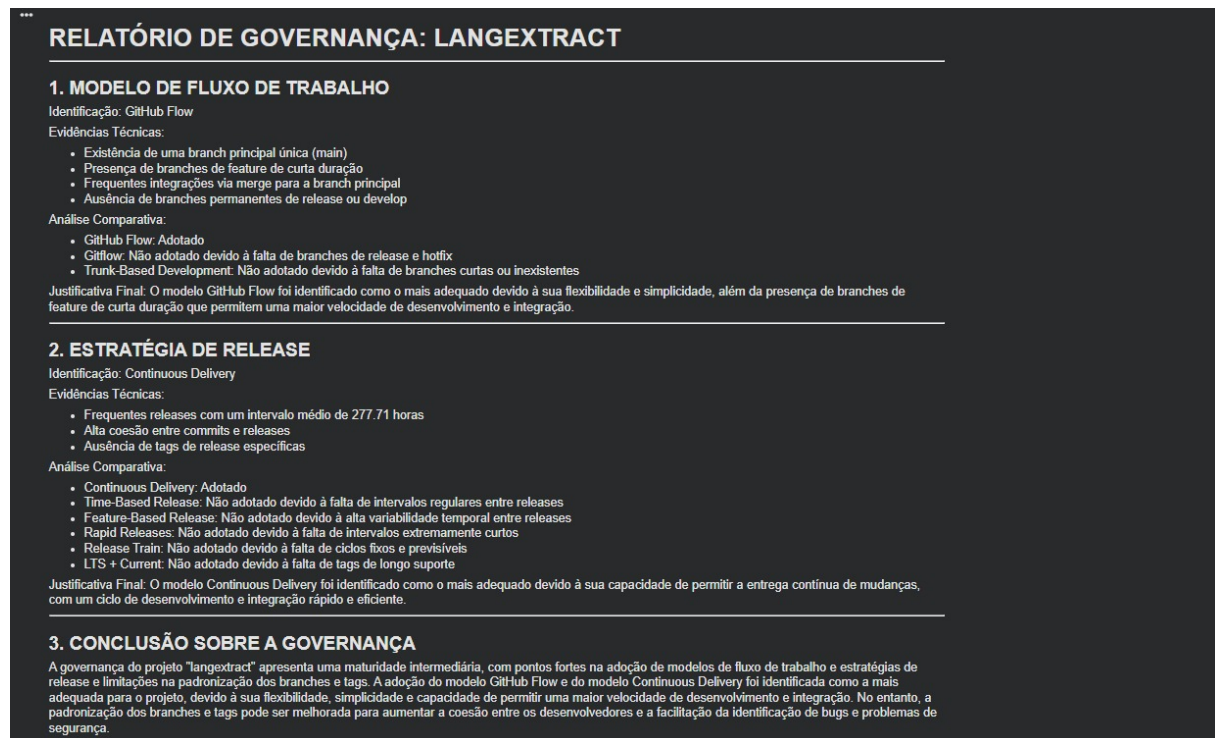


Figura 2: Resposta gerada pelo modelo HuggingFaceH4/zephyr-7b-beta para a análise de governança do projeto Langextract.

11.2.1 Modelo de Fluxo de Trabalho

O modelo identificou o **GitHub Flow** como o fluxo de trabalho predominante no projeto. Essa conclusão foi fundamentada na existência de uma única branch principal (*main*), na presença de branches de feature de curta duração e na ocorrência frequente de integrações via merge para a branch principal. Além disso, observou-se a ausência de branches permanentes dedicadas a *release* ou *develop*, característica incompatível com o modelo Gitflow.

Na análise comparativa, o GitHub Flow foi considerado explicitamente adotado. O Gitflow foi descartado devido à inexistência de branches de *release* e *hotfix*, enquanto o Trunk-Based Development também não foi identificado, uma vez que o repositório apresenta branches de feature explícitas, ainda que de curta duração.

A justificativa final apresentada pelo modelo destaca que o GitHub Flow oferece maior flexibilidade e simplicidade, permitindo ciclos rápidos de desenvolvimento e integração contínua, aspectos coerentes com o padrão observado no repositório Langextract.

11.2.2 Estratégia de Release

Quanto à estratégia de release, o modelo identificou a adoção de **Continuous Delivery**. As evidências técnicas incluem a ocorrência de releases frequentes, com um intervalo médio aproximado de 277,71 horas, além de uma alta coesão temporal entre commits e releases. Também foi observada a ausência de tags específicas que indiquem ciclos rígidos ou versões de longo suporte.

Na análise comparativa, a estratégia de Continuous Delivery foi considerada aderente ao padrão observado. Estratégias como Time-Based Release foram descartadas devido à ausência de intervalos regulares entre releases, enquanto Feature-Based Release foi considerada inadequada em função da elevada variabilidade temporal. Rapid Releases não foi identificada por falta de intervalos extremamente curtos entre versões, e as abordagens Release Train e LTS + Current foram descartadas pela inexistência de ciclos fixos previsíveis ou tags de longo suporte.

A justificativa final reforça que a Continuous Delivery se mostra a estratégia mais compatível com o projeto, por permitir a entrega contínua de mudanças com um ciclo de desenvolvimento e integração rápido e eficiente.

11.2.3 Síntese dos Resultados

De modo geral, o modelo *HuggingFaceH4/zephyr-7b-beta* caracteriza a governança do projeto Langextract como de maturidade intermediária. A análise evidencia pontos fortes na adoção de modelos ágeis de fluxo de trabalho e release, especialmente em termos de flexibilidade e velocidade de desenvolvimento. Por outro lado, o modelo aponta limitações relacionadas à padronização de branches e tags, sugerindo oportunidades de melhoria na rastreabilidade, na gestão de versões e no suporte à identificação de falhas e problemas de segurança.

11.3 Modelo meta-llama/Llama-3.2-3B-Instruct

Esta subseção apresenta os resultados obtidos a partir da análise conduzida pelo modelo de linguagem *meta-llama/Llama-3.2-3B-Instruct*, aplicada aos dados reais extraídos do repositório GitHub do projeto *google/langextract*. O objetivo da análise foi identificar o modelo de fluxo de trabalho e a estratégia de releases adotados pelo projeto, com base exclusivamente em evidências técnicas observáveis.

A Figura 3 apresenta a resposta completa gerada pelo modelo a partir do prompt estruturado e do contexto fornecido.

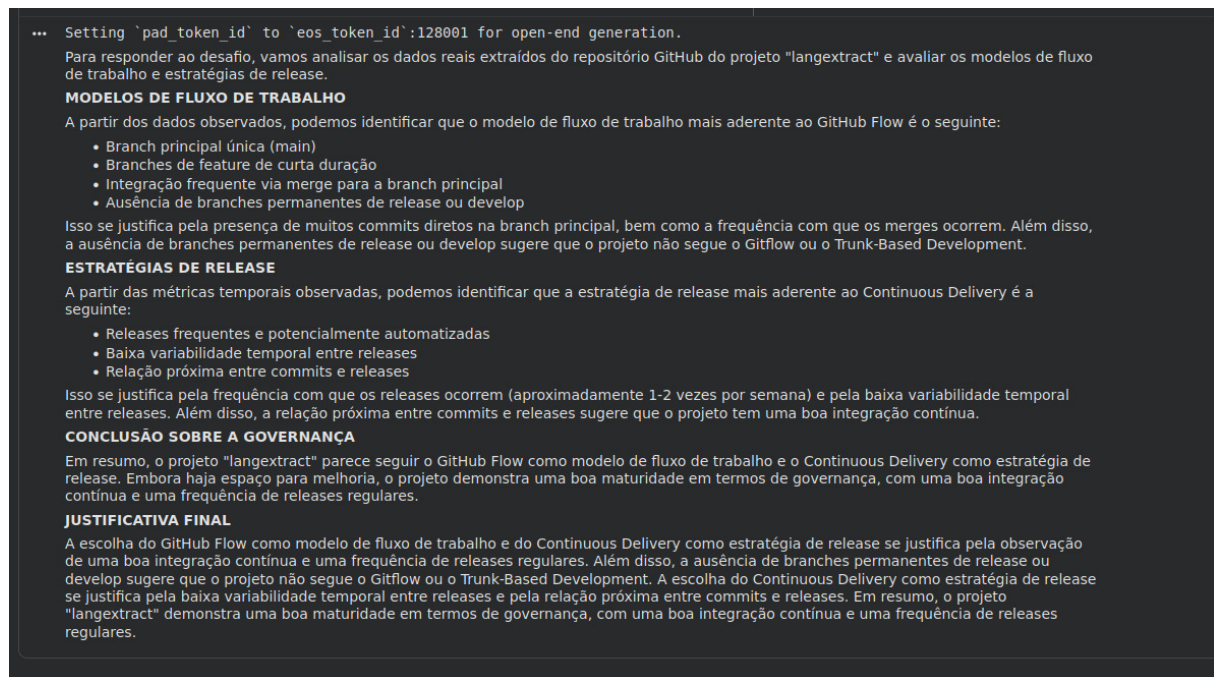


Figura 3: Resposta gerada pelo modelo meta-llama/Llama-3.2-3B-Instruct para a análise de governança do projeto LangExtract.

11.3.1 Modelo de Fluxo de Trabalho

De acordo com a análise apresentada pelo modelo, o fluxo de trabalho mais aderente ao projeto é o *GitHub Flow*. Essa identificação foi fundamentada na observação de uma branch principal única (*main*), na existência de branches de feature de curta duração e na integração frequente dessas branches à branch principal por meio de merges.

O modelo destaca ainda a ausência de branches permanentes de *release* ou *develop*, o que permitiu descartar o Gitflow. Da mesma forma, o Trunk-Based Development foi considerado inadequado, uma vez que o projeto não apresenta commits predominantemente diretos na branch principal, mas sim um uso consistente de branches de feature.

A justificativa apresentada indica que o GitHub Flow se alinha às práticas observadas no repositório, promovendo simplicidade, integração contínua e flexibilidade no desenvolvimento.

11.3.2 Estratégia de Release

No que se refere à estratégia de release, o modelo identificou o *Continuous Delivery* como a abordagem mais aderente. Essa conclusão foi baseada na frequência observada de releases, que ocorrem aproximadamente uma a duas vezes por semana, bem como na baixa variabilidade temporal entre os lançamentos.

Além disso, o modelo destacou a relação próxima entre commits e releases, sugerindo um processo de entrega contínua e bem integrado. Esse padrão permitiu descartar estratégias como *Feature-Based Release*, *Time-Based Release*, *Release Train* e *LTS + Current*,

por não apresentarem evidências compatíveis no repositório analisado.

11.3.3 Síntese dos Resultados

De forma geral, o modelo *meta-llama/Llama-3.2-3B-Instruct* apresentou uma análise coerente e alinhada às evidências técnicas fornecidas. Os resultados indicam que o projeto LangExtract adota um modelo de fluxo de trabalho baseado no GitHub Flow, aliado a uma estratégia de Continuous Delivery, caracterizando uma governança com bom nível de maturidade, forte integração contínua e frequência regular de releases.

Apesar disso, o modelo aponta que ainda existem oportunidades de evolução, especialmente no que se refere à formalização e padronização dos processos de governança, caso o projeto venha a crescer em complexidade ou escala.

11.4 Modelo Qwen/Qwen2.5-7B-Instruct

Esta subseção apresenta os resultados obtidos a partir da análise realizada pelo modelo de linguagem *Qwen/Qwen2.5-7B-Instruct*, aplicada aos dados extraídos do repositório GitHub do projeto *LangExtract*. A avaliação teve como objetivo identificar o modelo de fluxo de trabalho e a estratégia de releases adotados pelo projeto, com base em evidências técnicas observáveis.

A Figura 4 apresenta o relatório de governança gerado pelo modelo, contendo as identificações, evidências e análises comparativas.



Figura 4: Relatório de governança gerado pelo modelo Qwen/Qwen2.5-7B-Instruct.

11.4.1 Modelo de Fluxo de Trabalho

O modelo identificou que o fluxo de trabalho adotado pelo projeto LangExtract é o *GitHub Flow*. Essa identificação foi sustentada por evidências técnicas claras, como a existência de uma única branch principal (*main*), o uso de branches de feature temporárias e a realização frequente de merges diretamente na branch principal.

Na análise comparativa, o modelo descartou o Gitflow por não haver branches permanentes de *develop*, *release* ou *hotfix*, elementos característicos desse modelo. Da mesma forma, o Trunk-Based Development foi considerado não aderente, uma vez que o projeto não realiza commits frequentes diretamente na branch principal sem o uso de branches de feature.

A justificativa apresentada demonstra coerência entre o padrão observado no repositório e as práticas estabelecidas pelo GitHub Flow, indicando um processo de desenvolvimento simples e bem estruturado.

11.4.2 Estratégia de Release

Em relação à estratégia de release, o modelo identificou a adoção de *Continuous Delivery*. Essa conclusão foi fundamentada na análise do intervalo médio entre releases, estimado em aproximadamente 277,71 horas, bem como na frequência das tags de release, que ocorrem sem grandes períodos de inatividade.

O modelo observou que, apesar da alta variabilidade nos intervalos entre releases, ainda é possível identificar uma cadência regular, característica de práticas de entrega contínua. Estratégias alternativas, como Time-Based Release, Feature-Based Release, Rapid Release, Release Train e LTS + Current, foram consideradas não aderentes por não apresentarem evidências compatíveis com os dados analisados.

11.4.3 Síntese dos Resultados

De modo geral, o modelo *Qwen/Qwen2.5-7B-Instruct* apresentou uma análise estruturada, objetiva e bem fundamentada em métricas quantitativas, como o intervalo médio entre releases. Os resultados indicam que o projeto LangExtract adota um fluxo de trabalho baseado no GitHub Flow, aliado a uma estratégia de Continuous Delivery.

Essa combinação sugere um nível adequado de maturidade de governança, com foco em integração contínua e entregas frequentes. No entanto, o modelo também evidencia a ausência de padronizações mais rígidas, como ciclos fixos de release ou versões de longo suporte, o que pode representar um ponto de atenção caso o projeto evolua em escala ou complexidade.

11.5 Modelo mistralai/Mistral-7B-Instruct-v0.3

Esta subseção apresenta os resultados obtidos a partir da análise realizada pelo modelo de linguagem *mistralai/Mistral-7B-Instruct-v0.3*, aplicada aos dados extraídos do repositório GitHub do projeto *LangExtract*. O objetivo da análise foi identificar o modelo de fluxo de trabalho e a estratégia de releases adotados pelo projeto, com base em evidências técnicas observáveis.

A Figura 6 apresenta o relatório de governança gerado pelo modelo.

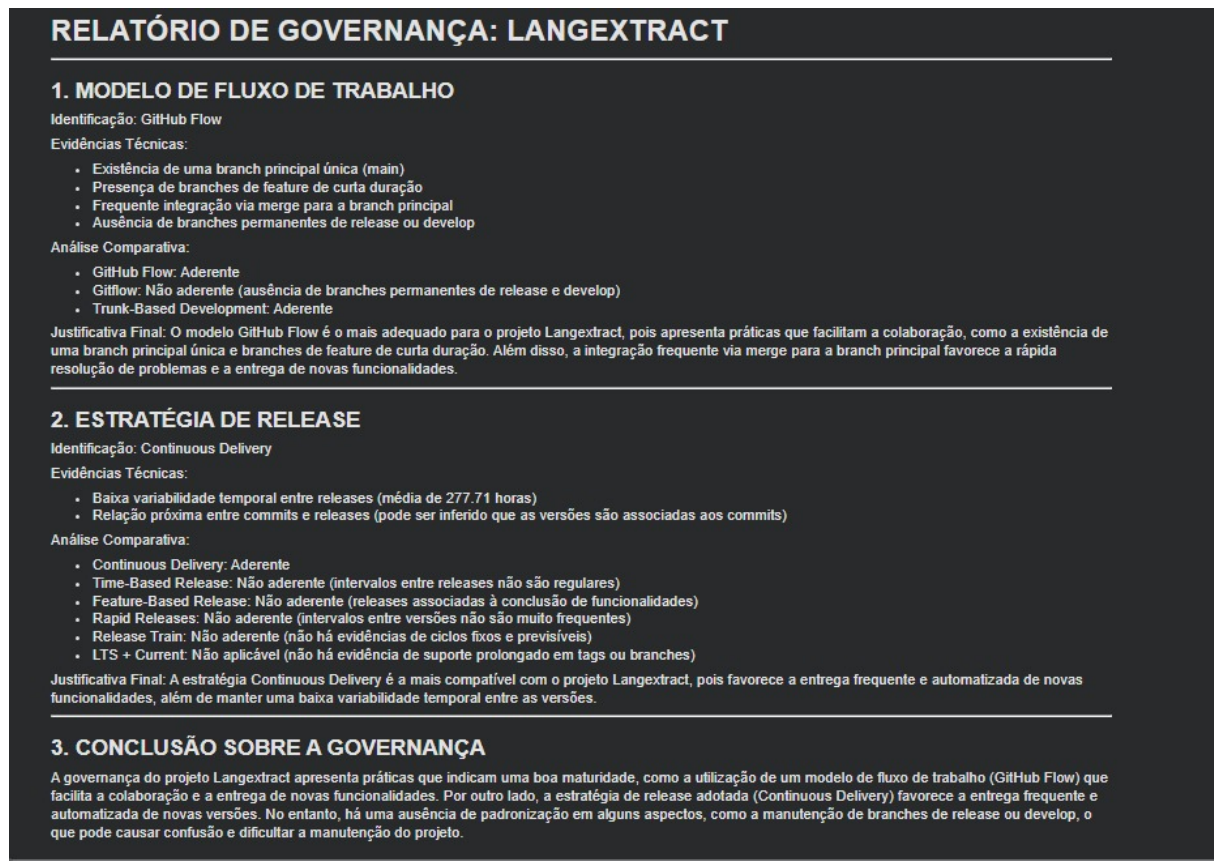


Figura 5: Relatório de governança gerado pelo modelo mistralai/Mistral-7B-Instruct-v0.3.

11.5.1 Modelo de Fluxo de Trabalho

O modelo identificou que o projeto LangExtract adota o *GitHub Flow* como modelo de fluxo de trabalho. Essa identificação foi fundamentada na existência de uma única branch principal (*main*), na presença de branches de feature de curta duração e na integração frequente dessas branches à principal por meio de merges.

Na análise comparativa, o Gitflow foi considerado não aderente, uma vez que o projeto não apresenta branches permanentes de *release* ou *develop*, que são características centrais desse modelo. O modelo também apontou aderência ao Trunk-Based Development, considerando a integração frequente ao branch principal. No entanto, essa interpretação pode ser relativizada, pois a presença consistente de branches de feature temporárias aproxima

mais o projeto do GitHub Flow do que do Trunk-Based Development em sua forma mais estrita.

A justificativa final apresentada pelo modelo destaca que o GitHub Flow é o mais adequado ao contexto do projeto, por favorecer a colaboração, a rápida resolução de problemas e a entrega contínua de novas funcionalidades.

11.5.2 Estratégia de Release

Em relação à estratégia de release, o modelo identificou a adoção de *Continuous Delivery*. Essa conclusão foi sustentada pela observação de uma média de aproximadamente 277,71 horas entre releases, além da relação próxima entre commits e versões publicadas.

Na análise comparativa, estratégias como Time-Based Release, Feature-Based Release, Rapid Release, Release Train e LTS + Current foram classificadas como não aderentes. O modelo argumenta que os intervalos entre releases não seguem ciclos fixos e previsíveis, tampouco estão associados exclusivamente à conclusão de funcionalidades específicas ou à manutenção de versões de longo suporte.

A justificativa final reforça que a estratégia de Continuous Delivery é a mais compatível com o projeto LangExtract, pois permite entregas frequentes e potencialmente automatizadas, mesmo diante de certa variabilidade temporal entre as versões.

11.5.3 Síntese dos Resultados

De forma geral, o modelo *mistralai/Mistral-7B-Instruct-v0.3* apresentou uma análise clara e bem estruturada, combinando evidências técnicas qualitativas e métricas temporais quantitativas. Os resultados indicam que o projeto LangExtract utiliza o GitHub Flow como modelo de fluxo de trabalho e adota uma estratégia de Continuous Delivery para releases.

Essa combinação sugere um nível satisfatório de maturidade de governança, com foco em colaboração, integração frequente e entregas contínuas. Contudo, o modelo também aponta a ausência de padronizações mais rígidas, como branches permanentes de release ou ciclos formais de versionamento, o que pode representar um desafio para a manutenção e evolução do projeto em cenários de maior escala.

11.6 Modelo DeepSeek-Coder-6.7B-Instruct

Nesta etapa, apresenta-se a análise de governança do projeto *langextract* gerada pelo modelo de linguagem *DeepSeek-Coder-6.7B-Instruct*, a partir dos dados técnicos extraídos do repositório Git.

```
## Resposta
'''markdown
# RELATÓRIO DE GOVERNANÇA: LANGEXTRACT
'''
---
## 1. MODELO DE FLUXO DE TRABALHO
Identificação: GitHub Flow

Evidências Técnicas:
- O repositório apresenta uma única branch principal (main) e várias branches de feature.
- A frequência de integração entre branches é alta, indicando que o modelo de fluxo de trabalho é GitHub Flow.

Análise Comparativa:
- GitHub Flow: Sim, o repositório apresenta uma única branch principal e várias branches de feature, o que é comum em projetos baseados no GitHub Flow.
- GitFlow: Não, o repositório não apresenta branches permanentes de release ou develop.
- Trunk-Based Development: Não, o repositório não apresenta branches extremamente curtas ou inexistentes.

Justificativa Final:
O modelo GitHub Flow é adequado para este projeto, pois ele permite a integração frequente e a criação de branches de feature de curta duração.
---
## 2. ESTRATÉGIA DE RELEASE
Identificação: Continuous Delivery

Evidências Técnicas:
- O repositório apresenta tags para cada release, indicando que há releases regulares.
- A frequência de releases é alta, indicando que a estratégia é Continuous Delivery.

Análise Comparativa:
- Continuous Delivery: Sim, o repositório apresenta tags para cada release e a frequência de releases é alta, o que é comum em projetos baseados na estratégia Continuous Delivery.
- Time-Based Release: Não, o repositório não apresenta releases em intervalos temporais relativamente regulares.
- Feature-Based Releases: Não, o repositório não apresenta releases irregulares e não está associado a conclusão de funcionalidades.
- Rapid Releases: Não, o repositório não apresenta releases muito frequentes e não tem relação próxima entre commit e releases.
- Release Train: Não, o repositório não apresenta releases em ciclos fixos e previsíveis.
- LTS + Current: Não, o repositório não apresenta versões de longo suporte (LTS) e manutenção paralela de versões estáveis e versões correntes.

Justificativa Final:
A estratégia Continuous Delivery é adequada para este projeto, pois ele permite releases frequentes e potencialmente automatizadas.
---
## 3. CONCLUSÃO SOBRE A GOVERNANÇA
A governança do projeto parece estar bem estabelecida, com um modelo de fluxo de trabalho de GitHub Flow e uma estratégia de release de Continuous Delivery. No entanto, a ausência de padronização de branches e releases pode ser um ponto de melhoria.
```

Figura 6: Relatório de governança gerado pelo modelo DeepSeek-Coder-6.7B-Instruct

11.6.1 Modelo de Fluxo de Trabalho

O modelo identificou o **GitHub Flow** como o modelo de fluxo de trabalho predominante no projeto analisado.

As evidências técnicas apontadas incluem a existência de uma única branch principal (*main*), acompanhada por diversas branches de curta duração associadas ao desenvolvimento de funcionalidades específicas. Além disso, foi observada alta frequência de integração dessas branches à branch principal, característica típica do GitHub Flow.

Na análise comparativa conduzida pelo modelo, alternativas como GitFlow e Trunk-Based Development foram descartadas. O GitFlow foi considerado inadequado devido à ausência de branches permanentes de *develop* e *release*, enquanto o Trunk-Based Development não foi identificado em função da presença explícita de branches de feature.

Dessa forma, a identificação do GitHub Flow mostra-se consistente com a estrutura observada no repositório, indicando um fluxo de desenvolvimento simplificado e orientado à integração contínua.

11.6.2 Estratégia de Release

Quanto à estratégia de release, o modelo apontou a adoção de **Continuous Delivery**.

Essa identificação baseou-se na presença recorrente de tags associadas às releases e na alta frequência de versionamento, sugerindo um processo de entrega contínua e potencialmente automatizado. O modelo destacou que as releases não seguem ciclos longos e rígidos, reforçando a hipótese de entregas frequentes.

Outras estratégias foram analisadas e descartadas, como Time-Based Releases, Feature-Based Releases, Rapid Releases e LTS + Current. A ausência de ciclos temporais fixos, de versões de longo suporte e de associação direta entre funcionalidades específicas e releases reforçou a classificação como Continuous Delivery.

11.6.3 Síntese da Governança Identificada

A combinação entre o GitHub Flow e a estratégia de Continuous Delivery sugere um nível de governança considerado adequado para projetos open source ativos, permitindo integração frequente, rápida evolução do código e entregas incrementais.

Entretanto, o modelo observou que a ausência de uma padronização mais rígida na nomenclatura de branches e na documentação do processo de release pode representar oportunidades de melhoria na maturidade da governança do projeto.

12 Conclusão

Este trabalho teve como objetivo analisar a governança do projeto open source *LangExtract* a partir da interpretação automática de dados técnicos extraídos de seu repositório Git, utilizando diferentes modelos de linguagem de grande porte como ferramentas de apoio à análise.

A aplicação de sete modelos distintos, todos submetidos ao mesmo contexto de entrada, ao mesmo prompt estruturado e aos mesmos parâmetros de inferência, permitiu a realização de uma comparação direta entre os resultados obtidos. A principal observação decorrente dessa análise comparativa foi a forte convergência das conclusões apresentadas pelos modelos.

Em relação ao modelo de fluxo de trabalho, todos os modelos analisados identificaram o **GitHub Flow** como o padrão predominante no projeto. Essa identificação baseou-se, de forma consistente, na presença de uma única branch principal (*main*), no uso recorrente de branches de curta duração associadas a funcionalidades específicas e na alta frequência de integração dessas branches ao ramo principal. A ausência de branches permanentes de *develop* e *release* levou os modelos a descartar o GitFlow, enquanto a existência explícita de branches de feature afastou a hipótese de Trunk-Based Development em sua definição estrita.

Quanto à estratégia de releases, houve também convergência significativa entre os modelos, que classificaram o projeto como adepto de **Continuous Delivery**. Essa conclusão foi sustentada principalmente pela presença recorrente de tags de versionamento, pela frequência relativamente elevada de releases e pela ausência de ciclos temporais rígidos ou versões de longo suporte (LTS). Alguns modelos reforçaram essa interpretação por meio de análises quantitativas, como o cálculo do intervalo médio entre releases, enquanto outros se apoiaram em análises qualitativas do padrão de versionamento observado.

Apesar da convergência nos resultados finais, observou-se variação na forma como os modelos justificaram suas conclusões. Alguns modelos apresentaram análises mais descritivas e estruturais, focando na organização das branches e no fluxo de merges, enquanto outros incorporaram métricas temporais e comparações mais explícitas entre estratégias

alternativas. Essa diferença evidencia que, embora os modelos cheguem a conclusões semelhantes, a profundidade e o estilo da argumentação podem variar significativamente.

De maneira geral, os resultados indicam que modelos de linguagem, quando utilizados de forma controlada e fundamentados em dados técnicos reais, podem atuar como ferramentas eficazes de apoio à análise de governança em projetos open source. A convergência das interpretações obtidas reforça a consistência do processo de análise adotado e sugere que a governança do projeto *LangExtract* apresenta um nível adequado de maturidade, caracterizado por integração frequente, entregas contínuas e um fluxo de trabalho simplificado.

Por fim, ressalta-se que os modelos de linguagem não substituem a análise humana, mas podem complementar o processo analítico, auxiliando na interpretação de grandes volumes de informações técnicas. Trabalhos futuros podem explorar a aplicação dessa abordagem em projetos com diferentes níveis de maturidade ou comparar os resultados obtidos com análises conduzidas exclusivamente por especialistas humanos.

Tabela 2: Comparação dos resultados obtidos pelos modelos de linguagem

Modelo	Workflow	Estratégia de Release	Observações
Mistral-7B-Instruct	GitHub Flow	Continuous Delivery	Análise equilibrada, com justificativas estruturais e temporais.
Zephyr-7B	GitHub Flow	Continuous Delivery	Ênfase em padrões de integração e simplicidade do fluxo.
Falcon-7B-Instruct	GitHub Flow	Continuous Delivery	Argumentação predominantemente qualitativa.
Gemma-7B-IT	GitHub Flow	Continuous Delivery	Identificação clara das evidências técnicas nas branches.
Qwen-2.5-7B-Instruct	GitHub Flow	Continuous Delivery	Uso explícito de métricas temporais entre releases.
Nous-Hermes-2	GitHub Flow	Continuous Delivery	Análise consistente, com foco na organização do repositório.
DeepSeek-Coder-6.7B	GitHub Flow	Continuous Delivery	Análise estrutural detalhada, com comparação entre modelos alternativos.