## Guia Definitivo: Youtube Translator

Este documento é o registro completo e o manual de operações para o projeto "Youtube Transcript Translator", uma aplicação de desktop desenvolvida em Python para automatizar o processo de captura, tradução e síntese de voz de transcrições de vídeos do YouTube.

Sumário

[Guia Definitivo: Youtube Translator 1](#_Toc200230589)

[1. Visão Geral do Projeto 2](#_Toc200230590)

[1.1. Descrição do Projeto 2](#_Toc200230591)

[1.2. Prompt para Replicação por IA 2](#_Toc200230592)

[1.3. Glossário de Tecnologias 3](#_Toc200230593)

[2. Configuração do Ambiente de Desenvolvimento 4](#_Toc200230594)

[2.1. Dica Essencial: Iniciando o Ambiente de Trabalho 4](#_Toc200230595)

[2.2. Preparação do Ambiente no VS Code 4](#_Toc200230596)

[2.3. Instalação de Dependências 5](#_Toc200230597)

[3. Estrutura do Projeto e Fluxo de Trabalho 5](#_Toc200230598)

[3.1. Estrutura de Diretórios e Arquivos 5](#_Toc200230599)

[3.2. Fluxo de Operação da Aplicação 6](#_Toc200230600)

[4. Código Fonte da Aplicação 6](#_Toc200230601)

[4.1. Arquivo de Configuração (.gitignore) 6](#_Toc200230602)

[4.2. Módulo de Lógica (core\_logic.py) 6](#_Toc200230603)

[4.3. Módulo de Interface (main.py) 13](#_Toc200230604)

[5. Execução e Teste 17](#_Toc200230605)

[6. Base de Conhecimento do Projeto 17](#_Toc200230606)

[6.1. Diretrizes Chave do Projeto (Refinamento das Solicitações) 17](#_Toc200230607)

[6.2. Log de Erros Histórico (system\_errors.csv) 18](#_Toc200230608)

### 1. Visão Geral do Projeto

## 1.1. Descrição do Projeto

O "Youtube Transcript Translator" é uma aplicação de desktop criada com Python e a biblioteca de interface gráfica CustomTkinter. O objetivo principal da ferramenta é automatizar o fluxo de trabalho de obtenção de conteúdo de vídeos do YouTube, permitindo duas vias de captura:

1. **Download Direto:** Extrai a transcrição gerada automaticamente pelo YouTube.
2. **Análise de Áudio:** Baixa o áudio do vídeo e utiliza o modelo de IA Whisper (da OpenAI) para realizar uma transcrição local de alta fidelidade.

Após a captura, a aplicação detecta o idioma original, permite a tradução para o Português-BR (utilizando a biblioteca deep-translator) e, por fim, sintetiza a tradução em um arquivo de áudio .mp3 (usando a biblioteca TTS da Coqui-AI).

Todos os artefatos gerados (arquivos .txt, .json, .mp3) são organizados em uma estrutura de pastas lógica. O sistema também mantém dois logs em formato .csv para análise de dados: um master\_log.csv para registrar todas as operações bem-sucedidas e um system\_errors.csv como base de conhecimento para registrar falhas e suas respectivas soluções aplicadas durante o desenvolvimento.

## 1.2. Prompt para Replicação por IA

Crie uma aplicação de desktop em Python chamada "Youtube Transcript Translator" utilizando a biblioteca CustomTkinter para a interface gráfica. O ambiente de desenvolvimento deve ser gerenciado pelo Conda, utilizando Python 3.11.

**Funcionalidades Principais:**

1. **Interface Gráfica:** A UI deve ter um campo de entrada para o link do vídeo, botões para cada ação principal (Baixar Transcrição, Transcrição via Áudio, Traduzir, Gerar Áudio) e um painel de log para exibir o progresso das tarefas em tempo real.
2. **Captura de Transcrição (Duas Vias):**
   * **Via 1 (Download Direto):** Usar yt-dlp para baixar a melhor transcrição automática disponível de um vídeo.
   * **Via 2 (Análise de Áudio):** Usar yt-dlp para baixar o áudio e a biblioteca openai-whisper (modelo 'tiny') para transcrevê-lo localmente.
3. **Detecção e Armazenamento:** Ambas as vias de captura devem detectar e armazenar o idioma original da transcrição. Os resultados devem ser salvos em formato .txt (texto puro) e .json (com metadados: ID do vídeo, título, data, idioma, etc.) na pasta logs/transcricao\_original/. Os nomes dos arquivos devem seguir o padrão [Data]\_[IDvideo]\_[Sufixo].ext, onde o sufixo \_audio diferencia a transcrição via áudio.
4. **Tradução Inteligente:**
   * Ao acionar a tradução, o sistema deve verificar o idioma de origem.
   * Se for português, deve informar que a tradução não é necessária.
   * Se for outro idioma, deve usar a biblioteca deep-translator para traduzir para Português-BR.
   * O resultado deve ser salvo em logs/transcricao\_traduzida/.
5. **Síntese de Voz (TTS):**
   * Após uma tradução bem-sucedida, o usuário pode gerar um arquivo de áudio .mp3 do texto traduzido.
   * Utilize a biblioteca TTS (da Coqui-AI) com o modelo tts\_models/pt/cv/vits.
   * O resultado deve ser salvo em logs/audio\_da\_traducao/mp3/.
6. **Sistema de Log Robusto:**
   * **Log Visual:** A interface deve exibir cada etapa de cada processo.
   * **Log Mestre de Sucessos:** Um arquivo logs/master\_log.csv, delimitado por ponto e vírgula, deve registrar cada operação bem-sucedida com detalhes (timestamp, ID do vídeo, título, tipo de operação, status, caminhos dos arquivos).
   * **Log de Erros:** Um arquivo logs/system\_errors.csv deve registrar qualquer exceção capturada, detalhando a função, biblioteca, mensagem de erro e a solução proposta.
7. **Execução Assíncrona:** Todas as operações demoradas (downloads, IA, etc.) devem rodar em threads separadas para não travar a interface gráfica.

## 1.3. Glossário de Tecnologias

* **Python:** Versão 3.11.
* **Gerenciador de Ambiente:** Conda (Miniconda).
* **Bibliotecas Principais:**
  + customtkinter: Construção da interface gráfica moderna.
  + yt-dlp: Ferramenta para download de conteúdo e metadados do YouTube.
  + openai-whisper: Modelo de IA para transcrição de áudio de alta qualidade (usamos o modelo 'tiny').
  + deep-translator: Biblioteca leve para tradução de texto utilizando serviços como o Google Translate.
  + TTS (da Coqui-AI): Modelo de IA para síntese de voz (Text-to-Speech) em alta qualidade.
* **Dependências Chave:**
  + ffmpeg: Software essencial para processamento de áudio e vídeo, usado pelo yt-dlp.
* **Comandos de Terminal:**
  + conda activate env: Ativa o ambiente virtual.
  + cd "caminho": Navega entre diretórios.
  + code .: Abre o VS Code na pasta atual.
  + python main.py: Executa a aplicação.
  + mkdir logs: Cria um diretório.
  + type nul > arquivo.ext: Cria um arquivo vazio (no Windows).

### 2. Configuração do Ambiente de Desenvolvimento

## 2.1. Dica Essencial: Iniciando o Ambiente de Trabalho

1. **Ação:** Abrir o "Anaconda Prompt" a partir do Menu Iniciar do Windows.
2. **Ação:** Navegar para a pasta raiz do seu projeto.
   * **Comando para Executar:**  
     cd "C:\Users\nome\_usuario\OneDrive - nome\_usuario\Documentos\GitHub\youtube\_translator"
3. **Ação:** Abrir o VS Code a partir deste terminal.
   * **Comando para Executar:**  
     code .

## 2.2. Preparação do Ambiente no VS Code

1. **Ação:** Selecionar o Interpretador Python: Python 3.11.x ('env').
2. **Verificação:** O canto inferior direito do VS Code deve agora exibir Python 3.11.x ('env').

## 2.3. Instalação de Dependências

**Lembrete de Preparação do Terminal:**

1. **Ação:** Navegar para a pasta raiz do seu projeto.
   * **Comando:** cd "C:\Users\nome\_usuario\OneDrive - nome\_usuario\Documentos\GitHub\youtube\_translator"
2. **Ação:** Ativar o ambiente virtual.
   * **Comando:** conda activate env
3. **Ação:** Garantir que todas as bibliotecas corretas estão instaladas.
   * **Comando para Executar:**  
     pip install customtkinter yt-dlp openai-whisper deep-translator TTS playsound
4. **Ação:** Garantir que o FFmpeg está instalado.
   * **Comando para Executar:**  
     conda install -c conda-forge ffmpeg -y

# 3. Estrutura do Projeto e Fluxo de Trabalho

## 3.1. Estrutura de Diretórios e Arquivos

A estrutura final do projeto, após a execução, será a seguinte:

youtube\_translator/  
|  
├── logs/  
│ ├── transcricao\_original/  
│ │ ├── txt/  
│ │ └── json/  
│ ├── transcricao\_traduzida/  
│ │ ├── txt/  
│ │ └── json/  
│ ├── audio\_da\_traducao/  
│ │ └── mp3/  
│ ├── master\_log.csv  
│ └── system\_errors.csv  
│  
├── .gitignore  
├── core\_logic.py  
└── main.py

## 3.2. Fluxo de Operação da Aplicação

1. O usuário insere um link do YouTube e clica em uma das opções de captura.
2. A aplicação executa a tarefa, exibindo cada passo no log visual.
3. Um arquivo .txt e um .json são salvos na pasta transcricao\_original.
4. Uma entrada é adicionada ao master\_log.csv registrando o sucesso.
5. O botão "Traduzir para Português" é habilitado.
6. O usuário clica para traduzir, e os arquivos traduzidos são salvos em transcricao\_traduzida.
7. Uma nova entrada é adicionada ao master\_log.csv.
8. O botão "Gerar Áudio (TTS)" é habilitado.
9. O usuário clica para gerar o áudio, que é salvo em audio\_da\_traducao.
10. Uma terceira entrada é adicionada ao master\_log.csv.
11. Se ocorrer um erro, ele é registrado no system\_errors.csv e exibido no log visual.

# 4. Código Fonte da Aplicação

## 4.1. Arquivo de Configuração (.gitignore)

# Ambiente virtual e cache  
/env/  
\_\_pycache\_\_/  
\*.pyc  
# Pasta de resultados gerados pela aplicação  
/logs/  
# Arquivos temporários  
temp\_\*  
# Configurações do VS Code  
/.vscode/

## 4.2. Módulo de Lógica (core\_logic.py)

import yt\_dlp  
import os  
import json  
import re  
from datetime import datetime  
import whisper  
from deep\_translator import GoogleTranslator  
from TTS.api import TTS  
  
# --- Sistemas de Log Estruturado ---  
def log\_master\_record(video\_info, operation, status, source\_file, output\_file):  
 log\_dir, log\_file = "logs", os.path.join("logs", "master\_log.csv")  
 os.makedirs(log\_dir, exist\_ok=True)  
 header = "timestamp;video\_id;video\_title;operation\_type;source\_language;status;source\_file\_path;output\_file\_path\n"  
 if not os.path.exists(log\_file):  
 with open(log\_file, 'w', encoding='utf-8') as f:  
 f.write(header)  
   
 timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")  
 video\_id = video\_info.get('id', 'N/A')  
 title = video\_info.get('title', 'N/A').replace('"', "'")  
 lang = video\_info.get('source\_language', 'N/A')  
  
 log\_entry = f"{timestamp};{video\_id};\"{title}\";{operation};{lang};{status};{source\_file or 'N/A'};{output\_file or 'N/A'}\n"  
 with open(log\_file, 'a', encoding='utf-8') as f:  
 f.write(log\_entry)  
  
def log\_error(function\_name, library, error, solution):  
 log\_dir, log\_file = "logs", os.path.join("logs", "system\_errors.csv")  
 os.makedirs(log\_dir, exist\_ok=True)  
 header = "timestamp;function\_name;library\_used;error\_message;proposed\_solution\n"  
 if not os.path.exists(log\_file):  
 with open(log\_file, 'w', encoding='utf-8') as f:  
 f.write(header)  
 timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")  
 error\_cleaned = str(error).replace('"', "'").replace('\n', ' ')  
 log\_entry = f"{timestamp};{function\_name};{library};\"{error\_cleaned}\";\"{solution}\"\n"  
 with open(log\_file, 'a', encoding='utf-8') as f:  
 f.write(log\_entry)  
  
# --- Funções Utilitárias ---  
def clean\_vtt\_content(vtt\_content):  
 lines = vtt\_content.splitlines()  
 cleaned\_lines = []  
 for line in lines:  
 if "WEBVTT" in line or "Kind:" in line or "Language:" in line or line.strip() == "" or "-->" in line:  
 continue  
 line = re.sub(r'<[^>]+>', '', line).strip()  
 line = re.sub(r'\[.\*?\]', '', line).strip()  
 if line and (not cleaned\_lines or line != cleaned\_lines[-1]):  
 cleaned\_lines.append(line)  
 return " ".join(cleaned\_lines)  
  
def save\_transcript(base\_filename, video\_info):  
 content = video\_info.get('content')  
 txt\_path = os.path.join('logs', 'transcricao\_original', 'txt')  
 os.makedirs(txt\_path, exist\_ok=True)  
 full\_txt\_path = os.path.join(txt\_path, f"{base\_filename}.txt")  
 with open(full\_txt\_path, 'w', encoding='utf-8') as f:  
 f.write(content)  
  
 json\_path = os.path.join('logs', 'transcricao\_original', 'json')  
 os.makedirs(json\_path, exist\_ok=True)  
 full\_json\_path = os.path.join(json\_path, f"{base\_filename}.json")  
 with open(full\_json\_path, 'w', encoding='utf-8') as f:  
 json.dump(video\_info, f, ensure\_ascii=False, indent=4)  
 return full\_txt\_path, full\_json\_path  
  
# --- Funções Principais ---  
def download\_transcript(video\_url, log\_callback):  
 video\_info, final\_paths = {}, {}  
 try:  
 log\_callback("1. Iniciando download da transcrição...")  
 ydl\_opts = {'writeautomaticsub': True, 'skip\_download': True, 'outtmpl': 'temp\_subtitle', 'quiet': True}  
 with yt\_dlp.YoutubeDL(ydl\_opts) as ydl:  
 info = ydl.extract\_info(video\_url, download=False)  
 ydl.download([video\_url])  
 log\_callback("2. Arquivo bruto baixado.")  
  
 subtitle\_file\_path, downloaded\_lang = next(((f"temp\_subtitle.{lc}.vtt", lc.split('-')[0]) for lc in info.get('automatic\_captions', {}) if os.path.exists(f"temp\_subtitle.{lc}.vtt")), (None, None))  
 if not subtitle\_file\_path: raise ValueError("Nenhuma transcrição automática encontrada.")  
   
 with open(subtitle\_file\_path, 'r', encoding='utf-8') as f:  
 cleaned\_content = clean\_vtt\_content(f.read())  
 os.remove(subtitle\_file\_path)  
   
 log\_callback(f"3. Texto limpo. Idioma detectado: {downloaded\_lang}.")  
 if not cleaned\_content: raise ValueError("Falha ao extrair texto da transcrição.")  
   
 log\_callback("4. Salvando arquivos finais...")  
 video\_id, upload\_date\_str = info.get('id', 'desconhecido'), info.get('upload\_date', '19700101')  
 video\_info = {'id': video\_id, 'title': info.get('title', 'sem\_titulo'), 'upload\_date': datetime.strptime(upload\_date\_str, '%Y%m%d').strftime('%Y-%m-%d'), 'source\_language': downloaded\_lang, 'content': cleaned\_content}  
 base\_filename = f"{upload\_date\_str}\_{video\_id}\_original"  
 txt\_path, json\_path = save\_transcript(base\_filename, video\_info)  
 final\_paths = {'txt\_path': txt\_path, 'json\_path': json\_path, 'video\_info': video\_info}  
  
 log\_master\_record(video\_info, "download\_transcript", "SUCCESS", video\_url, txt\_path)  
 return True, f"Transcrição ({downloaded\_lang}) baixada!", final\_paths  
 except Exception as e:  
 log\_error("download\_transcript", "yt-dlp", str(e), "Verificar link do YouTube e conexão.")  
 log\_master\_record(video\_info, "download\_transcript", "FAIL", video\_url, None)  
 return False, f"Erro: {e}", None  
  
def transcribe\_audio\_local(video\_url, log\_callback):  
 audio\_file\_path, video\_info, final\_paths = None, {}, {}  
 try:  
 log\_callback("1. Iniciando download do áudio...")  
 ydl\_opts = {'format': 'bestaudio/best', 'outtmpl': 'temp\_audio.%(ext)s', 'quiet': True}  
 with yt\_dlp.YoutubeDL(ydl\_opts) as ydl:  
 info = ydl.extract\_info(video\_url, download=True)  
 audio\_file\_path = ydl.prepare\_filename(info)  
 log\_callback("2. Download do áudio concluído.")  
   
 if not audio\_file\_path or not os.path.exists(audio\_file\_path): raise FileNotFoundError("Falha ao baixar áudio.")  
  
 log\_callback("3. Carregando modelo 'tiny' do Whisper...")  
 model = whisper.load\_model("tiny")  
 log\_callback("4. Transcrevendo áudio...")  
 result = model.transcribe(audio\_file\_path, fp16=False)  
 transcribed\_text, detected\_language = result["text"], result["language"]  
 log\_callback(f"5. Transcrição concluída. Idioma: {detected\_language}.")  
  
 if not transcribed\_text: raise ValueError("Não foi possível extrair texto.")  
   
 log\_callback("6. Salvando arquivos finais...")  
 video\_id, upload\_date\_str = info.get('id', 'desconhecido'), info.get('upload\_date', '19700101')  
 video\_info = {'id': video\_id, 'title': info.get('title', 'sem\_titulo'), 'upload\_date': datetime.strptime(upload\_date\_str, '%Y%m%d').strftime('%Y-%m-%d'), 'source\_language': detected\_language, 'content': transcribed\_text}  
 base\_filename = f"{upload\_date\_str}\_{video\_id}\_original\_audio"  
 txt\_path, json\_path = save\_transcript(base\_filename, video\_info)  
 final\_paths = {'txt\_path': txt\_path, 'json\_path': json\_path, 'video\_info': video\_info}  
  
 log\_master\_record(video\_info, "transcribe\_audio\_local", "SUCCESS", video\_url, txt\_path)  
 return True, f"Áudio transcrito ({detected\_language}) com sucesso!", final\_paths  
 except Exception as e:  
 log\_error("transcribe\_audio\_local", "whisper/yt-dlp", str(e), "Verificar FFmpeg e bibliotecas.")  
 log\_master\_record(video\_info, "transcribe\_audio\_local", "FAIL", video\_url, None)  
 return False, f"Erro: {e}", None  
 finally:  
 if audio\_file\_path and os.path.exists(audio\_file\_path): os.remove(audio\_file\_path)  
  
def translate\_file\_local(source\_json\_path, log\_callback):  
 video\_info = {}  
 try:  
 log\_callback("1. Iniciando processo de tradução...")  
 if not source\_json\_path or not os.path.exists(source\_json\_path): raise FileNotFoundError("Arquivo JSON de origem não encontrado.")  
   
 log\_callback("2. Lendo metadados...")  
 with open(source\_json\_path, 'r', encoding='utf-8') as f:  
 video\_info = json.load(f)  
   
 source\_lang, original\_content = video\_info.get("source\_language", "auto"), video\_info.get("content")  
 log\_callback(f"3. Idioma de origem detectado: {source\_lang}.")  
 if source\_lang == 'pt':  
 log\_callback("4. Texto já está em PT-BR. Tradução não necessária.")  
 return True, "Texto já em Português.", {'json\_path': source\_json\_path, 'video\_info': video\_info}  
  
 if not original\_content: raise ValueError("Arquivo de origem está vazio.")  
 log\_callback("4. Traduzindo conteúdo...")  
 translated\_content = GoogleTranslator(source=source\_lang, target='pt').translate(original\_content)  
 log\_callback("5. Tradução concluída.")  
   
 translated\_filename = os.path.basename(source\_json\_path).replace("\_original", "\_traduzido")  
 log\_callback(f"6. Salvando arquivos traduzidos...")  
 txt\_path = os.path.join('logs', 'transcricao\_traduzida', 'txt', translated\_filename.replace('.json', '.txt'))  
 os.makedirs(os.path.dirname(txt\_path), exist\_ok=True)  
 with open(txt\_path, 'w', encoding='utf-8') as f:  
 f.write(translated\_content)  
   
 video\_info['content'] = translated\_content  
 json\_path = os.path.join('logs', 'transcricao\_traduzida', 'json', translated\_filename)  
 os.makedirs(os.path.dirname(json\_path), exist\_ok=True)  
 with open(json\_path, 'w', encoding='utf-8') as f:  
 json.dump(video\_info, f, ensure\_ascii=False, indent=4)  
   
 log\_master\_record(video\_info, "translate\_file\_local", "SUCCESS", source\_json\_path, json\_path)  
 return True, "Texto traduzido com sucesso!", {'json\_path': json\_path, 'video\_info': video\_info}  
 except Exception as e:  
 log\_error("translate\_file\_local", "deep-translator", str(e), "Verificar conexão com a internet.")  
 log\_master\_record(video\_info, "translate\_file\_local", "FAIL", source\_json\_path, None)  
 return False, f"Erro na tradução: {e}", None  
   
def generate\_tts\_audio(source\_translated\_json\_path, log\_callback):  
 video\_info = {}  
 try:  
 log\_callback("1. Iniciando geração de Áudio (TTS)...")  
 if not source\_translated\_json\_path or not os.path.exists(source\_translated\_json\_path): raise FileNotFoundError("Arquivo JSON traduzido não encontrado.")  
  
 log\_callback("2. Lendo conteúdo traduzido...")  
 with open(source\_translated\_json\_path, 'r', encoding='utf-8') as f:  
 video\_info = json.load(f)  
 translated\_text = video\_info.get("content")  
 if not translated\_text: raise ValueError("Conteúdo traduzido está vazio.")  
  
 log\_callback("3. Carregando modelo TTS...")  
 tts = TTS(model\_name="tts\_models/pt/cv/vits", progress\_bar=False, gpu=False)  
 log\_callback("4. Gerando áudio MP3...")  
   
 audio\_filename = os.path.basename(source\_translated\_json\_path).replace('.json', '.mp3')  
 audio\_path = os.path.join('logs', 'audio\_da\_traducao', 'mp3')  
 os.makedirs(audio\_path, exist\_ok=True)  
 full\_audio\_path = os.path.join(audio\_path, audio\_filename)  
   
 tts.tts\_to\_file(text=translated\_text, file\_path=full\_audio\_path)  
 log\_callback("5. Geração de áudio concluída.")  
 log\_master\_record(video\_info, "generate\_tts\_audio", "SUCCESS", source\_translated\_json\_path, full\_audio\_path)  
 return True, "Áudio gerado com sucesso!", {'audio\_path': full\_audio\_path, 'video\_info': video\_info}  
 except Exception as e:  
 log\_error("generate\_tts\_audio", "TTS (Coqui-AI)", str(e), "Verificar dependências do TTS.")  
 log\_master\_record(video\_info, "generate\_tts\_audio", "FAIL", source\_translated\_json\_path, None)  
 return False, f"Erro na geração de áudio: {e}", None

## 4.3. Módulo de Interface (main.py)

import customtkinter as ctk  
import tkinter as tk  
import threading  
import queue  
from core\_logic import download\_transcript, transcribe\_audio\_local, translate\_file\_local, generate\_tts\_audio  
  
class App(ctk.CTk):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.title("Youtube Transcript Translator")  
 self.geometry("850x700")  
  
 self.last\_original\_json\_path = None  
 self.last\_translated\_json\_path = None  
 self.log\_queue = queue.Queue()  
  
 self.grid\_columnconfigure(0, weight=1)  
 self.grid\_rowconfigure(2, weight=1)  
  
 self.control\_frame = ctk.CTkFrame(self)  
 self.control\_frame.grid(row=0, column=0, padx=10, pady=10, sticky="ew")  
 self.control\_frame.grid\_columnconfigure(0, weight=1)  
  
 self.link\_entry = ctk.CTkEntry(self.control\_frame, placeholder\_text="Cole o link do vídeo do YouTube aqui...")  
 self.link\_entry.grid(row=0, column=0, columnspan=2, padx=10, pady=10, sticky="ew")  
  
 self.transcript\_button = ctk.CTkButton(self.control\_frame, text="Baixar Transcrição", command=lambda: self.start\_task(download\_transcript))  
 self.transcript\_button.grid(row=1, column=0, padx=5, pady=5, sticky="ew")  
  
 self.transcribe\_audio\_button = ctk.CTkButton(self.control\_frame, text="Transcrição via Áudio", command=lambda: self.start\_task(transcribe\_audio\_local))  
 self.transcribe\_audio\_button.grid(row=1, column=1, padx=5, pady=5, sticky="ew")  
  
 self.translate\_button = ctk.CTkButton(self.control\_frame, text="Traduzir para Português", command=self.start\_translate\_task, state="disabled")  
 self.translate\_button.grid(row=2, column=0, padx=5, pady=5, sticky="ew")  
   
 self.tts\_button = ctk.CTkButton(self.control\_frame, text="Gerar Áudio (TTS)", command=self.start\_tts\_task, state="disabled")  
 self.tts\_button.grid(row=2, column=1, padx=5, pady=5, sticky="ew")  
  
 self.log\_frame = ctk.CTkFrame(self)  
 self.log\_frame.grid(row=2, column=0, padx=10, pady=10, sticky="nsew")  
 self.log\_frame.grid\_rowconfigure(0, weight=1)  
 self.log\_frame.grid\_columnconfigure(0, weight=1)  
  
 self.log\_textbox = ctk.CTkTextbox(self.log\_frame, state="disabled", wrap="word")  
 self.log\_textbox.grid(row=0, column=0, sticky="nsew")  
   
 self.process\_log\_queue()  
  
 def log(self, message):  
 self.log\_queue.put(message)  
  
 def process\_log\_queue(self):  
 try:  
 while True:  
 message = self.log\_queue.get\_nowait()  
 self.log\_textbox.configure(state="normal")  
 self.log\_textbox.insert("end", f"{message}\n")  
 self.log\_textbox.configure(state="disabled")  
 self.log\_textbox.see("end")  
 except queue.Empty:  
 pass  
 finally:  
 self.after(100, self.process\_log\_queue)  
  
 def start\_task(self, target\_function):  
 self.reset\_state()  
 url = self.link\_entry.get()  
 if not url:  
 self.log("!! ERRO: Por favor, insira um link do YouTube.")  
 return  
 task\_name\_map = {"download\_transcript": "Baixando Transcrição", "transcribe\_audio\_local": "Transcrevendo via Áudio"}  
 self.log(f"--- Iniciando Tarefa: {task\_name\_map.get(target\_function.\_\_name\_\_, 'Desconhecida')} ---")  
 threading.Thread(target=self.run\_task, args=(target\_function, url)).start()  
  
 def start\_translate\_task(self):  
 if not self.last\_original\_json\_path:  
 self.log("!! ERRO: Nenhuma transcrição original para traduzir.")  
 return  
 self.log("--- Iniciando Tarefa: Traduzindo para Português ---")  
 self.run\_task(translate\_file\_local, self.last\_original\_json\_path)  
  
 def start\_tts\_task(self):  
 if not self.last\_translated\_json\_path:  
 self.log("!! ERRO: Nenhum arquivo traduzido para gerar áudio.")  
 return  
 self.log("--- Iniciando Tarefa: Gerando o Áudio (TTS) ---")  
 self.run\_task(generate\_tts\_audio, self.last\_translated\_json\_path)  
   
 def run\_task(self, target\_function, \*args):  
 self.update\_buttons\_state("disabled")  
 success, message, result\_data = target\_function(\*args, log\_callback=self.log)  
 if success:  
 self.log(f"++ SUCESSO: {message}")  
 if target\_function in [download\_transcript, transcribe\_audio\_local]:  
 self.last\_original\_json\_path = result\_data.get('json\_path')  
 elif target\_function == translate\_file\_local:  
 self.last\_translated\_json\_path = result\_data.get('json\_path')  
 else:  
 self.log(f"!! ERRO: {message}")  
 self.log("--- Tarefa Finalizada ---\n")  
 self.update\_buttons\_state("normal")  
   
 def reset\_state(self):  
 self.log\_textbox.configure(state="normal")  
 self.log\_textbox.delete("1.0", "end")  
 self.log\_textbox.configure(state="disabled")  
 self.last\_original\_json\_path, self.last\_translated\_json\_path = None, None  
 self.update\_buttons\_state("normal")  
  
 def update\_buttons\_state(self, state):  
 is\_running = state == "disabled"  
 self.transcript\_button.configure(state=state)  
 self.transcribe\_audio\_button.configure(state=state)  
 self.translate\_button.configure(state="disabled" if is\_running else "normal" if self.last\_original\_json\_path else "disabled")  
 self.tts\_button.configure(state="disabled" if is\_running else "normal" if self.last\_translated\_json\_path else "disabled")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 ctk.set\_appearance\_mode("System")  
 ctk.set\_default\_color\_theme("blue")  
 app = App()  
 app.mainloop()

# 5. Execução e Teste

1. **Ação:** Executar a aplicação.
   * **Lembrete:** cd para a pasta do projeto, **conda activate env**, e **python main.py**.

# 6. Base de Conhecimento do Projeto

## 6.1. Diretrizes Chave do Projeto (Refinamento das Solicitações)

* **Escolha de Tecnologia:** Python 3.11, Conda, CustomTkinter.
* **Autonomia do Código:** Priorizar bibliotecas locais (Whisper, TTS) sobre APIs.
* **Estrutura de Arquivos:** Categorizar resultados por tipo de conteúdo, não por vídeo.
* **Nomenclatura:** Usar padrão [Data]\_[ID]\_[Sufixo].ext, com sufixos claros.
* **Lógica de Tradução:** Detectar o idioma original e evitar traduções desnecessárias.
* **Feedback Visual:** A interface deve ter um log em tempo real, detalhando cada etapa do processo com espaçamento adequado entre tarefas.
* **Logging Estruturado:** Manter um master\_log.csv para sucessos e um system\_errors.csv para falhas.

## 6.2. Log de Erros Histórico (system\_errors.csv)

1. **Ação:** Criar o arquivo de log de erros com nosso histórico.
   * **Comando para Executar:**  
     type nul > logs\system\_errors.csv
2. **Ação:** Preencher o log.
   * Abra o arquivo logs\system\_errors.csv.
   * **Conteúdo para Colar:**  
     timestamp;function\_name;library\_used;error\_message;proposed\_solution  
     2025-06-07 20:00:00;instalação;TTS (Coqui-AI);No matching distribution found for TTS;A biblioteca não era compatível com Python 3.12. A solução foi recriar o ambiente com Python 3.11.  
     2025-06-07 20:15:00;transcribe\_audio\_local;yt-dlp/ffmpeg;Postprocessing: audio conversion failed: Encoder not found;O yt-dlp não estava encontrando o executável do FFmpeg. A solução foi mudar a estratégia: baixar o melhor áudio disponível (qualquer formato) e passar diretamente para o Whisper, evitando a conversão para MP3.  
     2025-06-07 20:45:00;transcribe\_audio\_local;whisper;Aplicação travou por mais de 1 hora para um vídeo de 6 min;O modelo "base" do Whisper era muito pesado para a CPU. A solução foi trocar para o modelo "tiny", otimizado para performance.  
     2025-06-07 21:00:00;translate\_file\_local;transformers;Helsinki-NLP/opus-mt-en-pt is not a valid model identifier;O modelo Helsinki-NLP requeria uma dependência de tokenização não instalada. A solução foi instalar 'sentencepiece'.  
     2025-06-07 21:15:00;translate\_file\_local;transformers;Falha silenciosa na tradução (texto não era traduzido);O modelo 't5-base' não se comportou como esperado. A solução final foi trocar para a biblioteca 'deep-translator', que é mais leve e estável.