

# apresentacao\_ajustada

2025-05-15

## Contents

<b>1</b>	<b>Introdução</b>	<b>3</b>
<b>2</b>	<b>Código Original</b>	<b>3</b>
<b>3</b>	<b>Motivação para Melhoria</b>	<b>5</b>
<b>4</b>	<b>Uso do GitHub Copilot</b>	<b>5</b>
<b>5</b>	<b>Exemplos e Resultados</b>	<b>7</b>
<b>6</b>	<b>Considerações técnicas</b>	<b>9</b>
<b>7</b>	<b>Conclusão</b>	<b>10</b>

UNIVERSIDADE FEDERAL DE SERGIPE  
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TÓPICOS AVANÇADOS EM ENGENHARIA DE SOFTWARE E SISTEMAS DE INFORMAÇÃO I

**Atividade 3**

Alberto Bastos  
Ana Beatriz  
Victor Caetano

Abril 2025

# 1 Introdução

Este relatório descreve a evolução de um código Python para coleta de discursos do Senado Federal via API, demonstrando como foi aprimorado com o uso do GitHub Copilot, uma ferramenta baseada em inteligência artificial.

## 2 Código Original

A primeira versão do código realizava a extração de discursos entre as datas de 01/03/2025 e 01/04/2025, utilizando a API de dados abertos do Senado Federal. Ele recuperava campos como:

- Resumo
- Autor
- Partido
- Tipo de uso da palavra
- Link para o texto completo

O código utilizava as bibliotecas `requests` e `xml.etree.elementTree` para fazer as requisições e parse do XML.

```
import requests
import xml.etree.ElementTree as ET

def buscar_discursos(data_inicio, data_fim):
    url = f"https://legis.senado.leg.br/dadosabertos/plenario/lista/discursos/{data_inicio}/{data_fim}"
    headers = {
        "Accept": "application/xml"
    }
    response = requests.get(url, headers=headers)
    # Exibe o status HTTP e o conteúdo completo da resposta para depuração
    print(f"Status HTTP: {response.status_code}")
    # print(f"Resposta Completa:\n{response.text}") # Descomente se precisar de mais detalhes
    if response.status_code == 200:
        try:
            root = ET.fromstring(response.content)
            # Vamos procurar pelos Pronunciamentos dentro das Sessões
            discursos = []
            for sessao in root.findall('.//Sessoes//Sessao//Pronunciamentos//Pronunciamento'):
                # Acessar os valores de forma segura
                codigo_pronunciamento = sessao.find('CodigoPronunciamento')
                tipo_uso_palavra_codigo = sessao.find('.//TipoUsoPalavra/Codigo')
                tipo_uso_palavra_descricao = sessao.find('.//TipoUsoPalavra/Descricao')
                resumo = sessao.find('Resumo')
                texto_integral = sessao.find('TextoIntegralTxt')
                url_texto_binario = sessao.find('UrlTextoBinario')
                # Acessar dados do autor
                nome_autor = sessao.find('NomeAutor')
                partido = sessao.find('Partido')
                # Adicionando os dados ao dicionário com verificações de 'None'
                discurso_info = {
                    'CodigoPronunciamento': codigo_pronunciamento.text if codigo_pronunciamento is not None else None,
                    'TipoUsoPalavra': {
                        'Codigo': tipo_uso_palavra_codigo.text if tipo_uso_palavra_codigo is not None else None,
                        'Descricao': tipo_uso_palavra_descricao.text if tipo_uso_palavra_descricao is not None else None
                    },
                    'Resumo': resumo.text if resumo is not None else None,
                    'TextoIntegral': texto_integral.text if texto_integral is not None else None,
                    'UrlTextoBinario': url_texto_binario.text if url_texto_binario is not None else None,
                    'NomeAutor': nome_autor.text if nome_autor is not None else None,
                    'Partido': partido.text if partido is not None else None
                }
                discursos.append(discurso_info)
            return discursos
        except Exception as e:
            print(f"Erro ao processar a resposta: {e}")
            return []
```

```

        'Resumo': resumo.text if resumo is not None else 'Não disponível',
        'TextoIntegral': texto_integral.text if texto_integral is not None else 'Não disponível',
        'UrlTextoBinario': url_texto_binario.text if url_texto_binario is not None else 'Não disponível',
        'NomeAutor': nome_autor.text if nome_autor is not None else 'Não disponível',
        'Partido': partido.text if partido is not None else 'Não disponível'
    }
    discursos.append(discurso_info)
    return discursos # Retorna a lista de discursos
except ET.ParseError as e:
    print("Erro ao fazer o parse do XML:", e)
    print("Resposta recebida (parcial):", response.text[:500])
    return None
else:
    print("Falha na requisição!")
    return None

# Exemplo de uso com datas que funcionam
discursos = buscar_discursos("20250301", "20250401")

# Exibindo os discursos de forma legível
if discursos:
    for i, discurso in enumerate(discursos, 1):
        print(f"Discurso {i}:")
        for key, value in discurso.items():
            print(f"{key}: {value}")
        print("="*40)
else:
    print("Não foi possível recuperar os discursos.")

```

Status HTTP: 200

Discurso 1:

CodigoPronunciamento: 512673

TipoUsoPalavra: {'Codigo': '4819', 'Descricao': 'Discurso'}

Resumo: Exposição sobre a importância da regulamentação da reforma tributária, da fiscalização da execução

TextoIntegral: <https://legis.senado.leg.br/dadosabertos/discurso/texto-integral/512673>

UrlTextoBinario: <https://legis.senado.leg.br/dadosabertos/discurso/texto-binario/512673>

NomeAutor: Confúcio Moura

Partido: MDB

=====

Discurso 2:

CodigoPronunciamento: 512672

TipoUsoPalavra: {'Codigo': '4819', 'Descricao': 'Discurso'}

Resumo: Críticas ao STF pelo suposto uso de processos judiciais como instrumento de pressão política. D

TextoIntegral: <https://legis.senado.leg.br/dadosabertos/discurso/texto-integral/512672>

UrlTextoBinario: <https://legis.senado.leg.br/dadosabertos/discurso/texto-binario/512672>

NomeAutor: Eduardo Girão

Partido: NOVO

=====

Discurso 3:

CodigoPronunciamento: 512670

TipoUsoPalavra: {'Codigo': '4819', 'Descricao': 'Discurso'}

Resumo: Defesa da redução da jornada de trabalho sem redução salarial, e registro da tramitação, na CCJ

TextoIntegral: <https://legis.senado.leg.br/dadosabertos/discurso/texto-integral/512670>

UrlTextoBinario: <https://legis.senado.leg.br/dadosabertos/discurso/texto-binario/512670>

NomeAutor: Paulo Paim

Partido: PT

=====

### 3 Motivação para Melhoria

Durante o uso do script, observamos que o campo Resumo trazia uma versão condensada do conteúdo dos discursos. Isso inspirou a ideia de aplicar classificação automática de temas com base nesse resumo, permitindo analisar tendências temáticas sem intervenção manual.

### 4 Uso do GitHub Copilot

Para que fosse possível classificar sem sair do VSCode pedimos ao Agent AI Copilot com o seguinte comando:

com base nos retornos dos discursos do senado, existe o campo resumo em que possui o resumo do discurso, com base nesse resumo voce é capaz de classificar o tema?

O Copilot sugeriu o uso da biblioteca `transformers` com o *pipeline* `zero-shot-classification` e o modelo `facebook/bart-large-mnli`.

Essa sugestão foi implementada para classificar os discursos em temas como: **Política, Economia, Educação, Saúde, Meio Ambiente, Tecnologia e Segurança.**

```
import requests
import xml.etree.ElementTree as ET
from transformers import pipeline

def classificar_tema(resumo):
    classificador = pipeline("zero-shot-classification", model="facebook/bart-large-mnli")
    temas = ["Política", "Economia", "Educação", "Saúde", "Meio Ambiente", "Tecnologia", "Segurança"]
    resultado = classificador(resumo, temas)
    return resultado['labels'][0]

def buscar_discursos(data_inicio, data_fim):
    url = f"https://legis.senado.leg.br/dadosabertos/plenario/lista/discursos/{data_inicio}/{data_fim}"

    headers = {
        "Accept": "application/xml"
    }

    response = requests.get(url, headers=headers)
    print(f"Status HTTP: {response.status_code}")
    if response.status_code == 200:
        try:
            root = ET.fromstring(response.content)
            discursos = []
            for sessao in root.findall('.//Sessoes//Sessao//Pronunciamentos//Pronunciamento'):
                codigo_pronunciamento = sessao.find('CodigoPronunciamento')
                tipo_uso_palavra_codigo = sessao.find('.//TipoUsoPalavra/Codigo')
                tipo_uso_palavra_descricao = sessao.find('.//TipoUsoPalavra/Descricao')
                resumo = sessao.find('Resumo')
                texto_integral = sessao.find('TextoIntegralTxt')
                url_texto_binario = sessao.find('UrlTextoBinario')
                nome_autor = sessao.find('NomeAutor')
                partido = sessao.find('Partido')
                discurso_info = {
```

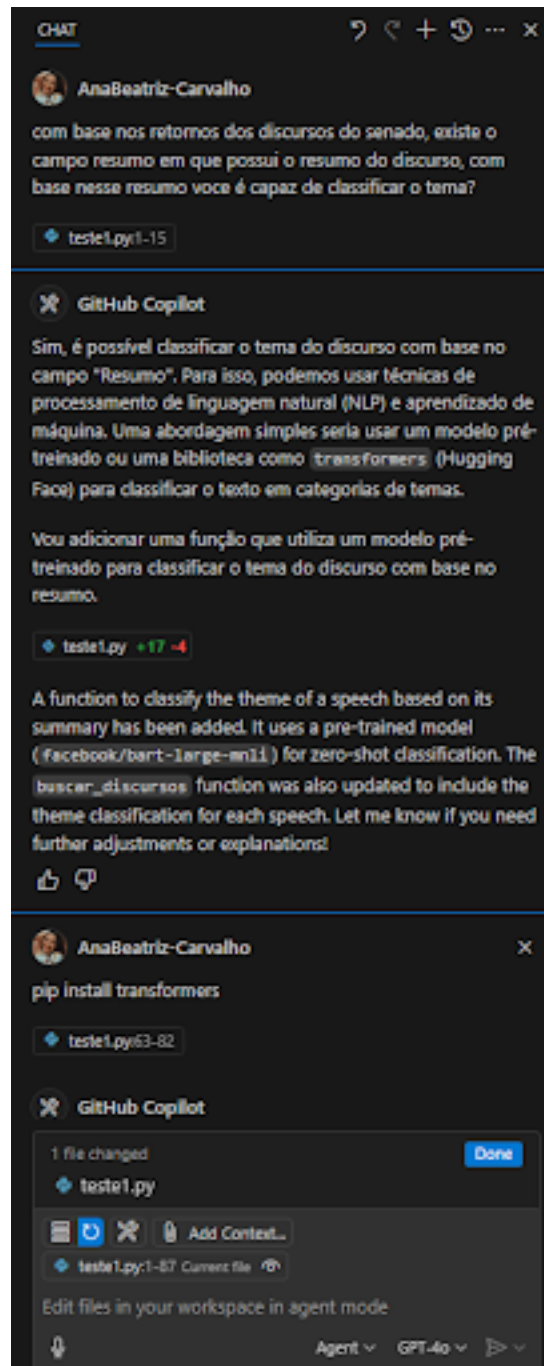


Figure 1: Agent AI Copilot

```

        'CodigoPronunciamento': codigo_pronunciamento.text if codigo_pronunciamento is not None else None,
        'TipoUsoPalavra': {
            'Codigo': tipo_uso_palavra_codigo.text if tipo_uso_palavra_codigo is not None else None,
            'Descricao': tipo_uso_palavra_descricao.text if tipo_uso_palavra_descricao is not None else None
        },
        'Resumo': resumo.text if resumo is not None else 'Não disponível',
        'TextoIntegral': texto_integral.text if texto_integral is not None else 'Não disponível',
        'UrlTextoBinario': url_texto_binario.text if url_texto_binario is not None else 'Não disponível',
        'NomeAutor': nome_autor.text if nome_autor is not None else 'Não disponível',
        'Partido': partido.text if partido is not None else 'Não disponível',
        # Classifica o tema com base no resumo
        'Tema': classificar_tema(resumo.text) if resumo is not None else 'Não disponível'
    }

    discursos.append(discurso_info)
    return discursos # Retorna a lista de discursos
except ET.ParseError as e:
    print("Erro ao fazer o parse do XML:", e)
    print("Resposta recebida (parcial):", response.text[:500])
    return None
else:
    print("Falha na requisição!")
    return None

# Exemplo de uso com datas que funcionam
discursos = buscar_discursos("20250301", "20250401")

# Exibindo os discursos de forma legível
if discursos:
    for i, discurso in enumerate(discursos, 1):
        print(f"Discurso {i}:")
        for key, value in discurso.items():
            print(f"{key}: {value}")
        print("="*40)
else:
    print("Não foi possível recuperar os discursos.")

```

Status HTTP: 200

config.json: 0%| | 0.00/1.15k [00:00<?, ?B/s]

model.safetensors: 0%| | 0.00/1.63G [00:00<?, ?B/s]

## 5 Exemplos e Resultados

Após a implementação, ao executar o código, os discursos passaram a ser retornados com seus respectivos temas.

Alguns exemplos da execução:

```

CodigoPronunciamento: 512673
TipoUsoPalavra: {'Codigo': '4819', 'Descricao': 'Discurso'}
Resumo: Exposicao sobre a importancia da regulamentacao da reforma tributaria, da
fiscalizacao da execucao de politicas publicas, a exemplo da Lei Geral de Saneamento,
e da regularizacao de terras no Brasil. Criticas ao excesso de burocracia da
Administracao Publica e a polarizacao politica que, segundo S. Exa., compromete o
avanco institucional.
TextoIntegral: https://legis.senado.leg.br/dadosabertos/discurso/texto-
integral/512673
UrlTextoBinario: https://legis.senado.leg.br/dadosabertos/discurso/texto-
binario/512673
NomeAutor: Confucio Moura
Partido: MDB
Tema: Politica
=====

Discurso 2:
CodigoPronunciamento: 512672
TipoUsoPalavra: {'Codigo': '4819', 'Descricao': 'Discurso'}
Resumo: Criticas ao STF pelo suposto uso de processos judiciais como instrumento de
pressao politica. Defesa da anistia aos presos pelos atos de 8 de janeiro de 2023.
Prestacao de contas e registro da doacao do salario de S. Exa. a Associacao dos
Familiares e Vitimas do 08 de Janeiro (Asfav).
TextoIntegral: https://legis.senado.leg.br/dadosabertos/discurso/texto-
integral/512672
UrlTextoBinario: https://legis.senado.leg.br/dadosabertos/discurso/texto-
binario/512672
NomeAutor: Eduardo Girao
Partido: NOVO
Tema: Politica
=====

```

Figure 2: Resposta API Senado



## 6 Considerações técnicas

Durante a execução inicial do modelo, foi exibida a mensagem `Device set to use cpu`, indicando que o modelo estava sendo carregado. Por ser um processo que estava demorando muito resolvemos questionar ao modelo de Inteligência Artificial GPT se era normal considerando nosso *script* o tempo que estava levando o processo. A resposta do GPT

Isso significa que agora o TensorFlow foi instalado corretamente e o seu código está conseguindo carregar o modelo facebook/bart-large-mnli para fazer a classificação.

Esse trecho aqui é só log do TensorFlow e da Hugging Face dizendo que tudo está sendo executado na CPU (o que é normal, já que você provavelmente não está usando uma GPU):

Mas por que está demorando? O modelo facebook/bart-large-mnli é grande e, na primeira vez, ele precisa ser:

Baixado da Hugging Face Hub (~1.6 GB). Carregado para a memória. Rodado para cada discurso (o que pode ser lento se houver muitos).

O que você pode fazer agora: Esperar terminar o processo Se seu código está ainda rodando e você não recebeu erro, deixe ele completar. Pode demorar alguns minutos, especialmente se forem muitos discursos ou se sua internet estiver lenta para baixar o modelo.

Dica de melhoria: carregar o modelo uma vez só! Como falei antes, você está carregando o modelo dentro da função `classificar_tema` a cada chamada. Isso torna tudo muito mais lento.”

O código sugerido pelo GPT ficou:

```
import requests
import xml.etree.ElementTree as ET
from transformers import pipeline

# Carrega o modelo UMA vez só
classificador = pipeline("zero-shot-classification", model="facebook/bart-large-mnli")

def classificar_tema(resumo):
    # Define os possíveis temas para classificação
    temas = ["Política", "Economia", "Educação", "Saúde", "Meio Ambiente", "Tecnologia", "Segurança"]
    try:
        # Realiza a classificação
        resultado = classificador(resumo, temas)
        return resultado['labels'][0]
    except Exception as e:
        print(f"Erro ao classificar tema: {e}")
        return 'Erro na classificação'

def buscar_discursos(data_inicio, data_fim):
    url = f"https://legis.senado.leg.br/dadosabertos/plenario/lista/discursos/{data_inicio}/{data_fim}"
    headers = {
        "Accept": "application/xml"
    }
    response = requests.get(url, headers=headers)
    print(f"Status HTTP: {response.status_code}")
    if response.status_code == 200:
        try:
            root = ET.fromstring(response.content)
            discursos = []
            for sessao in root.findall('.//Sessoes//Sessao//Pronunciamentos//Pronunciamento'):
                # Processar cada discurso aqui
```

```

        codigo_pronunciamento = sessao.find('CodigoPronunciamento')
        tipo_uso_palavra_codigo = sessao.find('..//TipoUsoPalavra/Codigo')
        tipo_uso_palavra_descricao = sessao.find('..//TipoUsoPalavra/Descricao')
        resumo = sessao.find('Resumo')
        texto_integral = sessao.find('TextoIntegralTxt')
        url_texto_binario = sessao.find('UrlTextoBinario')
        nome_autor = sessao.find('NomeAutor')
        partido = sessao.find('Partido')

        discurso_info = {
            'CodigoPronunciamento': codigo_pronunciamento.text if codigo_pronunciamento is not None else None,
            'TipoUsoPalavra': {
                'Codigo': tipo_uso_palavra_codigo.text if tipo_uso_palavra_codigo is not None else None,
                'Descricao': tipo_uso_palavra_descricao.text if tipo_uso_palavra_descricao is not None else None
            },
            'Resumo': resumo.text if resumo is not None else 'Não disponível',
            'TextoIntegral': texto_integral.text if texto_integral is not None else 'Não disponível',
            'UrlTextoBinario': url_texto_binario.text if url_texto_binario is not None else 'Não disponível',
            'NomeAutor': nome_autor.text if nome_autor is not None else 'Não disponível',
            'Partido': partido.text if partido is not None else 'Não disponível',
            'Tema': classificar_tema(resumo.text) if resumo is not None else 'Não disponível'
        }

        discursos.append(discurso_info)
    return discursos
except ET.ParseError as e:
    print("Erro ao fazer o parse do XML:", e)
    print("Resposta recebida (parcial):", response.text[:500])
    return None
else:
    print("Falha na requisição!")
    return None

# Teste
discursos = buscar_discursos("20250301", "20250401")

if discursos:
    for i, discurso in enumerate(discursos, 1):
        print(f"Discurso {i}:")
        for key, value in discurso.items():
            print(f"{key}: {value}")
        print("="*40)
else:
    print("Não foi possível recuperar os discursos.")

```

Enquanto o código ajustado pelo Agent AI Copilot levou 508.862 segundos para trazer os dados e classificá-los, o código foi ajustado pelo GPT 351.253 segundos, indicando um ganho de performance.

## 7 Conclusão

A implementação da IA neste projeto mostrou-se eficaz tanto na aplicação da classificação temática quanto na otimização da performance do código.

A contribuição do Github Copilot foi importante ao indicar o uso de um modelo NLP para resolver o problema. No entanto, a sugestão inicial, apesar de funcional, não foi otimização em termos de desempenho.

Por outro lado, a interação com o ChatGPT mostrou melhorias na estrutura do código e explicação do motivo do gargalo. A diferença entre as abordagens mostra como o Copilot atua como um assistente rápido para quem está no código e o ChatGPT como um consultor estratégico e explicativo para tratar de assuntos mais técnicos.

<https://github.com/AnaBeatriz-Carvalho/AgentAI>



Figure 3: QR Code Github