



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 1 – Padrões Arquiteturais de Software
Tutorial

O presente tutorial, tem como objetivo descrever resumidamente cada ação praticada no desenvolvimento do trabalho.

1. Participação da Equipe

Na tabela abaixo, serão apresentados os autores do trabalho, suas respectivas matrículas e uma breve descrição da atividade desenvolvida.

Nome	Matrícula	Descrição da atividade
Álex Santos Alencar	202300061518	Realização da análise manual do projeto no GitHub.
Ellen Karolliny dos Santos	202300114326	Definição sobre padrões arquiteturais, exemplos sobre os padrões mais conhecidos.
Gabriel Luiz Santos Gama Barreto	202300114335	Auxílio na construção do prompt dos modelos e do tutorial. Análise dos relatórios gerados pelo deepseek e codellama.
Gabriel Ramos de Carvalho	202300061920	Ajuda na escolha dos modelos e criação do prompt. Apresentação e discussão

<i>João Andryel Santos Menezes</i>		dos resultados obtidos a partir do MistralAI.
	202300061652	Escolha dos modelos, criação do prompt. Análise dos resultados e apresentação do Phi 3 mini.
<i>Larissa Batista dos Santos</i>	202300061705	Análise e apresentação dos resultados obtidos utilizando o modelo Qwen2.5.
<i>Paloma dos Santos</i>	202300061723	Comparação e análise dos modelos selecionados. Ajuda na criação do modelo do documento .docx (Resposta da análise e tutorial).
<i>Rauany Ingrid Santos de Jesus</i>	202300061760	Introdução de padrões arquiteturais, auxílio na criação do doc de análise, desenvolvimento dos slides e edição do vídeo.

2. 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/Engenharia_SoftwareII_2025-2_T04_DeepResearch.git

Endereço do vídeo de apresentação da análise LLM e seus resultados obtidos:
<https://drive.google.com/file/d/1LWLBEVQrYNaxog7Xz0Lq3KmnPvoMZx3o/view?usp=sharing>

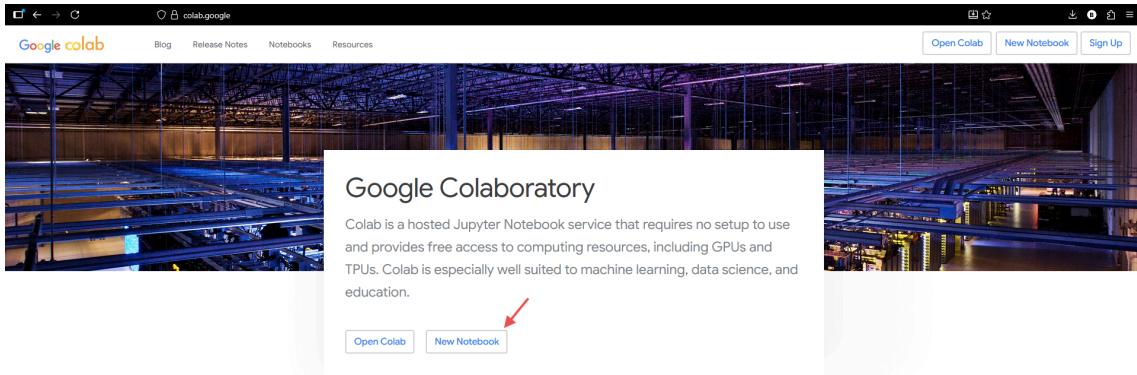
3. Tutorial: Análise de Padrões Arquiteturais

Este tutorial apresenta como objetivo demonstrar o processo, passo a passo, de como identificar os padrões arquiteturais do modelo de linguagem *DeepResearch* a partir de seu repositório no github, utilizando e simulando três grandes modelos de linguagem (LLMs), executados a partir do Google Colab.

3.1. 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

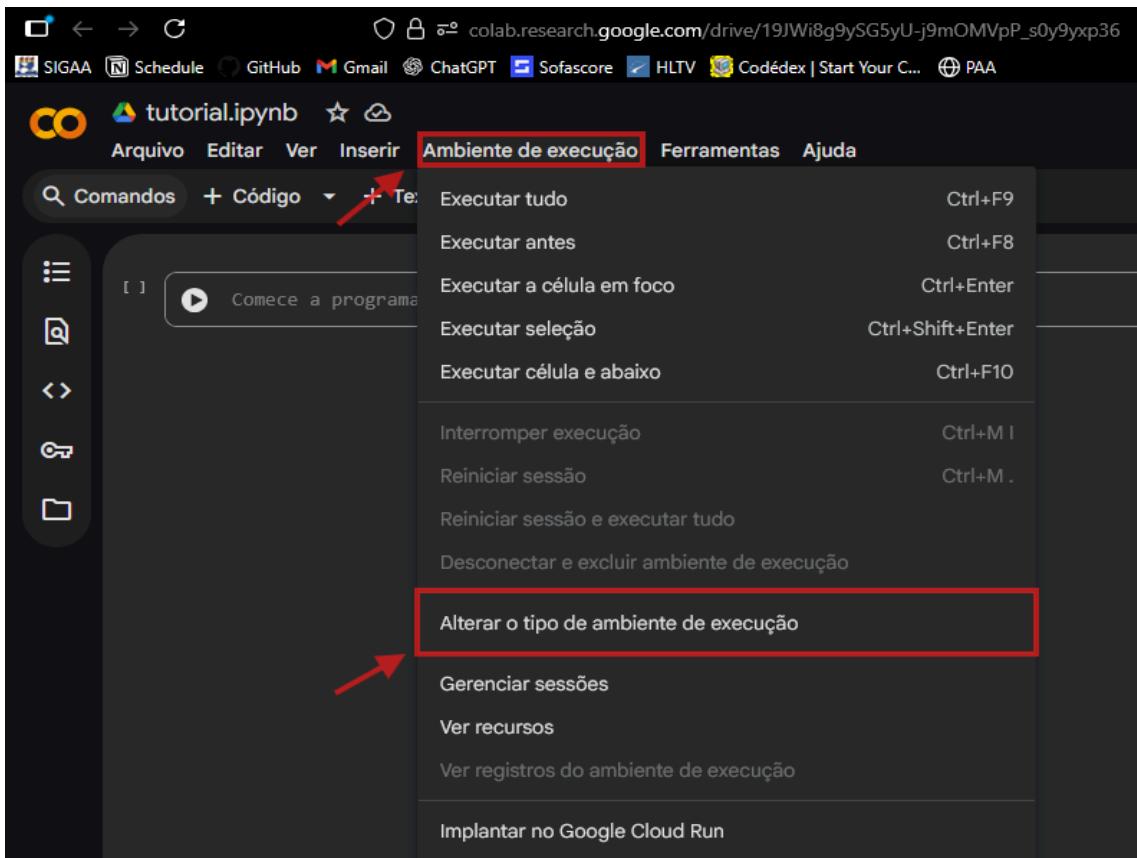


Fonte: Elaboração própria (2025)

3.2. Preparação do Ambiente

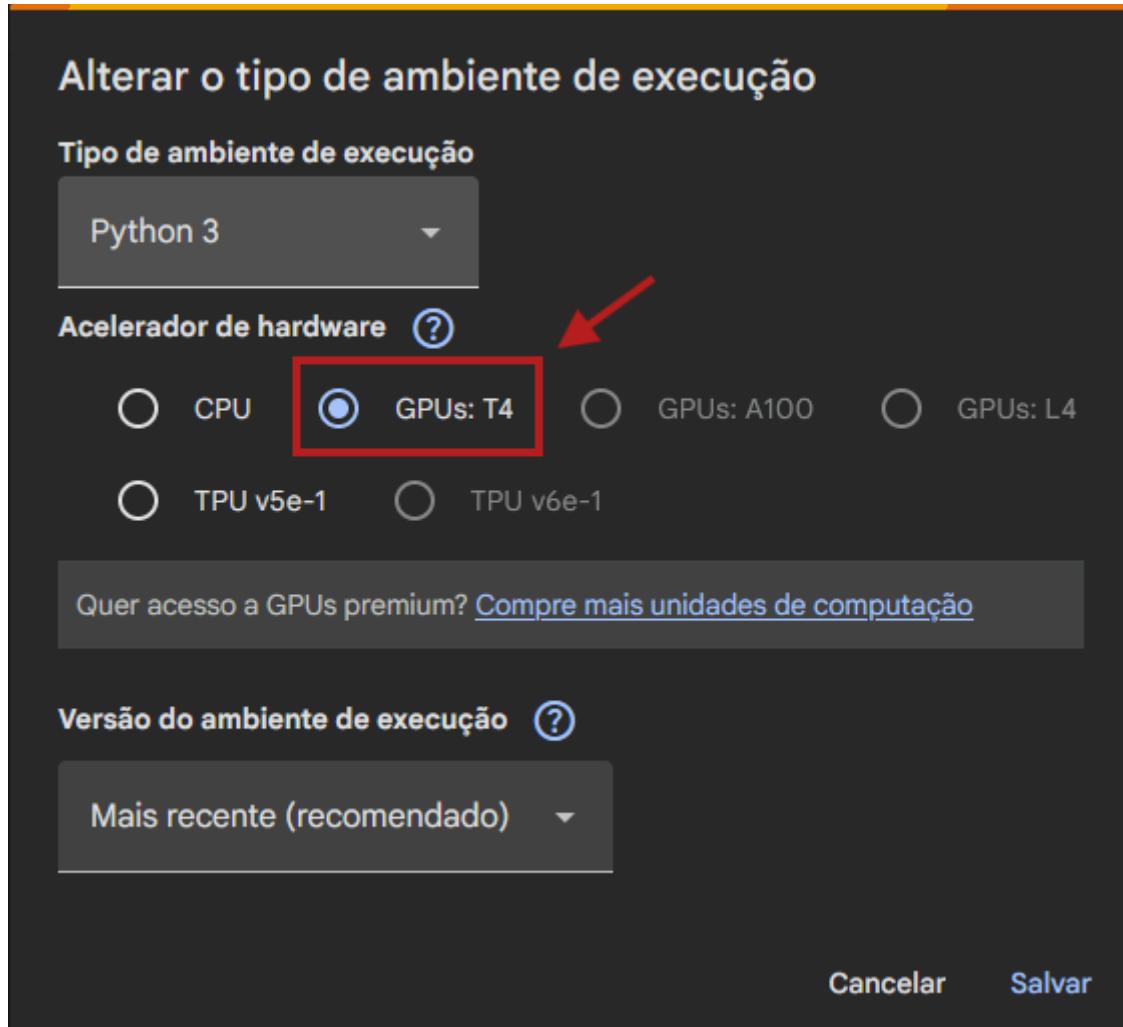
Nesse momento é importante definir o uso da GPU no Colab. Para isso, acesse o menu “Ambiente de Execução” na parte superior da página, em seguida pressione “Alterar o tipo de ambiente de execução”, selecione a opção “GPUs: T4” e clique em “salvar”, como segue as figuras:

Figura 2 - Acessar ambiente de execução.



Fonte: Elaboração própria (2025)

Figura 3 - Alterar o tipo de ambiente de execução.

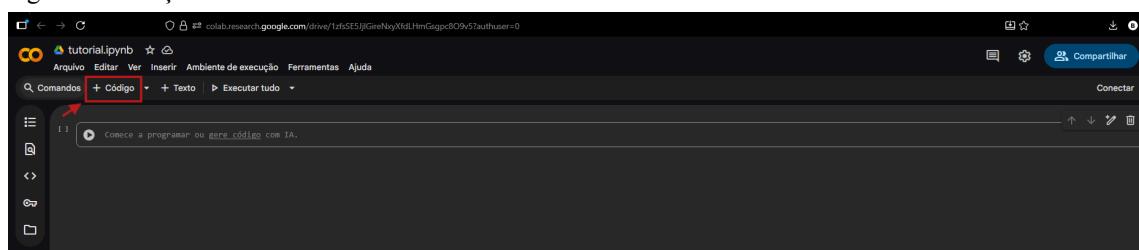


Fonte: Elaboração própria (2025)

3.3. 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:

Código 1 - Execução do modelo

```
# 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

# Reinicie a sessão
sempre que trocar o modelo
# Nome do modelo que você quer
model_id = "mistralai/Mistral-7B-Instruct-v0.3" # <--- Alterar pelo
modelo escolhido:
# deepseek-ai/deepseek-coder-6.7b-instruct
print(f"Carregando {model_id} em 4-bit...") # codellama/CodeLlama-7b-Instruct-hf
# mistralai/Mistral-7B-Instruct-v0.3
# --- Configuração de 4-bit ---
# microsoft/Phi-3-mini-128k-instruct
bnb_config = BitsAndBytesConfig(
Qwen/Qwen2.5-7B-Instruct
    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
```

```

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

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

import os
import subprocess

# --- PASSO 1: COLETAR OS METADADOS ---

repo_path = "/content/DeepResearch"
print(f"Coletando metadados do repositório em: {repo_path}")

# 1.1 Coletar Estrutura de Pastas
folder_structure = ""
for root, dirs, files in os.walk(repo_path, topdown=True):
    # Limitar a profundidade da árvore (nível 0, 1 e 2)
    depth = root.replace(repo_path, '').count(os.sep)
    if depth > 2:
        dirs[:] = [] # Não explorar mais fundo
        continue

    # Ignorar a pasta .git
    if '.git' in dirs:
        dirs.remove('.git')

    # Formatar o nome da pasta
    indent = " " * depth
    base_name = os.path.basename(root) if depth > 0 else 'DeepResearch'
    folder_structure += f"{indent}-{base_name}/\n"

print("... Estrutura de Pastas coletada.")

# 1.2 Ler o README.md
try:
    with open(os.path.join(repo_path, "README.md"), 'r',
encoding='utf-8') as f:
        readme_content = f.read(3000) + "\n... (README truncado)"
        print("... README.md coletado.")
except Exception as e:
    readme_content = f"Erro ao ler README: {e}"

# 1.3 Ler o requirements.txt
try:
    with open(os.path.join(repo_path, "requirements.txt"), 'r',
encoding='utf-8') as f:
        requirements_content = f.read()

```

```

        print("... requirements.txt coletado.")
    except Exception as e:
        requirements_content = f"Erro ao ler requirements.txt: {e}"

# 1.4 Coletar os logs de commit
try:
    log_command = ["git", "log", "--oneline", "-n", "20"] # 20 commits
    mais recentes
    result      = subprocess.run(log_command, cwd=repo_path,
capture_output=True, text=True, check=True, encoding='utf-8')
    git_log_content = result.stdout
    print("... Logs de Commit (últimos 20) coletados.")
except Exception as e:
    git_log_content = f"Erro ao ler logs do Git: {e}"

# 1.5 Ler o setup.py
try:
    with open(os.path.join(repo_path, "setup.py"), 'r',
encoding='utf-8') as f:
        setup_content = f.read(1000) + "\n... (setup.py truncado)"
    print("... setup.py coletado.")
except Exception as e:
    setup_content = "setup.py não encontrado ou erro."

# 1.6 Ler o Ponto de Entrada da API
api_main_path = "WebAgent/WebSailor/main.py"
try:
    with open(os.path.join(repo_path, api_main_path), 'r',
encoding='utf-8') as f:
        api_main_content = f.read(2000) + f"\n... ({api_main_path} truncado)"
    print(f"... {api_main_path} coletado.")
except Exception as e:
    api_main_content = f"{api_main_path} não encontrado ou erro."


# --- PASSO 2: CONSTRUIR O PROMPT ARQUITETURAL ---

prompt_arquitetural = f"""
[INST]
Você é um Arquiteto de Software Sênior. Sua tarefa é analisar os
seguintes metadados e **trechos de código** de um repositório para
identificar os Padrões Arquiteturais.

Preste muita atenção em como os arquivos de código (como o `main.py`)
**importam e usam** outros módulos.

---
### DADOS DO REPOSITÓRIO PARA ANÁLISE ###

#### 1. README.md (Truncado) ####
{readme_content}

```

```
#### 2. requirements.txt (Dependências) ####
{requirements_content}

#### 3. Estrutura de Pastas (Nível 2) ####
{folder_structure}

#### 4. Logs de Commit Recentes ####
{git_log_content}

#### 5. Conteúdo do setup.py (Define o Pacote) ####
{setup_content}

#### 6. Conteúdo do Ponto de Entrada da API ({api_main_path}) ####
{api_main_content}
```

RELATÓRIO DE HIPÓTESE ARQUITETURAL

Baseado **apenas** nos dados acima, responda:

Responda **precisamente** às seguintes perguntas, baseando-se **apenas** nas evidências fornecidas:

- ①. *Qual é o Padrão de Ponto de Entrada?*
* Qual tecnologia principal é usada como interface do sistema?
* *Justifique* analisando o arquivo {api_main_path} e o requirements.txt.
- ②. *Qual é o Padrão de Estrutura de Código?*
* *Justifique* sua escolha.
- ③. *Qual é o Padrão de Implantação?*
* Ao analise o código do `{api_main_path}` e `{folder_structure}`
- ④. *Resumo da Arquitetura:*
 - * Combine suas três respostas acima em uma descrição coesa da arquitetura geral.

```
[/INST]
"""

print("\nPrompt de análise arquitetural construído e pronto para envio.")

# --- PASSO 3: CHAMAR O LLM ---

print("Enviando prompt de análise arquitetural para o LLM...")

try:
```

```

        inputs = tokenizer(prompt_arquitetural, return_tensors="pt",
truncation=True, max_length=8192).to("cuda") # 8k para CodeLlama/Mistral

        output = model.generate(
            **inputs,
            max_new_tokens=2024, # Espaço para o relatório
            pad_token_id=tokenizer.eos_token_id,
            temperature=0.7 # Temperatura mais baixa ajuda a evitar
alucinações
        )

        response = tokenizer.decode(output[0][inputs.input_ids.shape[1]:],
skip_special_tokens=True)

        print("\n\n--- RELATÓRIO ARQUITETURAL DO LLM ---")
        print(response)

except NameError as ne:
    print(f"\nErro de Nome: {ne}")
except Exception as e:
    print(f"\nErro ao rodar a análise: {e}")

```

Fonte: Elaboração própria (2025)

3.4. Execução dos Modelos de Linguagem (LLMs)

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

Figura 5 - Execução do código

```

tutorial.ipynb
Arquivo Editar Ver Inserir Ambiente de execução Ferramentas Ajuda
Comandos Código + Texto ▶ Executar tudo
I 1 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("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("Repositório já existe em (repo_dir).")

from transformers import AutoModelForCausalLM, AutoTokenizer, BitsAndBytesConfig
import torch
# Nome do modelo que você quer
model_id = "mistralai/Mistral-7B-Instruct-v0.3" # ... Alterar pelo modelo escolhido:
# bnb_config = BitsAndBytesConfig( # deepseek-a1/deepspeed-coder-a1-7b-Instruct
#     load_in_4bit=True, # alpaca-lm/CodeLlama-7B-Instruct-v0.3
#     bnb_4bit_compute_dtype="bf16", # microsoft/DialoGPT-mini-instruct
#     bnb_4bit_compute_dtype="bf16" # Qwen/Qwen2.5-7B_Instruct
# )

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

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

```

Fonte: Elaboração própria (2025)

3.5. Repetição da simulação

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

Figura 6 - Substituição do LLM

```

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克隆ado com sucesso.")
else:
    print("Repositório já existe em {repo_dir}.")

from transformers import AutoModelForCausalLM, AutoTokenizer, BitsAndBytesConfig
import torch

# Nome do modelo que você quer
model_id = "mistralai/Mistral-7B-Instruct-v0.3" # ----- Alterar pelo modelo escolhido:
# 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
# )

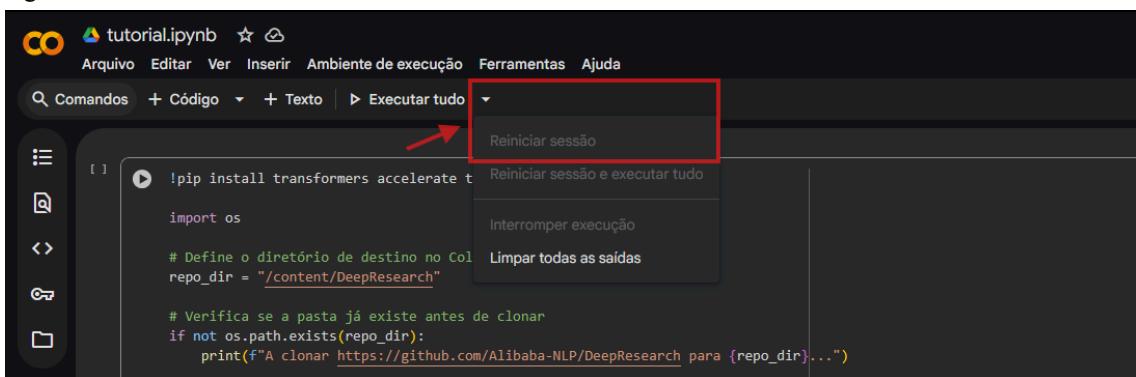
# Carregar o tokenizador
tokenizer = AutoTokenizer.from_pretrained(model_id)

```

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 “Reinic平 sessão”.

Figura 7 - Reinício da sessão



Fonte: Elaboração própria (2025)