UNIVERSIDADE FEDERAL DE SERGIPE
DEPARTAMENTO DE COMPUTAÇÃO-DCOMP
Engenharia de Software II

Docente: Dr. Glauco de Figueiredo Carneiro
Discentes:
Álex Santos Alencar - 202300061518
Ellen Karolliny dos Santos - 202300114326
Gabriel Luiz Santos Gama Barreto - 202300114335
Gabriel Ramos de Carvalho - 202300061920
João Andryel Santos Menezes - 202300061652
Larissa Batista dos Santos - 202300061705
Paloma dos Santos - 202300061723
Rauany Ingrid Santos de Jesus - 202300061760

Atividade 2 – Gerenciamento de Releases e Fluxo no Software

Tabela 1 - Contribuição de cada membro da equipe.

Nome	**Matrícula**	**Descrição da atividade**
Álex Santos Alencar	202300061518	Realização da análise manual do projeto no GitHub.
Ellen Karolliny dos Santos	202300114326	Análise dos resultados do microsoft/Phi-3-mini
Gabriel Luiz Santos Gama Barreto	202300114335	Auxílio na construção do prompt dos modelos e análise dos relatórios gerados pelo deepseek coder.
Gabriel Ramos de Carvalho	202300061920	Ajuda na criação do prompt. Apresentação e discussão dos resultados obtidos a partir do MistralAI.
João Andryel Santos Menezes	202300061652	Escolha dos modelos e criação do prompt.
Larissa Batista dos Santos	202300061705	Auxílio na construção do tutorial, análise e apresentação dos resultados obtidos utilizando o modelo Qwen2.5.
Paloma dos Santos	202300061723	Comparação e análise dos modelos selecionados.
Rauany Ingrid Santos de Jesus	202300061760	Introdução de Estratégia de releases e modelo de fluxo de trabalho, bem como

	definição dos conceitos , exemplificação e definição do objetivo da atividade. Alimentação dos slides. Contribuição no documento de análise.

Fonte: Elaboração própria (2025)

1. Objetivo

O presente trabalho tem como objetivo identificar a estratégia de releases e o modelo de fluxo de trabalho do projeto trabalhado na atividade 1. Além disso, é apresentado o resultado da análise detalhada com as respectivas justificativas para a indicação da estratégia do projeto do DeepResearch.

2. Do que se trata o DeepResearch e por que escolhemos ele?

A escolha do LLM foi definida por meio de votação. Cada integrante do grupo analisou os repositórios disponíveis e selecionou aquele que considerou mais pertinente e interessante para o objetivo do trabalho. A partir dessa avaliação coletiva, decidiu-se, majoritariamente, pela análise da arquitetura do DeepResearch.

O DeepResearch, desenvolvido pela Alibaba, é um sistema de pesquisa autônoma baseado em agentes, capaz de realizar investigações profundas e multietapas. Ele combina raciocínio estruturado com ações externas, permitindo buscar informações na web, visitar páginas, extrair conteúdo, ler documentos, executar cálculos em Python e sintetizar tudo em análises coerentes. Sua operação envolve ciclos iterativos de pensar, agir e observar, o que possibilita verificação de fatos, integração de múltiplas fontes e produção de relatórios complexos e fundamentados.

3. Estratégias de Releases e Modelo do fluxo de trabalho

Em engenharia de software, a estratégia de releases e o modelo do fluxo de trabalho são elementos fundamentais para entregar um software de qualidade de forma consistente e previsível.

A estratégia de releases define como e quando novas versões do software serão disponibilizadas para os usuários. Existem alguns modelos, vamos apresentar os principais:

Release Contínuo: Funciona de forma onde cada alteração aprovada vai automaticamente para produção, sendo assim o processo é 100% automatizado do commit ao deploy. Ideal para aplicações web/SaaS, como Netflix e Spotify.

Release por Funcionalidade: Novas funcionalidades são agrupadas e liberadas juntas, permitindo "grandes lançamentos" de marketing, com ciclos que geralmente duram 2-6 semanas.

Release por Data: Os lançamentos são realizados em intervalos fixos (ex: a cada 3 meses), garantindo a previsibilidade para stakeholders.

Release por Versão: São desenvolvidas versões numeradas, incluindo versões major, minor e patch.

Já o modelo do fluxo de trabalho define o ciclo de vida do desenvolvimento de uma funcionalidade, do planejamento à produção. É definido e implementado considerando alguns fatores, como o tamanho da equipe, frequência de releases, riscos do negócio e cultura da equipe. Os modelos mais comuns são:
GitFlow: funciona como uma fábrica organizada com várias linhas - você trabalha em branches separadas para cada funcionalidade (feature), junta tudo no branch develop, prepara a versão final em branches release e corrige bugs de produção em hotfix, mantendo o main sempre estável.
GitHub Flow: funciona de maneira mais simples - você cria uma branch, faz a alteração, abre um pull request e, após aprovação, mescla direto na branch main que vai automaticamente para produção.
GitLab Flow: que adiciona ambientes - funcionalidades são criadas em branches secundárias e integradas à main, após isso, vai para o ambiente staging e só depois para produção.
Trunk-Based: que é o mais radical - todos codificam direto no main usando feature flags para controlar funcionalidades, com deploys contínuos e reversões instantâneas.
Cada modelo equilibra estrutura e agilidade conforme a necessidade da equipe - do mais controlado (GitFlow) ao mais ágil (Trunk-Based).

4. Análise manual

A avaliação manual da **estratégia de release** do projeto DeepResearch foi realizada por meio da inspeção direta dos elementos públicos do repositório no GitHub, incluindo a aba Releases, histórico de commits, uso de tags e padrões de versionamento.
Observa-se que o projeto não possui releases formais publicadas. A aba Releases está vazia, não há versões numeradas, notas de release ou pacotes explicitamente definidos para entrega. Além disso, não foram identificadas tags de versão de evolução do software.
O histórico de commits apresenta um fluxo contínuo de alterações, integradas de forma frequente à main. Não há evidências de ciclos de estabilização, congelamento de código ou preparação de versões, características que são comuns em Release Train, Rapid Releases ou LTS + Current.
A ausência de versionamento semântico e de releases empacotadas indica que o projeto não adota um modelo tradicional. Em vez disso, o código disponível no repositório representa constantemente o estado mais recente do desenvolvimento.

A análise manual do **modelo de fluxo de trabalho** foi realizada a partir da observação da estrutura de branches, pull requests e práticas de integração de código.
O repositório apresenta uma branch principal como centro do desenvolvimento, não observada a existência de branches permanentes adicionais, como develop, release ou hotfix. As contribuições são realizadas por meio de branches

temporárias, integradas posteriormente à main por pull requests. Esse padrão é característico do GitHub Flow, um modelo de fluxo de trabalho no qual existe uma única branch principal estável, novas funcionalidades ou correções são desenvolvidas em branches curtas, a integração ocorre por meio de pull requests revisados e o código integrado à branch principal está sempre potencialmente entregável.

5. Seleção e análise dos modelos

A escolha dos quatro modelos: DeepSeek-Coder, Mistral, Phi-3 e Qwen, baseia-se em uma estratégia de multiplas perspectivas, na qual cada arquitetura contribui com uma competência específica para a análise do DeepResearch. O DeepSeek funciona como o especialista técnico, focado em entender a lógica por trás de cada mudança no código. Já o Mistral 7B e o Qwen 2.5 7B oferecem um raciocínio mais generalista e boa flexibilidade linguística, o que os torna adequados para interpretar mensagens de commit, comentários e documentações. Por fim, o Phi-3 Mini se destaca pela eficiência, permitindo a análise de grandes janelas de contexto (até 128 mil tokens) com menor custo computacional. Em conjunto, esses modelos possibilitam uma avaliação da estratégia de branches do DeepResearch a partir de múltiplas perspectivas, combinando suas particularidades em prol da análise.

5.1. Modelo 1: *mistralai/Mistral-7B-Instruct-v0.3:*

1) **Estratégia de Releases:**

   O projeto não apresenta nenhum fluxo de trabalho definido para o desenvolvimento e entrega de código. A análise dos dados brutos não revelou a presença de nenhuma ferramenta de gerenciamento de código ou de fluxo de trabalho definido.

2) **Modelo de Fluxo de Trabalho (Git Workflow):**

   O projeto não apresenta uma estratégia de releases definida. A análise dos dados brutos não revelou a presença de nenhuma estratégia de releases definida.

3) **Resumo Geral:**

   O repositório DeepResearch contém um conjunto de código relacionado ao modelo Tongyi DeepResearch. Além disso, há tags (versões) no repositório, mas não foi possível identificar qual versão corresponde à última release. O histórico de commits mostra que houve várias atualizações no código, incluindo correção de erros, adição de recursos, e atualização de documentação. No entanto, não foi possível identificar uma estratégia de releases definida para o projeto.

5.2. Modelo 2: *microsoft/Phi-3-mini-128k-instruct:*

1) Estratégia de Releases:

   O repositório adota o modelo Trunk-Based Development, evidenciado pela presença exclusiva da branch main. A ausência de ramificações persistentes, como uma branch develop, confirma que o fluxo de trabalho prioriza integrações diretas na linha principal. Embora essa abordagem favoreça a velocidade, a falta de evidências de um fluxo de automação associado sugere que o processo depende fortemente da disciplina manual dos desenvolvedores para evitar a instabilidade do código.

2) **Modelo de Fluxo de Trabalho:**

   Atualmente, não existe uma estratégia de versionamento formal no projeto. A falta de tags de versão (como v1.0 ou v2.0) impossibilita a identificação de marcos estáveis de lançamento. Na prática, o software opera sem um histórico de releases definido, o que compromete a rastreabilidade e dificulta o gerenciamento de mudanças, uma vez que não há distinção clara entre o código em desenvolvimento e as versões prontas para uso.

3) **Resumo Geral:**

   O projeto utiliza um fluxo de trabalho simplificado (Trunk-Based), mas falha na gestão de configuração por não estabelecer uma política de lançamentos. Essa ausência de tags e ramificações de controle sugere um processo de entrega pouco maduro, o que pode prejudicar a escalabilidade e o rastreio de erros. Para alinhar o repositório às boas práticas de Engenharia de Software, recomenda-se a implementação de versionamento semântico e a criação de marcos de estabilidade para garantir maior controle sobre o ciclo de vida do software.

5.3. Modelo 3 : *deepseek-ai/deepseek-coder-6.7b-instruct:*

1) **Estratégia de Releases:**

   Diferente de softwares que utilizam versões semânticas, por exemplo *v1.0,* este projeto utiliza uma estratégia de releases baseada em marcos. Com o uso de *tags* como “1217” e “1101” (referentes a datas), indica que o projeto lança versões conforme atinge momentos específicos, permitindo o rastreio do histórico. Essa estratégia oferece aos usuários pontos de restauração eficientes, separando os commits experimentais (direto na *branch* main) dos lançamentos com *tags*.

2) **Modelo de Fluxo de Trabalho:**

O modelo identificou um modelo de trabalho dinâmico, possivelmente centrado em um desenvolvedor principal com uma equipe pequena e sincronizada. O fluxo possui a presença de *reverts* (reversões) frequentes que indicam um ambiente com código sempre funcional, corrigindo falhas de integração de forma imediata.

3) **Resumo Geral:**

O DeepResearch tem um processo otimizado para colaboração, utilizando o histórico de commits e merges para evitar gargalos. No qual, é frequentemente mesclado e revertido para evitar conflitos e rastrear as alterações facilmente. Existem também muitas releases significantes, indicando potenciais lançamentos de produto.

5.4. Modelo 4: *Qwen/Qwen2.5-7B-Instruct:*

1) **Estratégia de Releases:**

A estratégia de releases não está clara. As tags de versão não foram listadas, e o histórico de commits sugere uma série de merges e atualizações sem uma estrutura clara de versões.

2) **Modelo de Fluxo de Trabalho (Git Workflow):**

Não há workflow de CI/CD encontrado. O relatório indica que nenhum workflow de CI/CD foi encontrado no repositório.

3) **Resumo Geral:**

O projeto parece estar em desenvolvimento ativo com várias atualizações recentes. O repositório contém várias branches e commits recentes, indicando atividade de desenvolvimento. No entanto, a ausência de workflows de CI/CD e a falta de tags de versão sugerem que a organização e a estrutura do projeto ainda estão em evolução. Além disso, o README.md apresenta vários links importantes, indicando que o projeto está sendo promovido publicamente, mas a falta de versões claras pode dificultar o gerenciamento e a documentação do projeto.

6. Comparação dos modelos e sua análise

A Tabela 1 apresenta a comparação entre os quatro modelos de linguagem analisados, considerando a Estratégia de Releases, o Workflow (Git), a maturidade processual e o ponto forte da análise.

Tabela 1- Comparação entre os Modelos.

Critério	*Modelo 1 (Mistral-7B)*	*Modelo 2 (Phi-3-mini)*	*Modelo 3 (DeepSeek-Code)*	*Modelo 4 (Qwen2.5-7B)*
Estratégia de Releases	**Inexistente;** não foi possível identificar ferramentas de gerenciamento ou a versão da última release, embora mencione a existência de tags.	**Ausente**; crítica a falta de SemVer (Versionamento Semântico), classificando-o como "eterno beta".	**Identifica uma estratégia baseada em marcos** (milestones) com tags datadas (ex: tags "1217",1101"), interpretando-as como pontos de restauração e separação de código experimental	**Inconclusiva**; Relata que a estratégia não está clara e que as tags não foram listadas, focando na falta de estrutura.
Workflow (Git)	**Sem fluxo definido** ou ferramentas de CI/CD.	**Desenvolvimento manual/*ad-hoc***; alto risco de conflitos de merge por falta de padrões (Git Flow/Trunk-Based) e ausência crítica de CI/CD.	**Fluxo dinâmico e centralizado**, destacando o uso de **reversões (reverts) frequentes** como tática para manter o código funcional.	**Ausência de CI/CD**, mas nota a presença de várias branches e atividade intensa de merges, sugerindo um projeto em evolução.
Maturidade Processual	**Baixa;** foco apenas no histórico de commits, mas mantém a visão de desorganização.	**Baixa/Crítica**; aponta uma análise crítica de **negócio/engenharia**, mencionando "Débito Técnico de Processo".	**Alta/Otimizada**; vê eficiência na colaboração e merges. Eficiente para evitar gargalos.	**Em evolução;**Observa o contraste entre um projeto "promovido publicamente" (via README) e a falta de rigor técnico na estruturação de versões.
Ponto Forte da Análise	Descrição básica do histórico de commits.	Profundidade técnica sobre riscos de engenharia de software.	Capacidade de extração de dados específicos (tags e padrões).	Observação da discrepância entre marketing (README) e estrutura.

Fonte: Elaborado pelo próprio Autor (2025).

Os modelos mais efetivos foram o **Modelo 3 (DeepSeek)** e o **Modelo 2 (Phi-3)**, por motivos complementares:

1º Lugar: Modelo 3 (deepseek-ai/deepseek-coder-6.7b-instruct)

- **Justificativa:** Foi o único modelo que demonstrou uma **capacidade analítica superior de extração de dados brutos**. Enquanto os Modelos 1, 2 e 4 afirmaram que não havia tags ou que a estratégia era inexistente, o DeepSeek identificou o padrão específico de versionamento do Alibaba-NLP, como tags datadas como "1217". Ele conseguiu interpretar o comportamento real do repositório, como uso de reverts para estabilidade, em vez de apenas reportar a ausência de padrões convencionais.

2º Lugar: Modelo 2 (microsoft/Phi-3-mini-128k-instruct)

- **Justificativa:** Embora tenha falhado em ver as tags datadas, ele foi o mais efetivo na **análise qualitativa e de risco**. Ele utilizou terminologia técnica precisa como o SemVer, Baselines, CI/CD e Débito Técnico para diagnosticar a saúde do projeto, oferecendo recomendações práticas (CONTRIBUTING.md) que agregam valor real a uma auditoria de software.

Em suma, o **Modelo 3 (deepseek-ai/deepseek-coder-6.7b-instruct)** é o mais efetivo para **identificar o que existe** (mesmo que fora do padrão), sendo essencial para entender a operação real do projeto. O **Modelo 2 (microsoft/Phi-3-mini-128k-instruct)** é o mais efetivo para **avaliar a maturidade**, sendo ideal para um relatório de governança ou engenharia de software. Os Modelos 1 e 4 foram menos efetivos por apresentarem conclusões genéricas de "não encontrado". Foram muito superficiais, tratando a ausência de padrões comuns (como SemVer e CI/CD) como uma "ausência de dados", quando na verdade a ausência de padrão é um dado sobre a cultura do projeto.

7. Limitações e Dificuldades

O uso dos modelos de 7 bilhões de parâmetros no Google Colab acaba sendo um desafio principalmente por causa da VRAM limitada das GPUs disponíveis (cerca de 16GB na GPU T4). Mesmo utilizando quantização em 4 bits para melhorar a eficiência reduzindo o tamanho do modelo, ele ainda ocupa quase toda a memória, de forma que se deixa pouco espaço para processar os arquivos, causando lentidão e falta de memória. Isso pesa bastante quando o modelo a ser analisado é grande, como o DeepResearch, que possui várias pastas e módulos diferentes.

Outra limitação detectada durante a análise, é que modelos 7B, apesar de úteis como demonstrado nessa pesquisa, não têm capacidade suficiente para manter uma visão ampla do projeto. Ou seja, como só conseguem trabalhar com trechos menores e têm uma permanência de contexto mais limitada, dificultando integrar todas as partes da arquitetura para gerar uma análise completa.

8. Tutorial

8.1. Acesso ao repositório completo

Para visualizar todos os artefatos utilizados na análise e também demonstrados neste tutorial, disponibilizou-se um repositório que reúne os códigos-fonte, arquivos de apoio, registros gerados e análises detalhadas. O acesso aos seguintes materiais pode ser realizado pelo endereço: https://github.com/GabrielGamaUFS/Evolucao_Software_2025-2_DeepResearch_atividade2

Endereço do vídeo de apresentação da análise LLM e seus resultados obtidos: https://drive.google.com/file/d/1mzDET2fs4TX35zRRkTenTXPzucnlFj1w/view?usp=drive_link

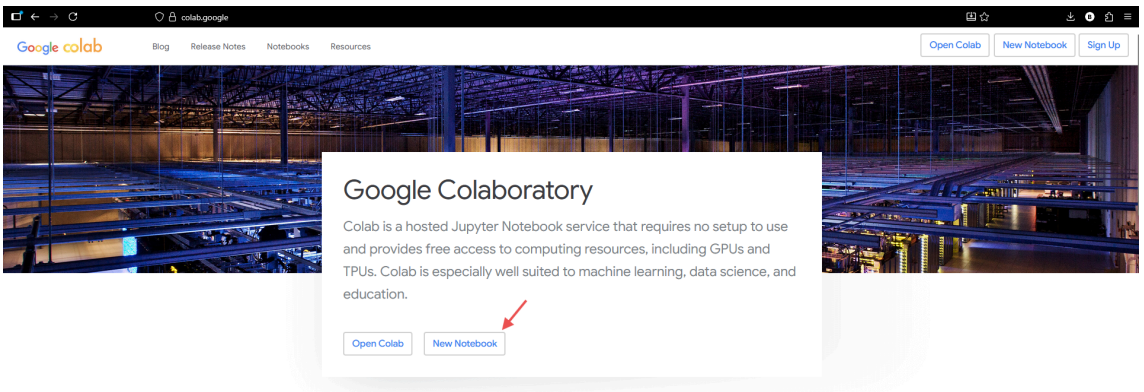
8.2. Tutorial e Replicabilidade: Análise da Estratégia de Releases e Modelo do fluxo de trabalho

Este tutorial apresenta como objetivo demonstrar o processo, passo a passo, de como identificar a estratégia de release e o modelo do fluxo de trabalho do projeto *DeepResearch* a partir de seu repositório no github, utilizando e simulando quatro grandes modelos de linguagem (LLMs), executados a partir do Google Colab.

8.3. Abertura do Ambiente Google Colab (IDE)

Nesta etapa, deve-se acessar o ambiente Google Colab, disponível no endereço https://colab.google/, e criar um "Novo Notebook" ou "*New Notebook*".

Figura 1 - Acesso ao notebook do Google Colab

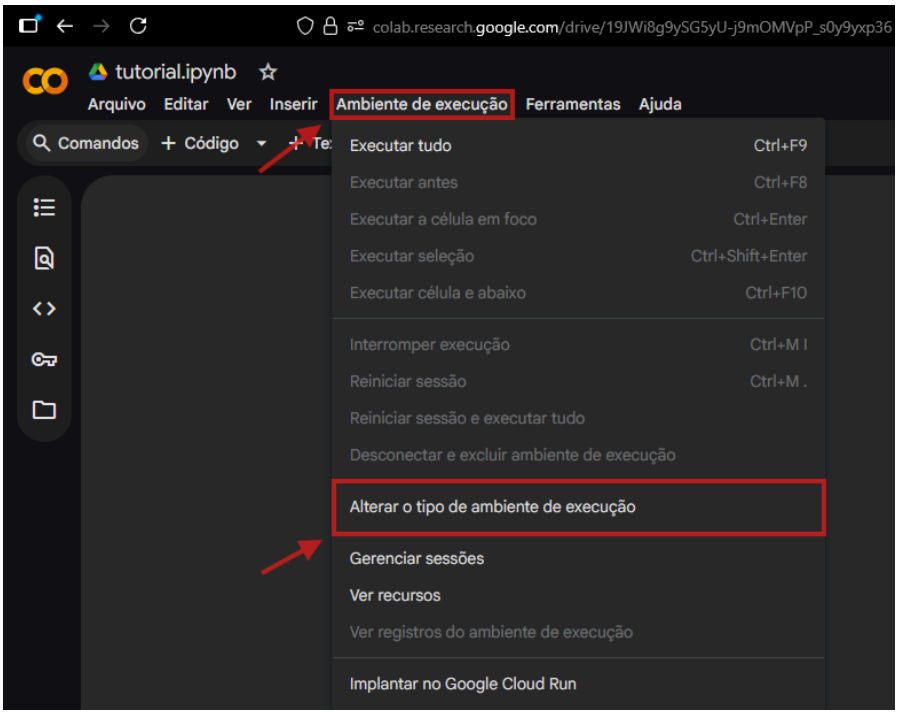


Fonte: Elaboração própria (2025)

8.4. Preparação do Ambiente

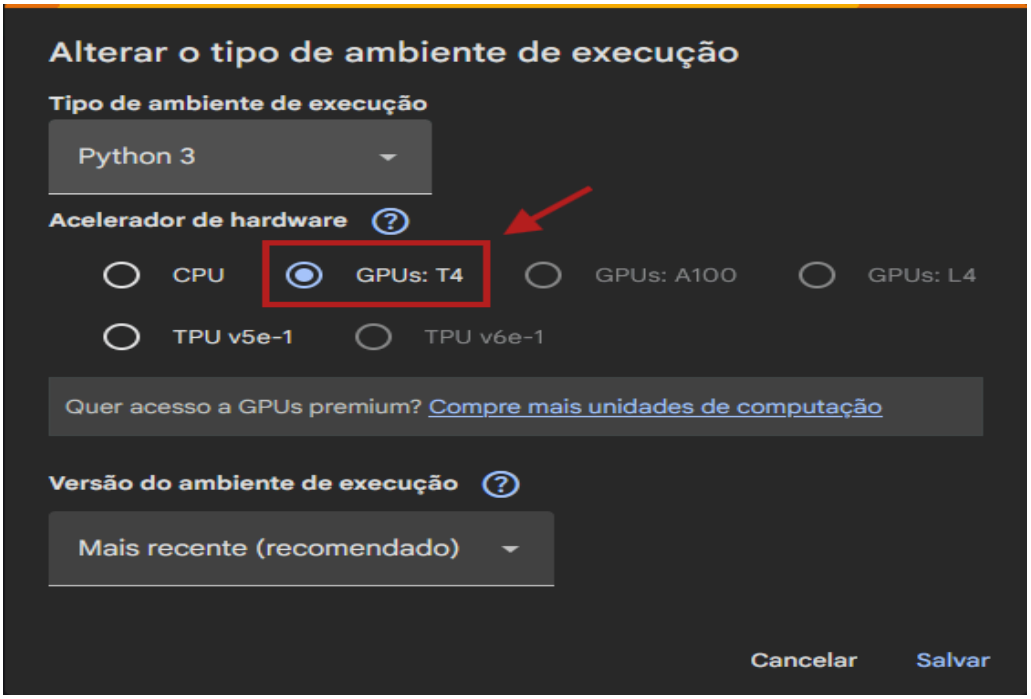
Nesse momento é importante definir o uso da GPU no Colab. Para isso, acesse o menu "Ambiente de Execução" na parte superior da página, em seguida pressione "Alterar o tipo de ambiente de execução", selecione a opção "GPUs: T4" e clique em "salvar", como segue as figuras:

Figura 2 - Acessar ambiente de execução



Fonte: Elaboração própria (2025)

Figura 3 - Alterar o tipo de ambiente de execução

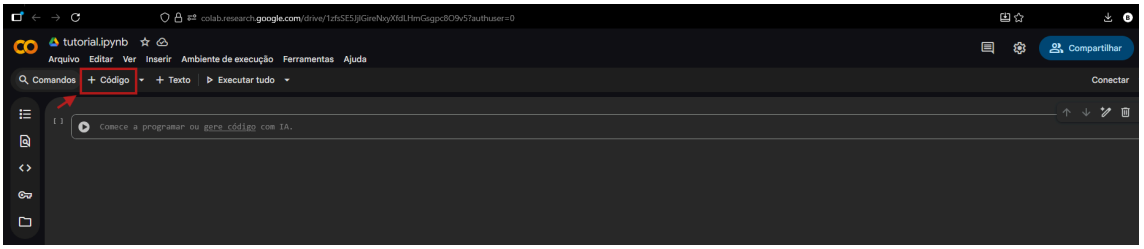


Fonte: Elaboração própria (2025)

8.5. Inserção do código-fonte

No ambiente do Google Colab, selecione a célula do código já existente. Caso não haja uma célula já criada, pressione a opção "+ Código" para inserir uma nova.

Figura 4 - Criação de célula



Fonte: Elaboração própria (2025)

Em seguida, cole o trecho do código abaixo:

Optamos por separar o código em quatro células para maior organização:

```
# 1. Instalar as bibliotecas necessárias

!pip install transformers accelerate torch bitsandbytes
```

```
import os

# Define o diretório de destino no Colab
repo_dir = "/content/DeepResearch"

# Verifica se a pasta já existe antes de clonar
if not os.path.exists(repo_dir):
   print(f"A clonar https://github.com/Alibaba-NLP/DeepResearch para
{repo_dir}...")

   !git clone https://github.com/Alibaba-NLP/DeepResearch.git
   print("Repositório clonado com sucesso.")
else:
   print(f"Repositório já existe em {repo_dir}.")
```

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

# Alterar pelo modelo escolhido:
model_id = "Qwen/Qwen2.5-7B-Instruct"
print(f"Carregando {model_id} em 4-bit...")
bnb_config = BitsAndBytesConfig(
```

```
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.bfloat16
)

# Carregar o tokenizador
tokenizer = AutoTokenizer.from_pretrained(model_id)

# Carregar o modelo aplicando a configuração de 4-bit
# Aplicar a configuração de 4-bit
# "auto" coloca o modelo na GPU
model = AutoModelForCausalLM.from_pretrained(
    model_id,
    quantization_config=bnb_config,
    device_map="auto"
)

print("------------------------------------------------------------")
print(f"Modelo {model_id} carregado com sucesso em 4-bit!")
print("------------------------------------------------------------")
```

```
import subprocess
import glob

# --- 1. FUNÇÃO PARA EXTRAIR INFORMAÇÕES DO REPOSITÓRIO ---
def get_repo_context(repo_path):
    context_data = ""

    # A. Listar Branches e Tags (Indica versionamento)
    try:
        branches = subprocess.check_output(["git", "-C", repo_path,
"branch", "-r"], text=True)
        tags = subprocess.check_output(["git", "-C", repo_path,
"tag"], text=True)
        context_data += f"--- BRANCHES REMOTAS ---\n{branches}\n"
        context_data += f"--- TAGS (VERSÕES) ---\n{tags}\n"
    except Exception as e:
        context_data += f"Erro ao ler git info: {e}\n"

    # B. Ler os últimos commits (Indica padrão de commit e merge)
    try:
        # Pega os últimos 20 commits formatados para mostrar merges
        logs = subprocess.check_output(
            ["git", "-C", repo_path, "log", "--graph", "--oneline",
```

```
"-n", "20"],
            text=True
        )
        context_data += f"--- HISTÓRICO DE COMMITS (GRÁFICO)
---\n{logs}\n"
    except:
        pass

    # C. Verificar Workflows do GitHub (Indica CI/CD e Release
Automatizada)
    workflows = glob.glob(f"{repo_path}/.github/workflows/*.yml") +
glob.glob(f"{repo_path}/.github/workflows/*.yaml")
    if workflows:
        context_data += "--- ARQUIVOS DE WORKFLOW (CI/CD)
ENCONTRADOS ---\n"
        for wf in workflows:
            filename = os.path.basename(wf)
            context_data += f"Nome do arquivo: {filename}\n"
            # Lê o conteúdo dos workflows para entender o que eles
fazem (ex: publish release)
            with open(wf, 'r') as f:
                context_data += f"Conteúdo de
{filename}:\n{f.read()}\n\n"
    else:
        context_data += "--- NENHUM WORKFLOW DE CI/CD ENCONTRADO
---\n"

    # D. Ler README e CONTRIBUTING (Busca regras escritas)
    for doc in ["README.md", "CONTRIBUTING.md", "RELEASE.md"]:
        path = os.path.join(repo_path, doc)
        if os.path.exists(path):
            with open(path, 'r') as f:
                content = f.read()
                # Trunca se for muito grande para não estourar o
contexto
                context_data += f"--- ARQUIVO {doc}
---\n{content[:2000]}...\n\n"

    return context_data

# --- 2. PREPARAR O PROMPT ---
print("Coletando dados do repositório...")
repo_context = get_repo_context(repo_dir)

# Ajustamos o texto para ser mais diretivo e evitar que o modelo se
alongue demais
```

```
instrucao_tarefa = """Você é um Auditor de Código Sênior e
Especialista em DevOps.
Sua tarefa é analisar os dados brutos de um repositório Git
fornecidos abaixo e extrair fatos reais.
Seja conciso e objetivo em cada ponto."""

corpo_dados = f"""Aqui estão os dados extraídos do repositório:
{repo_context}

Por favor, gere a análise detalhada seguindo EXATAMENTE o formato
abaixo."""

# Note que terminamos o prompt com o título do relatório para
"puxar" a resposta do modelo
prompt = f"""### Instruction:
{instrucao_tarefa}

{corpo_dados}

### FORMATO DE RESPOSTA ESPERADO:
## Relatório: DeepResearch

**1. Modelo de Fluxo de Trabalho:**
* Veredito:
* Justificativa:

**2. Estratégia de Releases:**
* Veredito:
* Justificativa:

**3. Resumo Geral:**

### Response:
## Relatório: DeepResearch"""

# --- EXECUTAR A INFERÊNCIA ---
print("Gerando análise...")

input_ids = tokenizer(prompt, return_tensors="pt").to("cuda")

outputs = model.generate(
    **input_ids,
    max_new_tokens=1024,   # Limite da resposta
    temperature=0.2,       # Baixa para manter o foco técnico
    repetition_penalty=1.1, # Evita que ele repita as instruções do
prompt
```

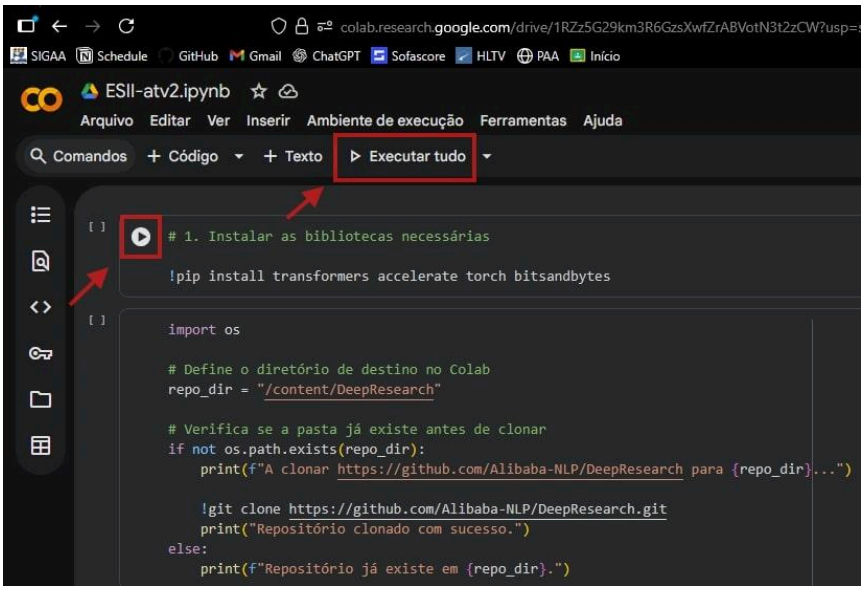
```
    do_sample=True,
    top_p=0.9,
    eos_token_id=tokenizer.eos_token_id
)

# Decodificando
response =
tokenizer.decode(outputs[0][input_ids.input_ids.shape[-1]:],
skip_special_tokens=True)

# Exibimos o título manualmente já que o usamos para induzir a
resposta
print("\n" + "="*50)
print("## Relatório: DeepResearch" + response)
print("="*50)
```

Fonte: Elaboração própria (2025)

8.6. Execução dos Modelos de Linguagem (LLMs)

Após a inserção do código, pressione o botão “Executar célula” (ícone de *play*) sequencialmente em cada bloco, ou “Executar tudo” para processar o modelo e aguarde a conclusão da análise realizada pelo LLM.

Figura 5 - Execução do código

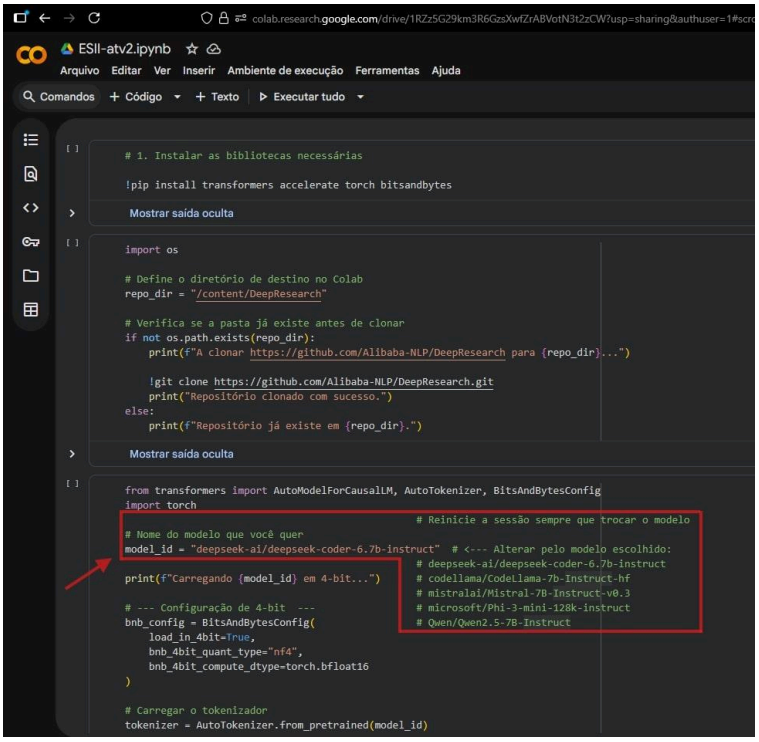


Fonte: Elaboração própria (2025)

8.7. Repetição da simulação

O procedimento pode ser repetido para cada um dos quatro modelos de linguagem utilizados na simulação, bastando substituir, na linha indicada abaixo, pelo modelo preferido.

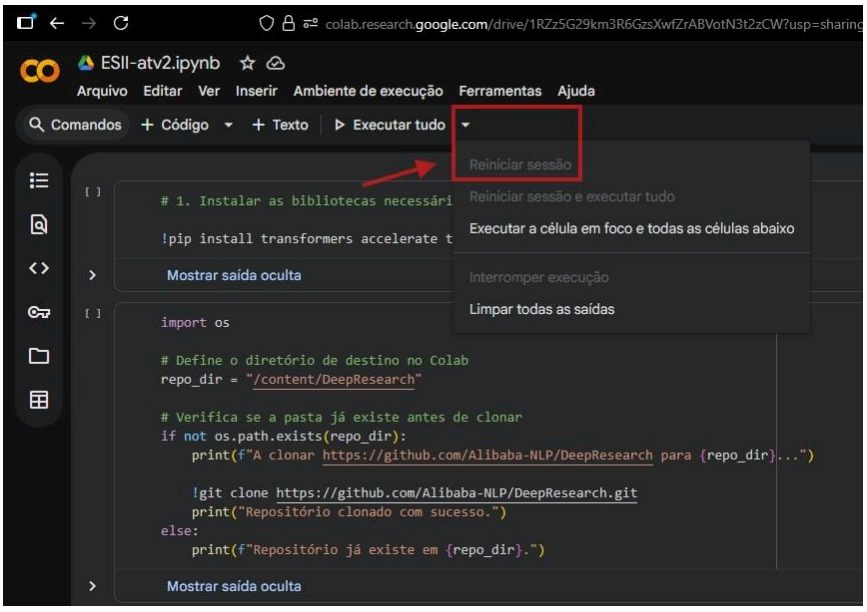
Figura 6 - Substituição do LLM



Fonte: Elaboração própria (2025)

Após a substituição do modelo, é necessário reiniciar a sessão do ambiente, Para isso, clique na seta para baixo (“Mais ações”) e em seguida “Reinicar sessão”. Logo depois, retorne ao passo anterior.

Figura 7 - Reinício da sessão



Fonte: Elaboração própria (2025)

9. Referências

Yao, S., Zhao, J., Yu, D., Shafran, I., Griffiths, T. L., Cao, Y., & Narasimhan, K. (2023). ReAct: Synergizing reasoning and acting in language models. arXiv preprint arXiv:2210.03629. https://arxiv.org/abs/2210.03629

INFOQ. Architecture & Design. Disponível em: https://www.infoq.com/architecture-design/. Acesso em: 12 nov. 2025.