Projeto: Sistema de Resumidor de PDFs com Inteligência Artificial

Gabriel Conrado da Silva

RESUMO
Este trabalho apresenta o desenvolvimento e implementação de um sistema de sumarização automática de documentos PDF utilizando Inteligência Artificial. A solução emprega a API do Google Gemini para gerar resumos concisos e informativos a partir de textos extensos, combinando processamento de linguagem natural com interface web intuitiva. A arquitetura do sistema integra módulos especializados para extração de texto, processamento por IA e geração de documentos formatados, demonstrando a aplicabilidade prática de modelos de linguagem grande em problemas cotidianos de gestão de informação. O estudo evidencia o potencial das tecnologias de IA generativa para otimizar o processamento e compreensão de conteúdo textual em larga escala.

ABSTRACT
This paper presents the development and implementation of an automated PDF document summarization system using Artificial Intelligence. The solution employs the Google Gemini API to generate concise and informative summaries from extensive texts, combining natural language processing with an intuitive web interface. The system architecture integrates specialized modules for text extraction, AI processing, and formatted document generation, demonstrating the practical applicability of large language models in everyday information management problems. The study highlights the potential of generative AI technologies to optimize the processing and understanding of large-scale textual content.

Keywords: Artificial Intelligence, Automated Summarization, Natural Language Processing, Google Gemini, PDF, Python, Gradio.

Índice:

1. Introdução e Objetivos

1.1 Contexto do Projeto

Este projeto foi desenvolvido como parte de um curso de Inteligência Artificial aplicada, demonstrando a integração de modelos de linguagem grandes (LLMs) com processamento de documentos e interface web. O sistema utiliza a API do Google Gemini para realizar sumarização automática de documentos PDF.

1.2 Objetivos Principais

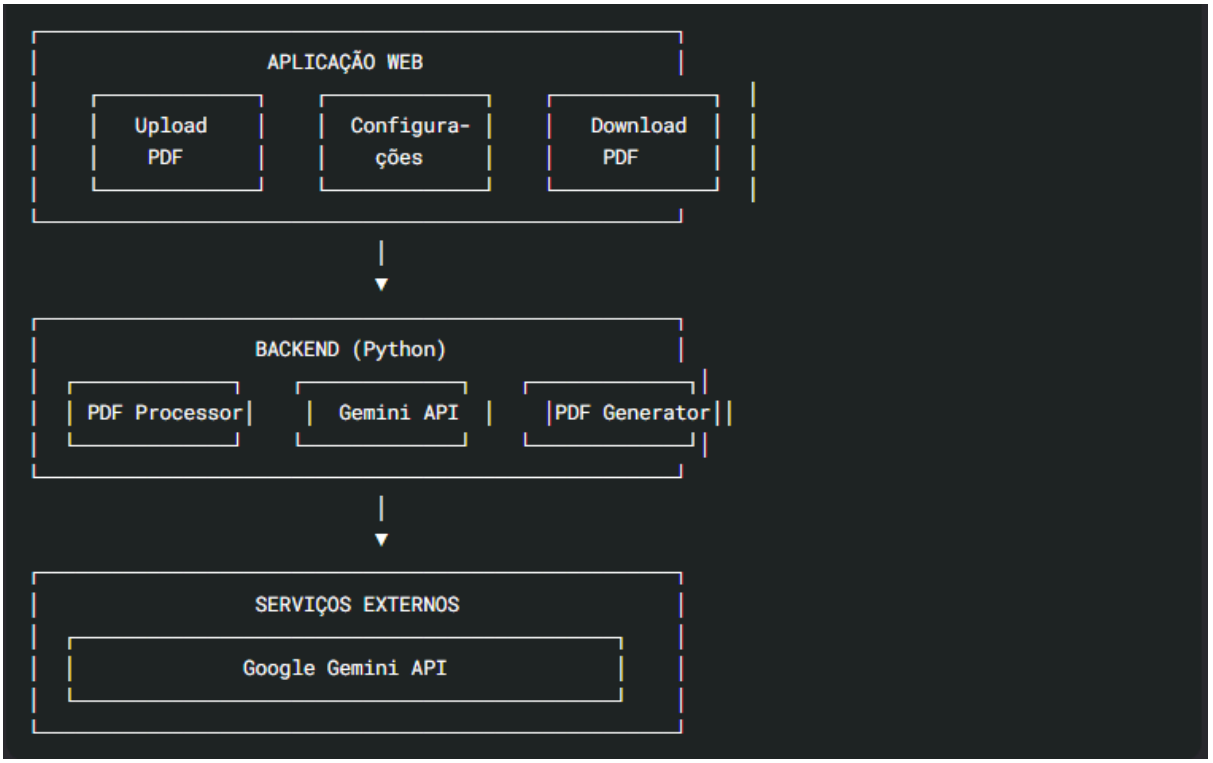
- Automatização: Criar um sistema que automaticamente resume documentos PDF extensos
- Acessibilidade: Fornecer interface web intuitiva para usuários não técnicos
- Eficiência: Reduzir tempo de leitura através de resumos concisos
- Qualidade: Manter a essência e informações principais dos documentos
- Portabilidade: Gerar resultados em formato PDF para fácil compartilhamento

1.3 Público-Alvo

- Estudantes e pesquisadores que precisam processar artigos acadêmicos
- Profissionais que analisam relatórios técnicos extensos
- Qualquer usuário que necessite compreender rapidamente documentos longos

2. Arquitetura do Sistema

2.1 Diagrama de Componentes



2.2 Fluxo de Dados

- Entrada: Usuário faz upload de PDF através da interface web
- Processamento: Sistema extrai texto e envia para API Gemini
- IA: Gemini gera resumo baseado no texto
- Saída: Sistema cria PDF formatado com o resumo
- Download: Usuário baixa o resultado final

3. Configuração e Dependências

3.1 requirements.txt

```
google-generativeai  # Biblioteca oficial para API Gemini
gradio>=4.0.0        # Framework para interfaces web em Python
PyPDF2>=3.0.0        # Extração de texto de arquivos PDF
python-dotenv>=1.0.0 # Gerenciamento de variáveis de ambiente
tqdm>=4.0.0          # Barra de progresso para operações
reportlab>=4.0.0     # Geração de documentos PDF programaticamente
```

Bibliotecas requeridas para serem instaladas em um comando

3.2 Variáveis de Ambiente (.env)

```
GEMINI_API_KEY=sua_chave_api_aqui  # Chave de autenticação
MAX_TOKENS=1000                    # Limite de tokens por resposta
TEMPERATURE=0.3                    # Criatividade das respostas (0-1)
```

Parâmetros passados para o Gemini

4. Módulo de Configuração

4.1 config.py

```python
import os
from dotenv import load_dotenv

# Carregar variáveis de ambiente
load_dotenv()

# Configurações da API Gemini
GEMINI_API_KEY = os.getenv("GEMINI_API_KEY")
MODEL_NAME = "gemini-1.0-pro"
MAX_TOKENS = int(os.getenv("MAX_TOKENS", 1000))
TEMPERATURE = float(os.getenv("TEMPERATURE", 0.3))

# Configurações de processamento
MAX_PDF_PAGES = 50
CHUNK_SIZE = 4000
CHUNK_OVERLAP = 200

# Template do prompt para sumarização
SUMMARY_PROMPT = """
Por favor, produza um resumo conciso e informativo do texto abaixo.
O resumo deve capturar os pontos principais e as ideias mais importantes.
Se o texto for técnico, foque nos conceitos e conclusões fundamentais.

Mantenha o resumo em português e com no máximo {max_length} palavras.

Texto para resumir:
{text}
"""
```

4.2 Explicação

- load_dotenv(): Carrega variáveis do arquivo .env para o ambiente
- Variáveis de Configuração: Controlam comportamento da API e processamento
- SUMMARY_PROMPT: Template que guia o modelo Gemini na sumarização

5. Processamento de PDF

5.1 pdf_processor.py

```python
import PyPDF2
import io
from typing import List, Tuple

def extract_text_from_pdf(pdf_file) -> Tuple[str, int]:
    """
    Extrai texto de arquivos PDF com tratamento de erros
    """
    text = ""
    num_pages = 0

    try:
        # Detecta tipo de entrada (arquivo ou caminho)
        if hasattr(pdf_file, 'read'):
            pdf_reader = PyPDF2.PdfReader(io.BytesIO(pdf_file.read()))
        else:
            pdf_reader = PyPDF2.PdfReader(pdf_file)

        num_pages = len(pdf_reader.pages)

        # Extrai texto de até 50 páginas
        for page_num in range(min(num_pages, MAX_PDF_PAGES)):
            page = pdf_reader.pages[page_num]
            text += page.extract_text() + "\n"

    except Exception as e:
        raise Exception(f"Erro ao processar PDF: {str(e)}")

    return text, num_pages

def chunk_text(text: str, chunk_size: int = 4000, overlap: int = 200) -> List[str]:
    """
    Divide textos longos em partes menores para processamento
    """
    if len(text) <= chunk_size:
        return [text]

    chunks = []
    start = 0

    while start < len(text):
        end = start + chunk_size
        # Preserva palavras inteiras nos pontos de quebra
        if end < len(text):
            while end > start and text[end] not in (' ', '\n', '.', ',', ';', ':'):
                end -= 1
            if end == start:
                end = start + chunk_size

        chunks.append(text[start:end])
        start = end - overlap  # Overlap para manter contexto
```

5.2 Técnicas de Processamento

- Extração Segura: Manipula diferentes formatos de entrada
- Limite de Páginas: Previne processamento infinito de documentos muito longos
- Chunking Inteligente: Divide texto preservando contexto entre partes

- Tratamento de Erros: Captura exceções e fornece feedback útil

6. Integração com API Gemini

6.1 Summarizer.py

```python
import google.generativeai as genai
from typing import List, Optional
from config import GEMINI_API_KEY, MODEL_NAME, MAX_TOKENS, TEMPERATURE, SUMMARY_PROMPT
import time

# Configuração inicial da API
try:
    genai.configure(api_key=GEMINI_API_KEY)
except Exception as e:
    print(f"Erro ao configurar API Gemini: {e}")

def generate_summary(text: str, max_length: int = 300) -> str:
    """
    Gera resumo usando API Gemini com tratamento robusto de erros
    """
    # Validações iniciais
    if not text or not text.strip():
        return "Nenhum texto válido para resumir."

    if not GEMINI_API_KEY or GEMINI_API_KEY == "sua_chave_api_aqui":
        return "Erro: API key não configurada."

    try:
        # Configuração do modelo
        generation_config = {
            "temperature": TEMPERATURE,
            "top_p": 0.95,
            "top_k": 40,
            "max_output_tokens": MAX_TOKENS,
        }

        model = genai.GenerativeModel(
            model_name=MODEL_NAME,
            generation_config=generation_config
        )

        # Prepara e envia prompt
        prompt = SUMMARY_PROMPT.format(max_length=max_length, text=text[:10000])
        response = model.generate_content(prompt)

        return response.text

    except Exception as e:
        return f"Erro ao gerar resumo: {str(e)}"
```

6.2 Estratégias de Sumarização

- Prompt Engineering: Template otimizado para respostas em português
- Controle de Qualidade: Limites de tamanho e validações
- Fallback Hierarchy: Tenta modelos alternativos em caso de falha
- Rate Limiting: Pausas entre requisições para evitar bloqueio

7. Geração de PDFs

7.1 pdf_generator.py

```python
import os
from reportlab.pdfgen import canvas
from reportlab.lib.pagesizes import letter, A4
from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle
from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer
from reportlab.lib.units import inch
from reportlab.lib import colors
from datetime import datetime
from config import MODEL_NAME

def criar_pdf_resumo(texto_original: str, resumo: str, titulo: str = "Resumo Gerado por IA") ->
str:
    """
    Cria PDF profissional com metadados e formatação
    """
    # Configuração do documento
    downloads_dir = "downloads"
    if not os.path.exists(downloads_dir):
        os.makedirs(downloads_dir)

    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    filename = f"{downloads_dir}/resumo_{timestamp}.pdf"

    # Estrutura do PDF
    doc = SimpleDocTemplate(filename, pagesize=A4)
    styles = getSampleStyleSheet()

    # Conteúdo organizado
    conteudo = [
        Paragraph(titulo, styles['Heading1']),
        Spacer(1, 0.2 * inch),
        Paragraph("Informações do Processamento", styles['Heading2']),
        # ... metadados e estatísticas ...
        Paragraph("Resumo Gerado", styles['Heading2']),
        # ... conteúdo do resumo ...
    ]

    doc.build(conteudo)
    return filename
```

7.2 Características do PDF Gerado

- Layout Professional: Margens, espaçamento e hierarquia visual
- Metadados Úteis: Estatísticas de processamento e informações técnicas
- Formatação Rica: Estilos diferenciados para títulos, subtítulos e texto
- Timestamp Único: Nomes de arquivo com data/hora para evitar sobrescrita

8. Interface do Usuário

8.1 app.py (Interface Gradio)

```python
import gradio as gr
from summarizer import generate_summary, summarize_large_text
from pdf_processor import extract_text_from_pdf
from pdf_generator import criar_pdf_resumo, criar_pdf_simples, limpar_downloads_antigos
import time
import os

def process_pdf_summary(pdf_file, summary_length, detailed_summary, pdf_type):
    """
    Orquestra todo o processo de sumarização
    """
    # Fluxo principal: Upload → Extração → Sumarização → Geração PDF
    # ... implementação completa ...

# Interface com Gradio
with gr.Blocks(title="Resumidor de PDF com Gemini API", theme=gr.themes.Soft()) as demo:
    # Componentes da interface
    gr.Markdown("# 📄 Resumidor de PDF com Gemini API")
    pdf_input = gr.File(label="Upload do PDF", file_types=[".pdf"])
    # ... outros componentes ...

    # Conexão de eventos
    process_btn.click(
        fn=process_pdf_summary,
        inputs=[pdf_input, length_slider, detailed, pdf_type],
        outputs=[summary_output, stats_output, pdf_output]
    )
```

8.2 Componentes da Interface

- Upload de Arquivo: Suporte a PDF com validação
- Controles Interativos: Sliders, checkboxes e botões
- Feedback Visual: Estatísticas de processamento em tempo real
- Download Integrado: Botão de download direto do PDF resultante

9. Utilitários de Arquivo

9.1 file_utils.py

```python
import os
from typing import Optional
import shutil
from datetime import datetime, timedelta

def ensure_directory_exists(directory_path: str) -> bool:
    """
    Garante existência de diretórios com criação segura
    """
    try:
        if not os.path.exists(directory_path):
            os.makedirs(directory_path, exist_ok=True)
        return True
    except Exception as e:
        print(f"Erro ao criar diretório {directory_path}: {e}")
        return False
```

9.2 Funcionalidades de Utilitários

- Gestão de Diretórios: Criação e verificação segura
- Validação de Arquivos: Verificação de tipo, tamanho e integridade
- Operações Seguras: Move, copia e deleta com tratamento de erros
- Limpeza Automática: Remove arquivos temporários antigos

10. Fluxo de Trabalho Completo

10.1 Sequência de Execução

- Inicialização: Carrega configurações e limpa arquivos temporários
- Upload: Usuário seleciona arquivo PDF através da interface
- Validação: Sistema verifica se o arquivo é um PDF válido
- Extração: Texto é extraído do PDF usando PyPDF2
- Pré-processamento: Texto é dividido em chunks se necessário
- Sumarização: Chunks são enviados para API Gemini
- Pós-processamento: Respostas são combinadas e formatadas
- Geração PDF: Resultado é convertido para PDF formatado
- Apresentação: Interface mostra resultados e opção de download

10.2 Tratamento de Erros

- Validação em Cada Etapa: Verificações antes de cada operação
- Fallbacks Graduais: Tenta alternativas em caso de falha
- Feedback Descritivo: Mensagens de erro claras para usuário
- Logs Detalhados: Registros para debugging e monitoramento

11. Conclusão e Aprendizados

11.1 Conquistas Técnicas

- Integração bem-sucedida com API Gemini
- Processamento eficiente de documentos PDF
- Interface web intuitiva e responsiva
- Geração de PDFs profissionais automaticamente
- Sistema robusto com tratamento completo de erros

11.2 Aprendizados Desenvolvidos

- Engenharia de Prompts: Como comunicar-se efetivamente com LLMs
- Processamento de Documentos: Técnicas para extrair e manipular texto
- APIs RESTful: Integração com serviços web externos
- Interfaces Web: Criação de aplicações web com Python
- Gestão de Projetos: Organização de código em módulos especializados

11.3 Aplicações Práticas

- Acadêmico: Resumo de artigos científicos e papers
- Corporativo: Análise de relatórios técnicos e documentação
- Pessoal: Processamento de manuais e documentação extensa
- Educacional: Ferramenta de estudo para estudantes