taes_2

June 5, 2025

[143]:
```python
# Instalar dependências se necessário
# !pip install nest_asyncio requests google-adk

import nest_asyncio
import asyncio
import requests
import xml.etree.ElementTree as ET
import json
import os
from datetime import datetime, date, timedelta
from typing import List, Dict, Any

nest_asyncio.apply()

# Configuração da API do Google AI
import google.generativeai as genai
from google.adk.agents.sequential_agent import SequentialAgent
from google.adk.agents import Agent
from google.adk.runners import Runner
from google.adk.sessions import InMemorySessionService
from google.genai import types

class SenadorDiscursoProcessor:
    def __init__(self, api_key: str = None):
        """
        Inicializa o processador de discursos do Senado Federal

        Args:
            api_key: Chave da API do Google AI. Se não fornecida, tentará usar␣
 ↪variável de ambiente GOOGLE_API_KEY
        """
        self.configure_api(api_key)
        self.setup_agents()

    def configure_api(self, api_key: str = None):
        """Configura a API do Google AI"""
        if api_key:
```

```
            self.api_key = api_key
        else:
            # Tentar obter da variável de ambiente
            self.api_key = os.getenv('GOOGLE_API_KEY')

        if not self.api_key:
            raise ValueError("""
              CHAVE DA API NÃO CONFIGURADA!

            Configure a chave da API do Google AI de uma das formas:

            1. Passar como parâmetro:
               processor = SenadorDiscursoProcessor(api_key="sua_chave_aqui")

            2. Definir variável de ambiente:
               os.environ['GOOGLE_API_KEY'] = "sua_chave_aqui"

            3. No Colab, usar:
               from google.colab import userdata
               api_key = userdata.get('GOOGLE_API_KEY')
               processor = SenadorDiscursoProcessor(api_key=api_key)

            Para obter sua chave: https://makersuite.google.com/app/apikey
            """)

        # Configurar as variáveis de ambiente para que o ADK possa acessar
        os.environ['GOOGLE_API_KEY'] = self.api_key

        # Também configurar via genai para compatibilidade
        genai.configure(api_key=self.api_key)
        print(f"  API do Google AI configurada com sucesso!")
        print(f"  Chave: {self.api_key[:8]}...{self.api_key[-4:]}")
        print(f"  Variável de ambiente GOOGLE_API_KEY definida para ADK")

    def setup_agents(self):
        """Configura os agentes individuais e o agente sequencial"""

        # Agente Classificador
        self.classificador = Agent(
            name="classificador_tematico",
            model="gemini-1.5-flash",
            instruction="""Você é um especialista em análise de discursos␣
 ↪legislativos brasileiros com profundo conhecimento político e social.

OBJETIVO: Faça uma análise detalhada e abrangente do discurso parlamentar␣
 ↪recebido em formato JSON.
```

```
ESTRUTURA OBRIGATÓRIA DA ANÁLISE:

##  CLASSIFICAÇÃO TEMÁTICA
- **Tema Principal:** [Ex: Economia, Saúde, Educação, Segurança, Meio Ambiente,␣
 ↪Política Externa, etc.]
- **Subtemas Específicos:** [Liste todos os subtemas abordados]
- **Área de Política Pública:** [Qual setor governamental é afetado]

##  POSICIONAMENTO POLÍTICO
- **Posição do Parlamentar:** [Favorável/Contrário/Neutro/Propositivo - com␣
 ↪explicação detalhada]
- **Linha Ideológica:** [Liberal, conservador, progressista, etc.]
- **Alinhamento Partidário:** [Se o discurso reflete linha oficial do partido]

##  TIPO DE PRONUNCIAMENTO
- **Categoria:** [Crítica, Proposta Legislativa, Elogio, Denúncia, Prestação de␣
 ↪Contas, etc.]
- **Tom do Discurso:** [Técnico, emotivo, confrontativo, conciliador, etc.]
- **Estratégia Retórica:** [Como o parlamentar construiu sua argumentação]

##  RELEVÂNCIA E IMPACTO
- **Relevância Política:** [Alta/Média/Baixa - com justificativa]
- **Público-Alvo:** [Quem o parlamentar está tentando atingir]
- **Potencial de Repercussão:** [Nacional, regional, local, setorial]

##  CONTEÚDO SUBSTANTIVO
- **Principais Argumentos:** [Liste e detalhe cada argumento principal]
- **Dados e Evidências:** [Cite números, estatísticas ou fatos mencionados]
- **Propostas Concretas:** [Soluções ou ações propostas pelo parlamentar]
- **Críticas Específicas:** [A quem ou o que está criticando e por quê]

##  CONTEXTO POLÍTICO
- **Cenário Atual:** [Relate o contexto político/social do momento]
- **Conexão com Agenda Nacional:** [Como se relaciona com temas em discussão no␣
 ↪país]
- **Impacto no Debate Público:** [Qual contribuição para o debate democrático]

Seja extremamente detalhado e analítico. Use linguagem clara mas tecnicamente␣
 ↪precisa."""
        )

        # Agente Resumidor
        self.resumidor = Agent(
            name="resumidor_executivo",
            model="gemini-1.5-flash",
            instruction="""Você é um especialista em síntese de conteúdo␣
 ↪político e legislativo. Crie resumos executivos abrangentes e informativos.
```

```
OBJETIVO: Com base na classificação detalhada recebida, produza um RESUMO␣
 ↪EXECUTIVO COMPLETO e INFORMATIVO.

ESTRUTURA OBRIGATÓRIA DO RESUMO:

#  RESUMO EXECUTIVO

##  SÍNTESE DO PRONUNCIAMENTO
[Parágrafo de 3-4 linhas explicando o que foi dito, por quem, e o contexto␣
 ↪geral]

##  PONTOS PRINCIPAIS ABORDADOS
[Liste em tópicos detalhados os 4-6 pontos principais do discurso, explicando␣
 ↪cada um]

### Argumentação Central
- **Tese Principal:** [Qual a ideia central defendida]
- **Justificativas:** [Como o parlamentar fundamentou sua posição]
- **Evidências Apresentadas:** [Dados, exemplos, casos citados]

### Propostas e Soluções
- **Medidas Propostas:** [Ações concretas sugeridas]
- **Instrumentos Sugeridos:** [Leis, políticas, programas mencionados]
- **Cronograma ou Urgência:** [Prazos ou urgência expressa]

##  POSICIONAMENTO POLÍTICO
- **Linha Adotada:** [Posição política clara do parlamentar]
- **Alvos de Crítica:** [Quem ou o que foi criticado]
- **Elogios e Reconhecimentos:** [O que foi elogiado ou defendido]

##  IMPACTOS E CONSEQUÊNCIAS MENCIONADAS
- **Efeitos Esperados:** [Resultados que o parlamentar prevê]
- **Riscos Apontados:** [Perigos ou problemas identificados]
- **Benefícios Destacados:** [Vantagens ou melhorias prometidas]

##  CONTEXTO POLÍTICO E SOCIAL
- **Cenário de Fundo:** [Situação política/social que motivou o discurso]
- **Agenda Legislativa:** [Relação com projetos ou debates em curso]
- **Repercussão Esperada:** [Possível impacto no debate público]

##  RELEVÂNCIA ESTRATÉGICA
- **Importância do Tema:** [Por que este assunto é relevante agora]
- **Stakeholders Afetados:** [Quem será impactado pelas propostas/críticas]
- **Conexões com Políticas Nacionais:** [Como se relaciona com agenda federal]
```

```
IMPORTANTE: Seja detalhado, informativo e contextualizado. O resumo deve␣
 ↪permitir compreender completamente o discurso sem precisar ler o original."""
        )

        # Agente Revisor
        self.revisor = Agent(
            name="revisor_qualidade",
            model="gemini-1.5-flash",
            instruction="""Você é um editor especializado em conteúdo político␣
 ↪e legislativo com foco em produzir análises de alta qualidade e␣
 ↪compreensibilidade.

OBJETIVO: Revisar e APRIMORAR o resumo executivo para máxima clareza,␣
 ↪completude e utilidade.

CRITÉRIOS DE REVISÃO:

##  COMPLETUDE DA INFORMAÇÃO
- Verificar se todos os aspectos importantes do discurso foram cobertos
- Adicionar detalhes relevantes que possam ter sido omitidos
- Garantir que o contexto político está bem explicado
- Confirmar que as propostas estão claramente descritas

##  CLAREZA E ACESSIBILIDADE
- Simplificar linguagem técnica excessiva sem perder precisão
- Explicar siglas, termos técnicos e referências políticas
- Organizar informações de forma lógica e fluida
- Garantir que um leitor não especializado possa compreender

##  PRECISÃO E OBJETIVIDADE
- Verificar se o tom é imparcial e jornalístico
- Corrigir possíveis interpretações tendenciosas
- Balancear críticas e elogios conforme o discurso original
- Manter fidelidade ao conteúdo original

##  ESTRUTURA E FORMATAÇÃO
- Melhorar a organização das seções
- Adicionar emojis e formatação para facilitar leitura
- Criar transições suaves entre seções
- Garantir hierarquia clara de informações

##  CONTEXTUALIZAÇÃO POLÍTICA
- Enriquecer o contexto histórico e político quando necessário
- Explicar implicações que podem não estar óbvias
- Conectar o discurso com tendências políticas atuais
- Destacar a relevância para o cenário nacional
```

```
##   CHAMADAS DE ATENÇÃO
Adicione seções especiais quando relevante:
- **  ALERTA:** Para questões urgentes ou polêmicas
- **  DESTAQUE:** Para propostas inovadoras ou importantes
- **  CONEXÕES:** Para links com outros temas políticos atuais

RESULTADO ESPERADO: Um texto que seja simultaneamente informativo, acessível,␣
 ↪completo e envolvente, permitindo compreensão total do discurso e seu␣
 ↪significado político."""
        )

        # Agente Redator de Relatório Narrativo
        self.redator_narrativo = Agent(
            name="redator_relatorio_narrativo",
            model="gemini-1.5-flash",
            instruction="""Você é um jornalista especializado em política␣
 ↪brasileira e comunicação institucional.

OBJETIVO: Transformar a análise técnica estruturada em um RELATÓRIO NARRATIVO␣
 ↪FLUÍDO E INFORMATIVO em texto corrido.

ESTRUTURA DO RELATÓRIO NARRATIVO:

Escreva um texto jornalístico de 3-5 parágrafos que conte a "história" do␣
 ↪discurso de forma envolvente e informativa. O texto deve fluir naturalmente,␣
 ↪como uma matéria jornalística bem escrita.

## PRIMEIRO PARÁGRAFO - LEAD JORNALÍSTICO
Comece com um parágrafo que responda: Quem falou? Sobre o quê? Quando? Por quê?␣
 ↪Qual foi a mensagem central? Use linguagem cativante que desperte interesse.

## PARÁGRAFOS INTERMEDIÁRIOS - DESENVOLVIMENTO
- Detalhe os principais argumentos e propostas apresentadas
- Contextualize politicamente o pronunciamento
- Explique as implicações e consequências mencionadas
- Relacione com o cenário político atual
- Cite dados, números ou exemplos específicos mencionados

## PARÁGRAFO FINAL - CONCLUSÃO E PERSPECTIVAS
Conclua com reflexões sobre a importância do discurso, possível repercussão, e␣
 ↪o que isso significa para o debate político nacional.

ESTILO DE ESCRITA:
- Use linguagem clara, objetiva mas envolvente
- Evite jargões excessivos (explique quando necessário)
- Mantenha imparcialidade jornalística
```

```
- Faça transições suaves entre ideias
- Use voz ativa e frases variadas
- Crie um texto que seja interessante de ler

IMPORTANTE:
- NÃO use formatação markdown (sem #, **, etc.)
- Escreva em texto corrido, como uma matéria de jornal
- Mantenha todos os fatos e detalhes importantes da análise original
- O texto deve ser informativo mas acessível ao público geral"""
        )

        # Agente Sequencial (Orquestrador)
        self.pipeline_sequencial = SequentialAgent(
            name="pipeline_analise_discursos",
            sub_agents=[self.classificador, self.resumidor, self.revisor, self.
 ↪redator_narrativo]
        )

    def buscar_discursos_senado(self, data_inicio: str, data_fim: str) ->␣
 ↪List[Dict[str, Any]]:
        """
        Busca discursos no web service do Senado Federal

        Args:
            data_inicio: Data início no formato 'AAAAMMDD'
            data_fim: Data fim no formato 'AAAAMMDD'

        Returns:
            Lista de dicionários com dados dos discursos
        """
        url = f"https://legis.senado.leg.br/dadosabertos/plenario/lista/
 ↪discursos/{data_inicio}/{data_fim}"

        try:
            response = requests.get(url, timeout=30)
            response.raise_for_status()

            # Parse do XML
            root = ET.fromstring(response.content)
            discursos = []

            # Extrair dados conforme estrutura do XML do Senado
            for pronunciamento in root.findall('.//Pronunciamento'):
                discurso = {
                    'CodigoPronunciamento': self._get_text(pronunciamento,␣
 ↪'CodigoPronunciamento'),
```

```
                    'TipoUsoPalavra': self._get_text(pronunciamento,␣
↪'TipoUsoPalavra'),
                    'Codigo': self._get_text(pronunciamento, 'Codigo'),
                    'Descricao': self._get_text(pronunciamento, 'Descricao'),
                    'Resumo': self._get_text(pronunciamento, 'Resumo'),
                    'TextoIntegral': self._get_text(pronunciamento,␣
↪'TextoIntegral'),
                    'UrlTextoBinario': self._get_text(pronunciamento,␣
↪'UrlTextoBinario'),
                    'NomeAutor': self._get_text(pronunciamento, 'NomeAutor'),
                    'Partido': self._get_text(pronunciamento, 'Partido')
                }
                discursos.append(discurso)

            print(f"  Coletados {len(discursos)} discursos do período␣
↪{data_inicio} a {data_fim}")
            return discursos

        except requests.RequestException as e:
            print(f"  Erro ao buscar dados do Senado: {e}")
            return []
        except ET.ParseError as e:
            print(f"  Erro ao processar XML: {e}")
            return []

    def _get_text(self, element, tag_name: str) -> str:
        """Extrai texto de elemento XML de forma segura"""
        child = element.find(tag_name)
        return child.text if child is not None and child.text else ""

    async def processar_discurso_individual(self, discurso: Dict[str, Any]) ->␣
↪Dict[str, Any]:
        """
        Processa um discurso individual através do pipeline sequencial

        Args:
            discurso: Dicionário com dados do discurso

        Returns:
            Dicionário com resultado da análise
        """
        try:
            # Converter dicionário para string JSON (entrada do classificador)
            discurso_json = json.dumps(discurso, ensure_ascii=False, indent=2)

            # Configurar sessão
            session_service = InMemorySessionService()
```

```
            session = await session_service.create_session(
                app_name="pipeline_analise_discursos",
                user_id="senado_analyzer",
                session_id=f"session_{discurso.get('CodigoPronunciamento',␣
↪'unknown')}"
            )

            # Executar pipeline sequencial
            runner = Runner(
                agent=self.pipeline_sequencial,
                app_name="pipeline_analise_discursos",
                session_service=session_service
            )

            content = types.Content(
                role="user",
                parts=[types.Part(text=discurso_json)]
            )

            # Usar runner.run() síncrono ao invés de run_async()
            events = runner.run(
                user_id="senado_analyzer",
                session_id=f"session_{discurso.get('CodigoPronunciamento',␣
↪'unknown')}",
                new_message=content
            )

            # Extrair resposta final do generator síncrono
            resultado_final = ""
            for event in events:
                # Verificar se é uma resposta final do agente
                if hasattr(event, 'final_response') and event.final_response:
                    if hasattr(event, 'content') and event.content:
                        for part in event.content.parts:
                            if hasattr(part, 'text') and part.text:
                                resultado_final += part.text + "\n"
                # Alternativa: verificar método is_final_response()
                elif hasattr(event, 'is_final_response') and event.
↪is_final_response():
                    if hasattr(event, 'content') and event.content and event.
↪content.parts:
                        for part in event.content.parts:
                            if hasattr(part, 'text') and part.text:
                                resultado_final += part.text + "\n"

            return {
                "CodigoPronunciamento": discurso.get("CodigoPronunciamento"),
```

```
                "NomeAutor": discurso.get("NomeAutor"),
                "Partido": discurso.get("Partido"),
                "DataProcessamento": datetime.now().isoformat(),
                "ModeloUsado": "gemini-1.5-flash",
                "VersaoScript": "v2.0 - Pipeline Sequencial com 4 Agentes",
                "AgentesUsados": "Classificador → Resumidor → Revisor → Redator␣
↪Narrativo",
                "AnaliseCompleta": resultado_final.strip(),
                "Status": "sucesso"
            }

        except Exception as e:
            return {
                "CodigoPronunciamento": discurso.get("CodigoPronunciamento"),
                "NomeAutor": discurso.get("NomeAutor"),
                "Partido": discurso.get("Partido"),
                "DataProcessamento": datetime.now().isoformat(),
                "ModeloUsado": "gemini-1.5-flash",
                "VersaoScript": "v2.0 - Pipeline Sequencial com 4 Agentes",
                "AgentesUsados": "Erro no processamento",
                "AnaliseCompleta": "",
                "Status": "erro",
                "ErroDetalhes": str(e)
            }

    async def processar_periodo_completo(self, data_inicio: str, data_fim: str)␣
↪-> List[Dict[str, Any]]:
        """
        Processa todos os discursos de um período

        Args:
            data_inicio: Data início no formato 'AAAAMMDD'
            data_fim: Data fim no formato 'AAAAMMDD'

        Returns:
            Lista com resultados de todas as análises
        """
        print(f"  Iniciando coleta de discursos de {data_inicio} a {data_fim}")

        # 1. Coleta via request
        discursos = self.buscar_discursos_senado(data_inicio, data_fim)

        if not discursos:
            print("  Nenhum discurso encontrado no período")
            return []

        # 2. Processamento através do pipeline sequencial
```

```
        print(f"  Iniciando análise de {len(discursos)} discursos...")
        resultados = []

        for i, discurso in enumerate(discursos, 1):
            print(f"  Processando discurso {i}/{len(discursos)} - {discurso.
↪get('NomeAutor', 'N/A')}")

            resultado = await self.processar_discurso_individual(discurso)
            resultados.append(resultado)

            if resultado["Status"] == "sucesso":
                print(f"  Discurso {i} processado com sucesso")
                # Exibir preview do resultado
                preview = resultado["AnaliseCompleta"][:200] + "..." if␣
↪len(resultado["AnaliseCompleta"]) > 200 else resultado["AnaliseCompleta"]
                print(f"  Preview: {preview}")
                print("-" * 50)
            else:
                print(f"  Erro no discurso {i}: {resultado.get('ErroDetalhes',␣
↪'Erro desconhecido')}")

        return resultados

    def gerar_relatorio_final(self, resultados: List[Dict[str, Any]], periodo:␣
↪str) -> str:
        """Gera relatório consolidado dos resultados"""
        total = len(resultados)
        sucessos = len([r for r in resultados if r["Status"] == "sucesso"])
        erros = total - sucessos

        # Análise de temas para o período
        temas_encontrados = []
        partidos_ativos = []
        parlamentares_ativos = []

        for resultado in resultados:
            if resultado["Status"] == "sucesso":
                partidos_ativos.append(resultado['Partido'])
                parlamentares_ativos.append(resultado['NomeAutor'])
                # Extrair temas das análises (busca por padrões)
                analise = resultado['AnaliseCompleta'].lower()
                if 'economia' in analise:
                    temas_encontrados.append('Economia')
                if 'saúde' in analise:
                    temas_encontrados.append('Saúde')
                if 'educação' in analise:
                    temas_encontrados.append('Educação')
```

```
                if 'segurança' in analise:
                    temas_encontrados.append('Segurança')
                if 'meio ambiente' in analise:
                    temas_encontrados.append('Meio Ambiente')

        # Contar frequências
        from collections import Counter
        temas_freq = Counter(temas_encontrados)
        partidos_freq = Counter(partidos_ativos)
        parlamentares_freq = Counter(parlamentares_ativos)

        relatorio = f"""#  Relatório de Análise de Discursos do Senado Federal

##  Período Analisado: {periodo}

##  VISÃO GERAL EXECUTIVA

### Estatísticas de Processamento
- **Total de discursos analisados:** {total}
- **Análises bem-sucedidas:** {sucessos}
- **Erros encontrados:** {erros}
- **Taxa de sucesso:** {(sucessos/total*100):.1f}%

###  Principais Temas Debatidos no Período
"""

        if temas_freq:
            for tema, freq in temas_freq.most_common(5):
                relatorio += f"- **{tema}:** {freq} pronunciamentos\n"
        else:
            relatorio += "- Análise temática detalhada disponível nos discursos␣
 ↪individuais\n"

        relatorio += f"""
###  Atividade Parlamentar por Partido
"""
        for partido, freq in partidos_freq.most_common(10):
            relatorio += f"- **{partido}:** {freq} pronunciamentos\n"

        relatorio += f"""
###  Parlamentares Mais Ativos
"""
        for parlamentar, freq in parlamentares_freq.most_common(10):
            relatorio += f"- **{parlamentar}:** {freq} pronunciamentos\n"

        relatorio += f"""
```

```
---

##   ANÁLISES DETALHADAS DOS DISCURSOS

"""

        for i, resultado in enumerate(resultados, 1):
            if resultado["Status"] == "sucesso":
                relatorio += f"""
###   Discurso {i} - {resultado['NomeAutor']} ({resultado['Partido']})
**Código do Pronunciamento:** {resultado['CodigoPronunciamento']}
**Data de Processamento:** {resultado['DataProcessamento'][:10]}

{resultado['AnaliseCompleta']}

---
"""
            else:
                relatorio += f"""
###   Discurso {i} - ERRO NO PROCESSAMENTO
**Autor:** {resultado['NomeAutor']} ({resultado['Partido']})
**Código:** {resultado['CodigoPronunciamento']}
**Erro:** {resultado.get('ErroDetalhes', 'Erro desconhecido')}

---
"""

        return relatorio

    async def gerar_relatorio_narrativo_periodo(self, resultados:␣
 ↪List[Dict[str, Any]], periodo: str) -> str:
        """Gera um relatório narrativo consolidado do período usando LLM"""

        # Preparar síntese dos dados para o LLM
        dados_para_sintese = {
            "periodo": periodo,
            "total_discursos": len(resultados),
            "sucessos": len([r for r in resultados if r["Status"] ==␣
 ↪"sucesso"]),
            "modelo_usado": "gemini-1.5-flash",
            "pipeline": "Classificador → Resumidor → Revisor → Redator␣
 ↪Narrativo",
            "discursos_analisados": []
        }

        # Extrair resumos dos discursos bem-sucedidos
        for resultado in resultados:
```

```
            if resultado["Status"] == "sucesso":
                dados_para_sintese["discursos_analisados"].append({
                    "autor": resultado["NomeAutor"],
                    "partido": resultado["Partido"],
                    "codigo": resultado["CodigoPronunciamento"],
                    "analise": resultado["AnaliseCompleta"][:1000] + "..." if␣
 ↪len(resultado["AnaliseCompleta"]) > 1000 else resultado["AnaliseCompleta"]
                })

        # Configurar agente sintético para relatório narrativo
        agente_sintese = Agent(
            name="sintetizador_periodo",
            model="gemini-1.5-flash",
            instruction=f"""Você é um analista político especializado em␣
 ↪produzir relatórios executivos sobre atividade parlamentar.

TAREFA: Criar um RELATÓRIO NARRATIVO CONSOLIDADO sobre a atividade no Senado␣
 ↪Federal no período {periodo}.

DADOS RECEBIDOS: Você receberá análises individuais de␣
 ↪{dados_para_sintese['total_discursos']} discursos parlamentares processados␣
 ↪por IA.

ESTRUTURA DO RELATÓRIO (EM TEXTO CORRIDO):

**INTRODUÇÃO** (1 parágrafo)
Apresente o período analisado, quantos discursos foram examinados, e uma visão␣
 ↪geral do cenário político do momento.

**PRINCIPAIS TEMAS DEBATIDOS** (2-3 parágrafos)
Identifique e descreva os temas dominantes nos pronunciamentos. Agrupe␣
 ↪discursos similares e explique as principais preocupações dos parlamentares.

**POSICIONAMENTOS POLÍTICOS OBSERVADOS** (2 parágrafos)
Analise as tendências ideológicas, críticas recorrentes, propostas similares, e␣
 ↪o tom geral dos debates.

**ATIVIDADE PARLAMENTAR** (1 parágrafo)
Comente sobre quais partidos e parlamentares se destacaram, e que tipo de␣
 ↪pronunciamentos predominaram.

**RELEVÂNCIA PARA O CENÁRIO NACIONAL** (2 parágrafos)
Contextualize os discursos dentro do panorama político brasileiro atual.␣
 ↪Explique como os temas se conectam com questões nacionais relevantes.

**CONCLUSÃO** (1 parágrafo)
```

```
Síntese sobre o que este período revela sobre as prioridades do Senado e␣
 ↪possíveis tendências futuras.

ESTILO:
- Texto jornalístico fluído, sem formatação markdown
- Linguagem clara e acessível
- Análise imparcial mas perspicaz
- Conecte os pontos entre discursos diferentes
- Identifique padrões e tendências

INFORMAÇÕES TÉCNICAS A INCLUIR:
- Modelo de IA usado: {dados_para_sintese['modelo_usado']}
- Pipeline de processamento: {dados_para_sintese['pipeline']}
- Taxa de sucesso no processamento
- Período específico analisado

O relatório deve ser informativo, bem escrito e útil para compreender a␣
 ↪atividade parlamentar do período."""
        )

        # Preparar dados como JSON para o agente
        dados_json = json.dumps(dados_para_sintese, ensure_ascii=False,␣
 ↪indent=2)

        try:
            # Configurar sessão para síntese
            session_service = InMemorySessionService()
            session = await session_service.create_session(
                app_name="sintese_periodo",
                user_id="analista_periodo",
                session_id=f"sintese_{periodo}"
            )

            runner = Runner(
                agent=agente_sintese,
                app_name="sintese_periodo",
                session_service=session_service
            )

            content = types.Content(
                role="user",
                parts=[types.Part(text=dados_json)]
            )

            # Executar síntese
            events = runner.run(
                user_id="analista_periodo",
```

```
                session_id=f"sintese_{periodo}",
                new_message=content
            )

            resultado_sintese = ""
            for event in events:
                if hasattr(event, 'is_final_response') and event.
 ↪is_final_response():
                    if hasattr(event, 'content') and event.content and event.
 ↪content.parts:
                        for part in event.content.parts:
                            if hasattr(part, 'text') and part.text:
                                resultado_sintese += part.text + "\n"

            return f"""#  RELATÓRIO NARRATIVO - ATIVIDADE SENATORIAL

**Período:** {periodo}
**Processamento realizado em:** {datetime.now().strftime('%d/%m/%Y às %H:%M')}
**Modelo de IA utilizado:** {dados_para_sintese['modelo_usado']}
**Pipeline de análise:** {dados_para_sintese['pipeline']}
**Discursos processados:** {dados_para_sintese['total_discursos']} (Taxa de␣
 ↪sucesso: {(dados_para_sintese['sucessos']/
 ↪dados_para_sintese['total_discursos']*100):.1f}%)

---

{resultado_sintese.strip()}

---

*Este relatório foi gerado automaticamente através de análise por Inteligência␣
 ↪Artificial utilizando o modelo {dados_para_sintese['modelo_usado']} do␣
 ↪Google. O processamento envolveu quatro etapas sequenciais: classificação␣
 ↪temática, resumo executivo, revisão editorial e redação narrativa. Os dados␣
 ↪analisados correspondem aos pronunciamentos oficiais do Senado Federal␣
 ↪brasileiro no período especificado.*
"""

        except Exception as e:
            return f"""#  ERRO NA GERAÇÃO DO RELATÓRIO NARRATIVO

Não foi possível gerar o relatório narrativo consolidado devido ao seguinte␣
 ↪erro:
{str(e)}

**Informações técnicas:**
- Período solicitado: {periodo}
```

```
- Modelo tentado: {dados_para_sintese['modelo_usado']}
- Total de discursos: {dados_para_sintese['total_discursos']}
"""

# Exemplo de uso
async def main():
    """Função principal para executar a análise"""

    # CONFIGURAÇÃO DA API KEY
    # Escolha UMA das opções abaixo:

    # OPÇÃO 1: Definir diretamente (não recomendado para produção)
    # api_key = "SUA_CHAVE_AQUI"
    # processor = SenadorDiscursoProcessor(api_key=api_key)

    # OPÇÃO 2: Usar variável de ambiente (recomendado)
    # os.environ['GOOGLE_API_KEY'] = "SUA_CHAVE_AQUI"
    # processor = SenadorDiscursoProcessor()

    # OPÇÃO 3: No Google Colab (recomendado para Colab)
    try:
        from google.colab import userdata
        api_key = userdata.get('GOOGLE_API_KEY')
        processor =␣
↪SenadorDiscursoProcessor(api_key="AIzaSyBXIwL63qEsD8UmHL_CF-5uulDTsJhPHDg")
    except ImportError:
        # Se não estiver no Colab, tentar variável de ambiente
        try:
            processor = SenadorDiscursoProcessor()
        except ValueError as e:
            print(e)
            print("\n" + "="*60)
            print("CONFIGURE SUA API KEY ANTES DE CONTINUAR!")
            print("="*60)
            return

    # Definir período de análise (últimos 30 dias)
    data_fim = date.today().strftime('%Y%m%d')  # Formato AAAAMMDD
    data_inicio = (date.today() - timedelta(days=30)).strftime('%Y%m%d')  #␣
↪Formato AAAAMMDD

    print(f"  Iniciando análise de discursos do Senado Federal")
    print(f"  Período: {data_inicio} a {data_fim}")

    # Processar período completo
    resultados = await processor.processar_periodo_completo(data_inicio,␣
↪data_fim)
```

```
    # Gerar relatórios finais
    if resultados:
        print("  Gerando relatório estruturado...")
        relatorio_estruturado = processor.gerar_relatorio_final(resultados,␣
↪f"{data_inicio} a {data_fim}")

        print("  Gerando relatório narrativo...")
        relatorio_narrativo = await processor.
↪gerar_relatorio_narrativo_periodo(resultados, f"{data_inicio} a {data_fim}")

        # Salvar relatório estruturado
        nome_arquivo_estruturado =␣
↪f"analise_discursos_senado_{data_inicio}_a_{data_fim}_estruturado.md"
        with open(nome_arquivo_estruturado, 'w', encoding='utf-8') as f:
            f.write(relatorio_estruturado)

        # Salvar relatório narrativo
        nome_arquivo_narrativo =␣
↪f"analise_discursos_senado_{data_inicio}_a_{data_fim}_narrativo.md"
        with open(nome_arquivo_narrativo, 'w', encoding='utf-8') as f:
            f.write(relatorio_narrativo)

        print(f"  Relatório estruturado salvo em: {nome_arquivo_estruturado}")
        print(f"  Relatório narrativo salvo em: {nome_arquivo_narrativo}")
        print(f"  Análise concluída! {len(resultados)} discursos processados.")
        print(f"  Modelo utilizado: gemini-1.5-flash")
        print(f"  Pipeline: Classificador → Resumidor → Revisor → Redator␣
↪Narrativo")
    else:
        print("  Nenhum resultado para gerar relatório")

# Executar o pipeline
if __name__ == "__main__":
    print("  SISTEMA DE ANÁLISE DE DISCURSOS DO SENADO FEDERAL")
    print("="*60)
    print("ANTES DE EXECUTAR, CONFIGURE SUA API KEY DO GOOGLE AI:")
    print("")
    print("1. Obtenha sua chave em: https://makersuite.google.com/app/apikey")
    print("2. Configure usando UMA das opções:")
    print("   • No Colab: Secrets → GOOGLE_API_KEY → sua_chave")
    print("   • Variável de ambiente: os.environ['GOOGLE_API_KEY'] =␣
↪'sua_chave'")
    print("   • Parâmetro direto:␣
↪SenadorDiscursoProcessor(api_key='sua_chave')")
    print("")
```

```
    print("="*60)

    try:
        asyncio.run(main())
    except Exception as e:
        print(f"\n ERRO DURANTE EXECUÇÃO: {e}")
        print("\n DICAS PARA RESOLVER:")
        print("1. Verifique se a API key está correta")
        print("2. Confirme que tem créditos na conta Google AI")
        print("3. Verifique sua conexão com internet")
        print("4. Tente executar novamente em alguns minutos")
```

```
 SISTEMA DE ANÁLISE DE DISCURSOS DO SENADO FEDERAL
============================================================
ANTES DE EXECUTAR, CONFIGURE SUA API KEY DO GOOGLE AI:

1. Obtenha sua chave em: https://makersuite.google.com/app/apikey
2. Configure usando UMA das opções:
   • No Colab: Secrets → GOOGLE_API_KEY → sua_chave
   • Variável de ambiente: os.environ['GOOGLE_API_KEY'] = 'sua_chave'
   • Parâmetro direto: SenadorDiscursoProcessor(api_key='sua_chave')

============================================================
 API do Google AI configurada com sucesso!
 Chave: AIzaSyBX…PHDg
 Variável de ambiente GOOGLE_API_KEY definida para ADK
 Iniciando análise de discursos do Senado Federal
 Período: 20250503 a 20250602
 Iniciando coleta de discursos de 20250503 a 20250602
 Coletados 16 discursos do período 20250503 a 20250602
 Iniciando análise de 16 discursos…
 Processando discurso 1/16 - Jorge Kajuru
 Discurso 1 processado com sucesso
 Preview: ##  CLASSIFICAÇÃO TEMÁTICA

- **Tema Principal:** Saúde
- **Subtemas Específicos:** Atenção à saúde, direitos das pessoas com
queimaduras, políticas públicas de saúde, sistema de saúde brasileiro.
- …
--------------------------------------------------
 Processando discurso 2/16 - Izalci Lucas
 Discurso 2 processado com sucesso
 Preview: ##  CLASSIFICAÇÃO TEMÁTICA

- **Tema Principal:** Corrupção e Desvios de Recursos Públicos
- **Subtemas Específicos:** INSS, Operação Sem Desconto, Polícia Federal,
Controladoria-Geral da União (CGU)…
```

```
--------------------------------------------------
  Processando discurso 3/16 - Jorge Kajuru
  Discurso 3 processado com sucesso
  Preview: ##   CLASSIFICAÇÃO TEMÁTICA

- **Tema Principal:** Desenvolvimento Humano
- **Subtemas Específicos:** Índice de Desenvolvimento Humano (IDH), Educação,
Democracia, Política Econômica, Desenvolvimento …
--------------------------------------------------
  Processando discurso 4/16 - Sergio Moro
  Discurso 4 processado com sucesso
  Preview: ##   CLASSIFICAÇÃO TEMÁTICA

- **Tema Principal:** Corrupção e Fraudes
- **Subtemas Específicos:**  Fraudes no INSS, corrupção no Brasil, transparência
pública, combate à corrupção, gestão pública efi…
--------------------------------------------------
  Processando discurso 5/16 - Cleitinho
  Discurso 5 processado com sucesso
  Preview: ##   CLASSIFICAÇÃO TEMÁTICA

- **Tema Principal:** Segurança Social/Previdência Social
- **Subtemas Específicos:** Fraude no INSS, investigação de crimes,
responsabilização de agentes públicos, Comiss…
--------------------------------------------------
  Processando discurso 6/16 - Plínio Valério

ERROR:asyncio:Task was destroyed but it is pending!
task: <Task pending name='Task-115' coro=<AsyncClient.aclose() running at
/usr/local/lib/python3.11/dist-packages/httpx/_client.py:1978>>
/usr/lib/python3.11/asyncio/base_events.py:679: RuntimeWarning: coroutine
'AsyncClient.aclose' was never awaited
  self._ready.clear()
RuntimeWarning: Enable tracemalloc to get the object allocation traceback
ERROR:asyncio:Task was destroyed but it is pending!
task: <Task pending name='Task-128' coro=<AsyncClient.aclose() running at
/usr/local/lib/python3.11/dist-packages/httpx/_client.py:1978>>
ERROR:asyncio:Task was destroyed but it is pending!
task: <Task pending name='Task-141' coro=<AsyncClient.aclose() running at
/usr/local/lib/python3.11/dist-packages/httpx/_client.py:1978>>
ERROR:asyncio:Task was destroyed but it is pending!
task: <Task pending name='Task-154' coro=<AsyncClient.aclose() running at
/usr/local/lib/python3.11/dist-packages/httpx/_client.py:1978>>

  Discurso 6 processado com sucesso
  Preview: ##   CLASSIFICAÇÃO TEMÁTICA

- **Tema Principal:** Transparência e Fiscalização de Recursos Públicos
- **Subtemas Específicos:** Orçamento público 2026,  CPI das ONGs, repasses a
```

```
ONGs, desvio de recur…
--------------------------------------------------
  Processando discurso 7/16 - Marcio Bittar
  Discurso 7 processado com sucesso
  Preview: ##   CLASSIFICAÇÃO TEMÁTICA

- **Tema Principal:** Política
- **Subtemas Específicos:** Morre, falecimento, pesar, condolências,
solidariedade.
- **Área de Política Pública:** Nenhuma área específica …
--------------------------------------------------
  Processando discurso 8/16 - Chico Rodrigues
  Discurso 8 processado com sucesso
  Preview: ##   CLASSIFICAÇÃO TEMÁTICA

- **Tema Principal:** Cibersegurança
- **Subtemas Específicos:** Marco legal para cibersegurança, infraestrutura
digital, atuação legislativa em cibersegurança, conferênci…
--------------------------------------------------
  Processando discurso 9/16 - Esperidião Amin
  Discurso 9 processado com sucesso
  Preview: ##   CLASSIFICAÇÃO TEMÁTICA

- **Tema Principal:**  Transparência e Combate à Corrupção
- **Subtemas Específicos:** Fraude no INSS,  investigação parlamentar, Comissão
Parlamentar Mista de Inquérito (…
--------------------------------------------------
  Processando discurso 10/16 - Eduardo Girão
  Discurso 10 processado com sucesso
  Preview: ##   CLASSIFICAÇÃO TEMÁTICA

- **Tema Principal:**  Combate à Corrupção
- **Subtemas Específicos:** Fraudes no INSS, Irregularidades na CBF,
Investigação Parlamentar (CPMI e CPI)
- **Área de Política…
--------------------------------------------------
  Processando discurso 11/16 - Jorge Kajuru
  Discurso 11 processado com sucesso
  Preview: ##   CLASSIFICAÇÃO TEMÁTICA

- **Tema Principal:** Gestão de Desastres Naturais e Políticas Públicas
- **Subtemas Específicos:** Comparação de ações governamentais em eventos
climáticos extremos (esti…
--------------------------------------------------
  Processando discurso 12/16 - Marcos do Val
  Discurso 12 processado com sucesso
  Preview: ##   CLASSIFICAÇÃO TEMÁTICA
```

```
- **Tema Principal:** Direitos Humanos e Poder Judiciário
- **Subtemas Específicos:** Violações de direitos humanos,  independência
judicial,  União Interparlamentar (UIP)…
--------------------------------------------------
 Processando discurso 13/16 - Veneziano Vital do Rêgo
 Discurso 13 processado com sucesso
 Preview: ##  CLASSIFICAÇÃO TEMÁTICA

- **Tema Principal:** Segurança Pública
- **Subtemas Específicos:** Guardas Municipais, Agentes de Trânsito, PEC nº
37/2022,  Segurança Pública Municipal,  Recursos Humano…
--------------------------------------------------
 Processando discurso 14/16 - Confúcio Moura

ERROR:asyncio:Task was destroyed but it is pending!
task: <Task pending name='Task-180' coro=<AsyncClient.aclose() running at
/usr/local/lib/python3.11/dist-packages/httpx/_client.py:1978>>
ERROR:asyncio:Task was destroyed but it is pending!
task: <Task pending name='Task-193' coro=<AsyncClient.aclose() running at
/usr/local/lib/python3.11/dist-packages/httpx/_client.py:1978>>
ERROR:asyncio:Task was destroyed but it is pending!
task: <Task pending name='Task-206' coro=<AsyncClient.aclose() running at
/usr/local/lib/python3.11/dist-packages/httpx/_client.py:1978>>
ERROR:asyncio:Task was destroyed but it is pending!
task: <Task pending name='Task-219' coro=<AsyncClient.aclose() running at
/usr/local/lib/python3.11/dist-packages/httpx/_client.py:1978>>
ERROR:asyncio:Task was destroyed but it is pending!
task: <Task pending name='Task-232' coro=<AsyncClient.aclose() running at
/usr/local/lib/python3.11/dist-packages/httpx/_client.py:1978>>
ERROR:asyncio:Task was destroyed but it is pending!
task: <Task pending name='Task-245' coro=<AsyncClient.aclose() running at
/usr/local/lib/python3.11/dist-packages/httpx/_client.py:1978>>
ERROR:asyncio:Task was destroyed but it is pending!
task: <Task pending name='Task-258' coro=<AsyncClient.aclose() running at
/usr/local/lib/python3.11/dist-packages/httpx/_client.py:1978>>

 Discurso 14 processado com sucesso
 Preview: ##  CLASSIFICAÇÃO TEMÁTICA

- **Tema Principal:** Educação
- **Subtemas Específicos:** Qualidade da educação brasileira, políticas
educacionais coordenadas entre os entes federativos, desenvolvimento…
--------------------------------------------------
 Processando discurso 15/16 - Esperidião Amin
 Discurso 15 processado com sucesso
 Preview: ##  CLASSIFICAÇÃO TEMÁTICA

- **Tema Principal:** Transporte Rodoviário
- **Subtemas Específicos:**  Segurança do Trabalho, legislação trabalhista,
```

```
saúde dos motoristas,  produtividade no transporte …
--------------------------------------------------
  Processando discurso 16/16 - Paulo Paim
  Discurso 16 processado com sucesso
  Preview: ##   CLASSIFICAÇÃO TEMÁTICA

- **Tema Principal:** Direito do Trabalho
- **Subtemas Específicos:** Pejotização,  Direitos Trabalhistas, Competência da
Justiça do Trabalho, Supremo Tribunal Federal (ST…
--------------------------------------------------
  Gerando relatório estruturado…
  Gerando relatório narrativo…
  Relatório estruturado salvo em:
analise_discursos_senado_20250503_a_20250602_estruturado.md
  Relatório narrativo salvo em:
analise_discursos_senado_20250503_a_20250602_narrativo.md
  Análise concluída! 16 discursos processados.
  Modelo utilizado: gemini-1.5-flash
  Pipeline: Classificador → Resumidor → Revisor → Redator Narrativo
```