

Voice Translator - Project IOM

Raicu Bianca Elena, Rusen Andreea Emela,
Oteșanu Alexandra Maria

1. Introducere și Obiective

1.1 Scopul proiectului

Scopul acestui proiect este dezvoltarea unei aplicații practice de traducere vocală în timp real, concepută să funcționeze ca o interfață modernă și adaptivă, utilizând framework-ul CustomTkinter pentru a asigura o estetică de ultimă generație și o ergonomie vizuală sporită. Ne-am propus să creăm un sistem capabil să asculte vorbirea naturală, să o transforme în text și să o traducă automat într-o altă limbă, oferind la final și o variantă audio a traducerii. Proiectul pune accent pe integrarea fluidă a acestor etape, astfel încât utilizatorul să poată comunica rapid fără a avea nevoie de cunoștințe tehnice despre modelele de inteligență artificială care rulează în fundal. Prin această aplicație, am urmărit să demonstrăm cum conceptele învățate la laboratorul de Interfață Om-Mașină pot fi utilizate pentru a transforma un computer într-un instrument de comunicare multilingv.

1.2 Problema Adresată

Aplicația noastră răspunde dificultăților de comunicare care apar atunci când două persoane nu vorbesc aceeași limbă, o problemă frecventă în călătorii, educație sau colaborări internaționale. În mod obișnuit, traducerea manuală a textelor sau folosirea unui dicționar încetinește foarte mult conversația și o face obosită. Proiectul abordează această barieră prin automatizarea procesului de prelucrare a sunetului și a textului. Sistemul elimină necesitatea de a tăsa frazele de mână, permitând utilizatorului să vorbească liber, în timp ce software-ul se ocupă de detectarea limbii, traducerea sensului și generarea unei voci sintetice. Astfel, problema izolării lingvistice este rezolvată printr-un flux continuu de date audio-vizuale care fac interacțiunea dintre oameni mult mai simplă și mai naturală.

1.3 Obiectivele Proiectului

- ❖ Acuratețea Transcrierii: Integrarea unui motor de recunoaștere vocală (Faster-Whisper) capabil să ignore zgomotul de fundal și să detecteze corect limba vorbită.
- ❖ Flexibilitate Lingvistică: Implementarea unui sistem de traducere care să suporte cel puțin 5 limbi de circulație internațională (Română, Engleză, Franceză, Italiană, Spaniolă).
- ❖ Responsivitate (Real-Time): Optimizarea latenței dintre finalizarea înregistrării și redarea traducerii prin utilizarea modelelor cuantizate și a procesării asincrone.
- ❖ Experiența Utilizatorului (UX/UI): Dezvoltarea unei interfețe grafice moderne care suportă personalizarea temelor (Dark/Light Mode) și oferă feedback vizual dinamic prin widget-uri customizate, asigurând o interacțiune fluidă între om și sistemul AI.
- ❖ Persistența Datelor: Menținerea unui istoric al sesiunii curente pentru a permite utilizatorului să revină asupra traducerilor anterioare.

2. Împărțirea sarcinilor în echipă

Proiectul a fost dezvoltat printr-o colaborare strânsă între membrii echipei, adoptând o metodologie de lucru modulară. Această abordare ne-a permis să segmentăm aplicația în componente logice clar definite, asigurând astfel o integrare eficientă a tehnologiilor și o distribuție echitabilă a volumului de muncă. Fiecare membru a contribuit cu peste 100 de linii de cod, acoperind arii specifice de dezvoltare, de la arhitectura sistemelor de inteligență artificială până la procesarea semnalelor și designul experienței utilizatorului.

Raicu Bianca Elena (Arhitectura Traducerii și Inteligența Artificială)

- ❖ Configurare interfață CustomTkinter: Implementarea ferestrei principale și a setărilor de sistem (Dark Mode, Color Themes)
- ❖ Arhitectura Modelului AI: Implementarea logicii de traducere prin Limbă Pivot și optimizarea modelelor neuronale
- ❖ Pipeline-ul NLP: Dezvoltarea funcțiilor de procesare text și rafinarea traducerii folosind tehnica Beam Search

Rusen Andreea Emela (Designul Interfeței și Controlul Vizual)

- ❖ Layout Modular și Organizare: Aranjarea elementelor vizuale folosind containere de tip CTkFrame pentru o interfață adaptivă
- ❖ Sistemul de Feedback Vizual: Dezvoltarea funcțiilor de tip „helper” pentru actualizarea automată a statusului și a culorilor elementelor UI
- ❖ Controlul Temelor și Navigarea: Implementarea toggle-ului pentru Dark/Light Mode și configurarea widget-urilor personalizate (butoane, meniuri)

Oteșanu Alexandra Maria (Procesare Audio și Integrare TTS)

- ❖ Managementul Audio (Capture): Configurarea fluxului de captare a sunetului la 16.000 Hz prin sistem de callback (asincron).
- ❖ Integrare TTS (Sinteză Vocală): Implementarea modulului de redare audio via Edge-TTS și gestionarea fișierelor temporare.
- ❖ Persistența Datelor (Istoric): Dezvoltarea sistemului de stocare a sesiunii (dataclasses) și a funcției de Replay din istoric.

3. Arhitectura Sistemului și Tehnologii

Aplicația se bazează pe un pipeline de procesare a semnalelor și a datelor lingvistice, integrând trei tehnologii majore care colaborează pentru a oferi o experiență de utilizare fluidă.

3.1 Speech-to-Text (STT) cu Faster-Whisper

Procesul de transformare a vocii în text reprezintă prima etapă critică a aplicației și se bazează pe modelul Faster-Whisper, o implementare optimizată care permite recunoașterea vorbirii cu o latență minimă chiar și pe configurații hardware fără accelerare grafică dedicată.

Acest modul integrează o funcție de detecție automată a limbii care analizează proprietățile spectrale ale primelor secunde de semnal audio pentru a identifica limba vorbită, eliminând astfel necesitatea unei selecții manuale din partea utilizatorului. Pentru a asigura o transcriere de înaltă precizie, am configurat algoritmul de căutare de tip „beam search” cu o lățime de 5 variante, permitând modelului să evaluateze mai multe ipoteze gramaticale în paralel și să o selecteze pe cea mai coerentă. În plus, utilizarea filtrului VAD (Voice Activity Detection) permite sistemului să ignore zgomotele ambientale sau perioadele de liniște, prevenind astfel generarea unor fragmente de text eronate și optimizând resursele de calcul prin procesarea strictă a segmentelor de vorbire activă. Pentru a asigura o latență scăzută, modelul Faster-Whisper este încărcat pe un fir de execuție separat (thread), permitând interfeței grafice CustomTkinter să rămână activă și să afișeze animații de tip status în timp ce vocea este procesată.

3.2 Text-to-Speech (TTS) cu Edge-TTS

Sinteza vocală constituie modulul de ieșire al sistemului, având rolul de a converti rezultatul traducerii într-o replică audio care imită natural intonația umană. Tehnologia Edge-TTS utilizează rețele neuronale avansate pentru a genera o voce fluidă, superioară sistemelor clasice de sinteză care sunau adesea robotic sau sacadat. Pentru a obține o experiență de utilizare autentică, am implementat un mecanism de mapare lingvistică ce asociază fiecărui cod de limbă de destinație o identitate vocală specifică, asigurând astfel că accentele și pronunția sunt corecte pentru fiecare regiune în parte. Întreg procesul de generare se desfășoară asincron prin utilizarea bibliotecii asyncio, ceea ce permite aplicației să comunice cu serverele de sinteză vocală fără a întrerupe activitatea interfeței grafice sau a bloca interacțiunea utilizatorului cu restul comenzilor disponibile. Sinteza vocală este declanșată printr-un fir de execuție separat pentru a nu bloca animațiile interfeței grafice moderne (CTK).

3.3 Prelucrarea Semnalelor și Managementul Audio

Gestionarea fluxului audio brut reprezintă fundamentalul tehnic al aplicației, asigurând captarea și redarea sunetului la standarde profesionale de fidelitate. Am stabilit frecvența de eșantionare a semnalului la 16.000 Hz pe un singur canal, acesta fiind formatul optim cerut de modelele de inteligență artificială pentru a extrage corect caracteristicile vocale fără a supraîncărca memoria sistemului cu date redundante. Înregistrarea se realizează printr-un sistem de „callback” care colectează continuu fragmente mici de date audio într-un buffer temporar, permitând aplicației să rămână responsivă în timp ce microfonul este activ. Un aspect esențial al managementului audio este reprezentat de mecanismul de siguranță care integrează utilitarul FFmpeg pentru conversia dinamică a formatelor; acesta intervene automat pentru a repară sau re-encode fișierele audio generate de motorul TTS atunci când apar incompatibilități cu placa de sunet a sistemului, garantând astfel succesul redării sonore în orice context de utilizare. Captura audio este gestionată asincron, starea microfonului fiind reflectată vizual prin schimbarea dinamică a culorii butonului de înregistrare (din albastru în roșu).

4. Implementare Tehnică și Design

4.1 Ingineria NLP și Modelele AI (Raicu Bianca Elena)

Această secțiune detaliază „creierul” aplicației, fiind responsabilă pentru încărcarea modelelor neuronale și procesarea lingvistică a textului.

1. Inițializarea și Optimizarea Modelelor

Primul pas tehnic a fost implementarea funcției `load_models()`. Pentru a asigura un echilibru între performanță și consumul de resurse, am ales modelul Faster-Whisper (configurat pe modul accurate sau fast). Acesta utilizează parametrul `compute_type="int8"`, o tehnică de cuantizare care reduce amprenta asupra memoriei RAM fără a degrada calitatea transcrierii. Pentru traducerea propriu-zisă, am integrat modele de tip Transformer din familia MarianMT (Helsinki-NLP), recunoscute pentru eficiența lor în traducerea neuronală.

```
def load_models(mode="accurate"):
    global tok_mul_en, mod_mul_en, tok_en_mul, mod_en_mul, whisper_model

    tok_mul_en = MarianTokenizer.from_pretrained(MODEL_MUL_EN)
    mod_mul_en = MarianMTModel.from_pretrained(MODEL_MUL_EN)
    tok_en_mul = MarianTokenizer.from_pretrained(MODEL_EN_MUL)
    mod_en_mul = MarianMTModel.from_pretrained(MODEL_EN_MUL)

    mod_mul_en.eval()
    mod_en_mul.eval()

    whisper_model_name = "tiny" if mode == "fast" else "small"
    whisper_model = WhisperModel(
        whisper_model_name,
        device="cpu",
        compute_type="int8"
    )
```

2. Implementarea Strategiei „Limbă Pivot”

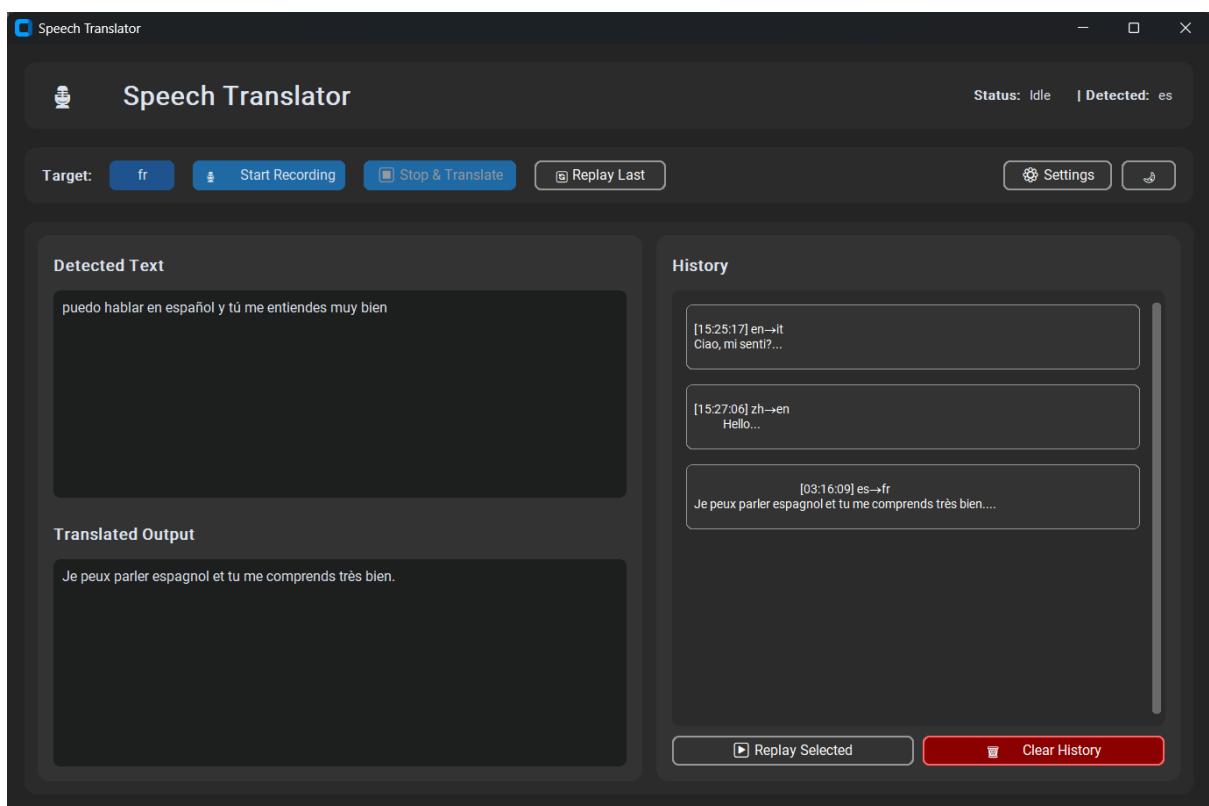
O provocare majoră în dezvoltarea acestui proiect a fost asigurarea suportului pentru o gamă largă de limbi fără a compromite resursele sistemului prin încărcarea unor modele dedicate pentru fiecare pereche lingvistică în parte. Soluția tehnică aleasă a fost implementarea unei logici de traducere în cascădă, utilizând limba engleză ca intermediar sau „pivot”. Astfel, dacă sistemul detectează o limbă sursă diferită de engleză, textul este procesat mai întâi printr-un model multi-lingv (mul-en) pentru a fi adus la o formă comună. Ulterior, acest text intermedier este tradus către limba țintă selectată de utilizator prin intermediul modelului en-mul. Această metodă oferă o flexibilitate maximă aplicației, permitând realizarea unor traduceri complexe între limbi care nu au o conexiune directă în baza de date a modelelor (de exemplu, o conversie fluidă din Spaniolă în Franceză).

```
#Dacă limba engleză nu este limba sursă, traducem mai întâi în engleză
en_text = text if source_lang == "en" else _translate_mul_to_en(text, source_lang)

#Dacă limba engleză nu este limba țintă, traducem din engleză în limba țintă
if target_lang == "en":
    return en_text

return _translate_en_to_mul(en_text, target_lang)
```

Practic, prima parte a codului "forțează" textul să devină Engleză (indiferent de sursă), iar a doua parte decide dacă îl lasă aşa sau îl traduce mai departe.



3. Rafinarea Traducerii prin Parametri de Generare

Pentru a obține traduceri naturale și a evita erorile specifice modelelor IA, am configurat procesul de generare folosind parametri de optimizare. Am implementat tehnica Beam Search cu un prag de 5 variante, permitând modelului să analizeze mai multe structuri de fraze în paralel și să o selecteze pe cea mai corectă gramatical. În plus, am aplicat o penalizare a repetiției (2.5) pentru a preveni erorile de tip buclă (repetarea obsesivă a aceluiași cuvânt), asigurând astfel fluiditatea și coerenta textului final.

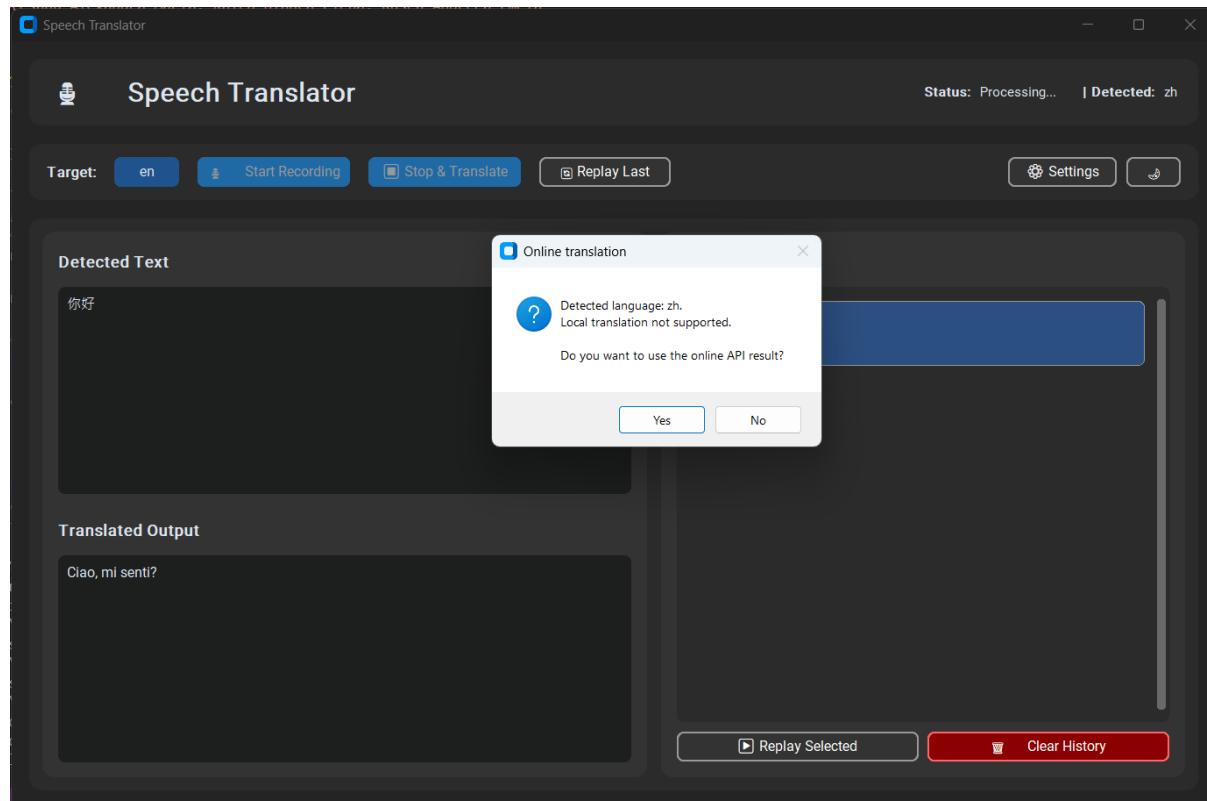
```

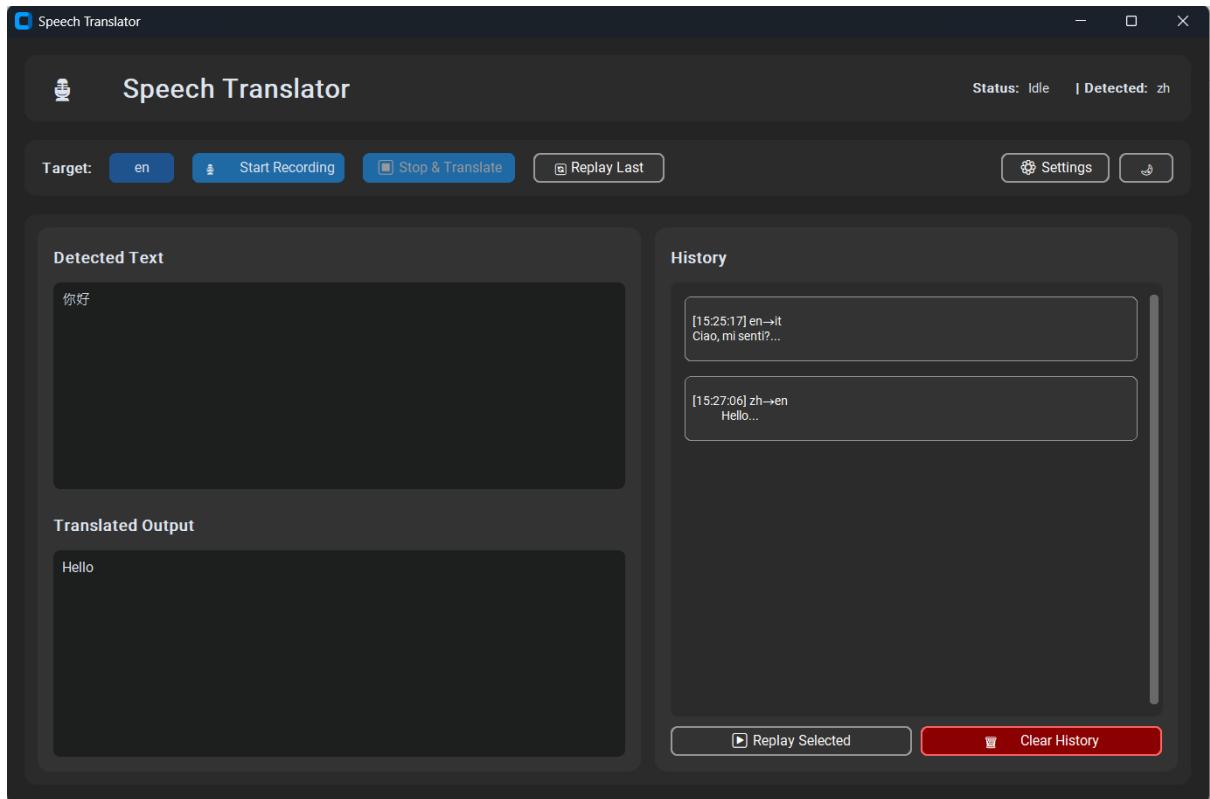
def _translate_mul_to_en(text: str, detected_lang: str) -> str:
    inputs = tok_mul_en([text], return_tensors="pt", padding=True)
    out = mod_mul_en.generate(
        **inputs,
        max_length=256,
        num_beams=5,
        repetition_penalty=2.5,
        no_repeat_ngram_size=2,
        early_stopping=True
    )
    return tok_mul_en.decode(out[0], skip_special_tokens=True)

```

4. Gestionarea Exceptiilor și a Limbilor Nesuportate

Pentru a asigura stabilitatea și versatilitatea sistemului, am evoluat de la un simplu modul de filtrare la o arhitectură de procesare hibridă. Deoarece modelele MarianMT rulate local sunt optimizate pentru un set specific de limbi, am implementat o logică de „Fallback API” care intervine automat atunci când limba detectată de Whisper nu se află în lista celor suportate local (ex. Chineză, Germană sau Turcă). Astfel, în loc să afișeze o eroare sau să blocheze execuția, aplicația redirecționează cererea către un motor de traducere extern (via Google Translate API). Această soluție tehnică asigură o experiență de utilizare continuă și fluidă, transformând limitările resurselor locale într-un sistem global capabil să gestioneze orice pereche lingvistică, menținând în același timp viteza ridicată și confidențialitatea pentru limbile principale prin procesare locală.





```
if used_fallback:
    # Întrebăm utilizatorul
    use_api = messagebox.askyesno(
        "Online translation",
        f"Detected language: {detected_lang}.\\nLocal translation not supported.\\n\\n"
        "Do you want to use the online API result?"
    )
```

4.2 Procesarea Semnalelor și TTS (Oteșanu Alexandra Maria)

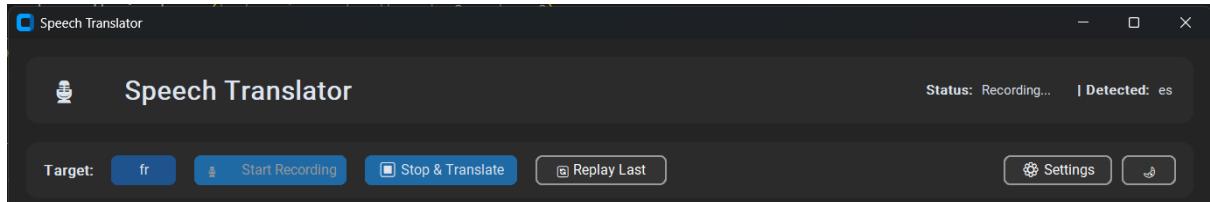
1. Managementul Semnalului și Tehnica de Callback

Fundamentul tehnic al aplicației îl reprezintă sistemul de captare audio, configurat la o frecvență de eşantionare de 16.000 Hz pe un singur canal (mono), formatul standard cerut de modelele de inteligență artificială pentru o precizie maximă. Am implementat o funcție de tip callback care rulează asincron în fundal; aceasta colectează fragmentele de sunet direct într-un buffer de memorie (audio_buffer) doar în momentele în care utilizatorul activează înregistrarea. Această abordare garantează că interfața grafică rămâne fluidă și responsivă, eliminând riscul de „înghețare” a ferestrei în timpul captării datelor audio.

```
def audio_callback(indata, frames, time, status):
    if recording:
        audio_buffer.append(indata.copy())
```

```
fs_local = 16000
```

```
stream = sd.InputStream(
    samplerate=fs,
    channels=1,
    callback=audio_callback
)
stream.start()
```



2. Sinteză Vocală (TTS) și Gestiunea Fișierelor Temporare

Pentru redarea audio a traducerilor, am integrat tehnologia Edge-TTS, care utilizează rețele neuronale pentru a genera o voce fluidă cu intonație naturală. Am dezvoltat un mecanism de gestiune a fișierelor care asigură o utilizare minimă a spațiului de stocare: aplicația generează un fișier .wav temporar pentru fiecare traducere, îl redă prin placă de sunet și, folosind un bloc de siguranță try...finally, îl șterge automat imediat după terminarea redării. Acest proces de tip „Cleanup” previne acumularea de date inutile pe hard disk-ul utilizatorului.

```
def edge_speak_blocking(text, voice, rate=0, volume=0):
    tmp = tempfile.NamedTemporaryFile(delete=False, suffix='.wav')
    tmp_name = tmp.name
    tmp.close()
    fs_local = 16000
    try:
        try:
            asyncio.run(_edge_synthesize_to_wav(text, voice, tmp_name, rate, volume))
        except RuntimeError:
            loop = asyncio.new_event_loop()
            asyncio.set_event_loop(loop)
            loop.run_until_complete(edge_tts.Communicate(text, voice).save(tmp_name))
            loop.close()

        with open(tmp_name, 'rb') as f:
            header = f.read(4)

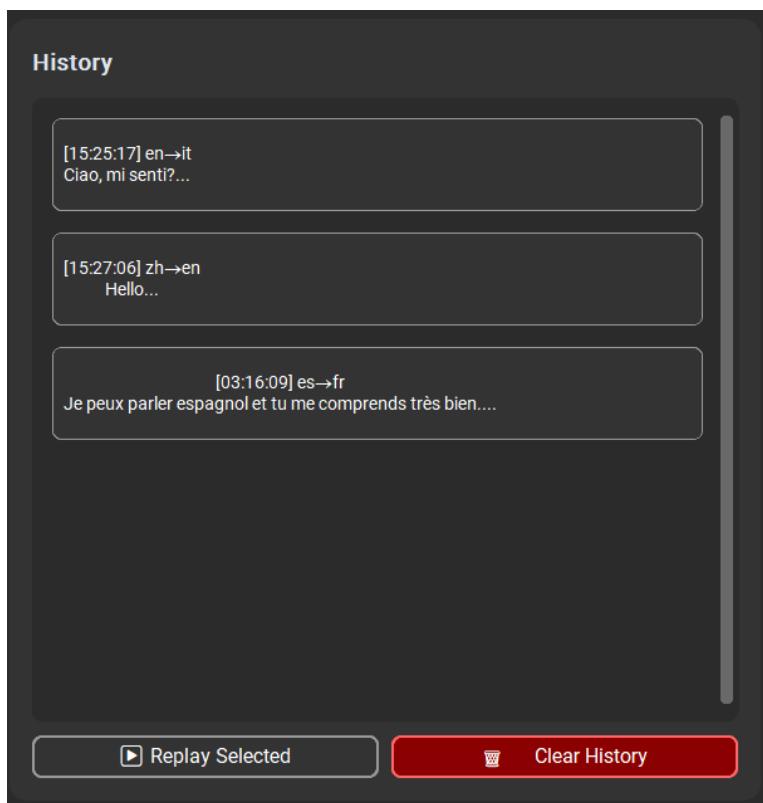
        if header.startswith(b'RIFF'):
            if os.name == 'nt':
                winsound.PlaySound(tmp_name, winsound.SND_FILENAME)
            else:
                data, sr = sf.read(tmp_name)
                sd.play(data, sr)
                sd.wait()
    else:
        conv_tmp = tmp_name + '.conv.wav'
```

```
finally:
    try:
        if os.path.exists(conv_tmp):
            os.remove(conv_tmp)
    except Exception:
        pass
```

3. Persistență Datelor prin Sistemul de Istoric

Pentru a oferi utilizatorului control total asupra sesiunii de lucru, am implementat o structură de date de tip HistoryItem (folosind dataclasses). Aceasta stochează organizat textul original, traducerea și identitatea vocii utilizate. O funcționalitate cheie este sistemul de Replay, care permite reascultarea traducerilor salvate. Pentru a nu îintrerupe experiența utilizatorului, redarea din istoric este lansată pe un fir de execuție separat (threading), permitând aplicației să rămână complet interactivă în timp ce sunetul este redat.

```
@dataclass
class HistoryItem:
    ts: str
    src_lang: str
    tgt_lang: str
    original: str
    translated: str
    voice: str
```



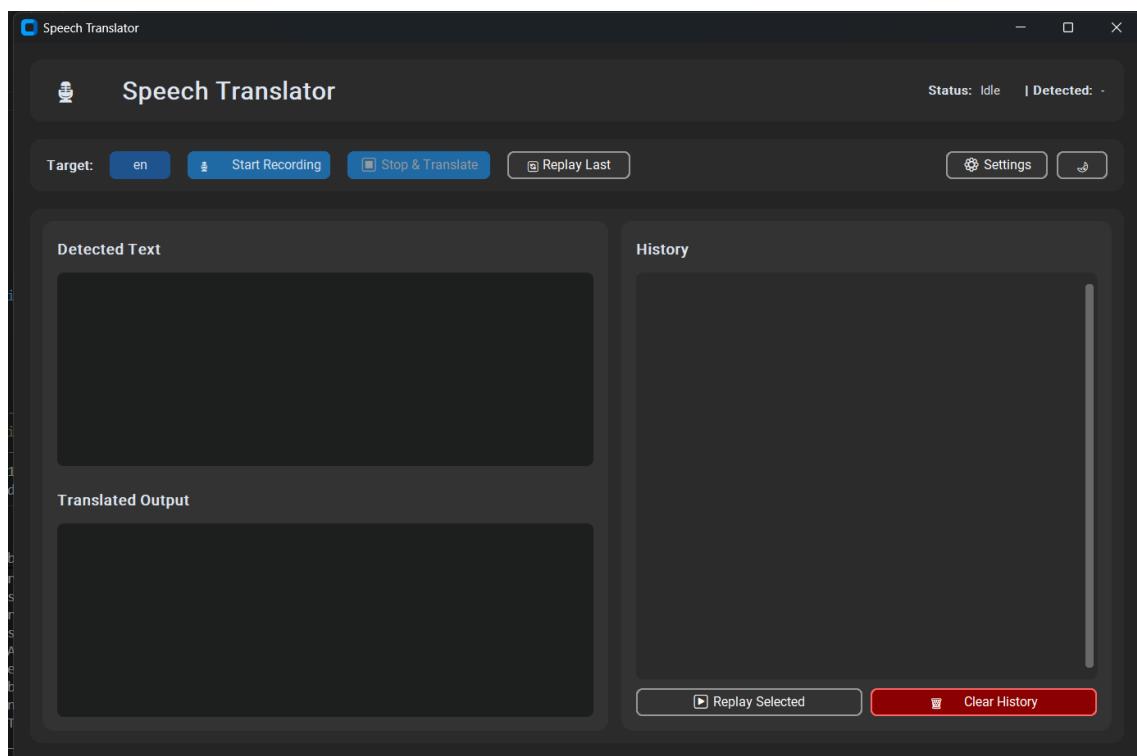
4.3 Arhitectura Vizuală și Logica de Control (Rusen Andreea Emela)

1. Layout Modular și Organizarea pe Cadre (ttk.Frame)

Am proiectat interfața utilizând o abordare orientată pe obiecte, structurând aplicația în containere specializate de tip CTkFrame. Această metodă ne permite să separăm logic zonele de interacțiune:

- ❖ Sidebar (Panoul Lateral): Gestionează setările de temă și controlul general.
- ❖ Main Console: Zona centrală unde sunt afișate textbox-urile pentru textul sursă și traducere.
- ❖ History Sidebar: Un panou dedicat în partea dreaptă pentru vizualizarea și reascoltarea traducerilor anterioare.

```
def _build_main_content(self):  
    """Zona principală cu text și istoric"""  
    main = ctk.CTkFrame(self, corner_radius=12)  
    main.grid(row=2, column=0, padx=16, pady=(8, 16), sticky="nsew")  
    main.grid_columnconfigure(0, weight=2)  
    main.grid_columnconfigure(1, weight=1)  
    main.grid_rowconfigure(0, weight=1)  
  
    # Left side - Text areas  
    left = ctk.CTkFrame(main, corner_radius=12)  
    left.grid(row=0, column=0, padx=(12, 6), pady=12, sticky="nsew")  
    left.grid_columnconfigure(0, weight=1)  
    left.grid_rowconfigure(1, weight=1)  
    left.grid_rowconfigure(3, weight=1)
```



2. Funcții Helper pentru Automatizarea UI

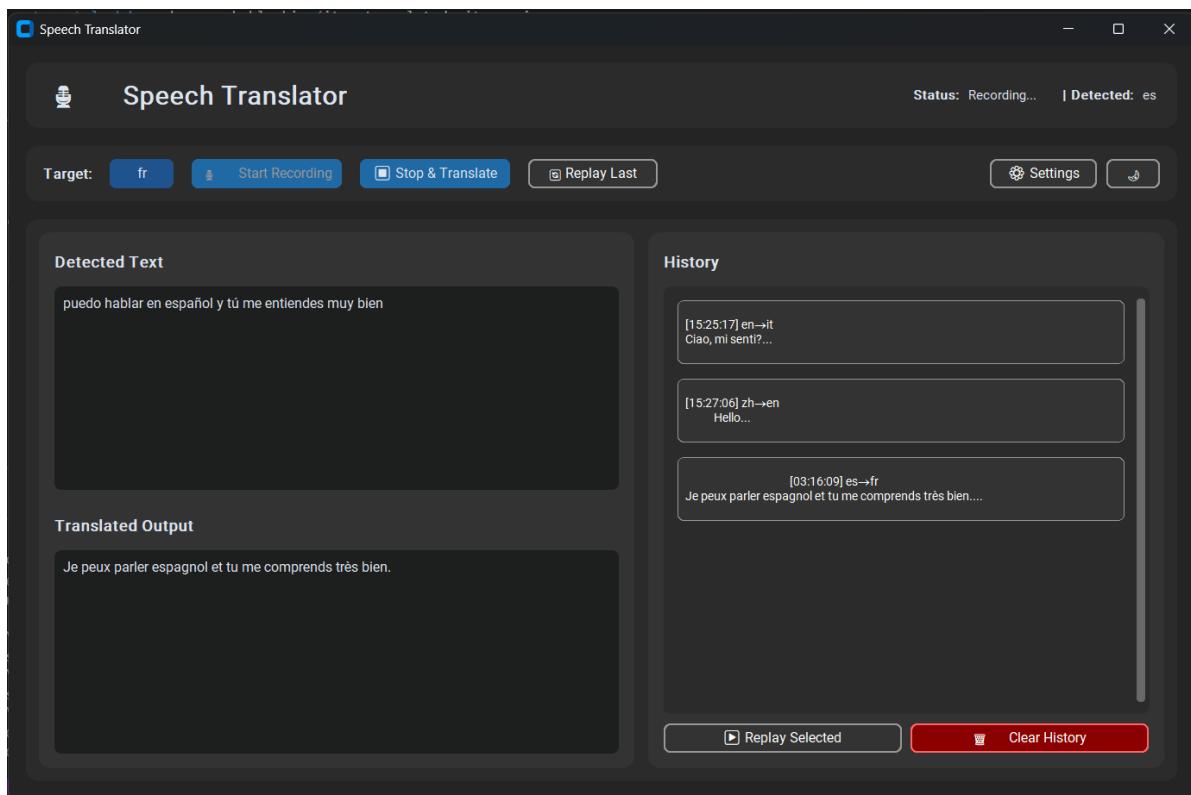
Pentru a menține codul curat și a evita redundanță, am creat funcții de tip „helper” care centralizează actualizarea interfeței. Acestea asigură că orice schimbare în starea sistemului (ex: trecerea de la înregistrare la procesare) este reflectată instantaneu în interfață prin modificarea textelor și culorilor label-urilor.

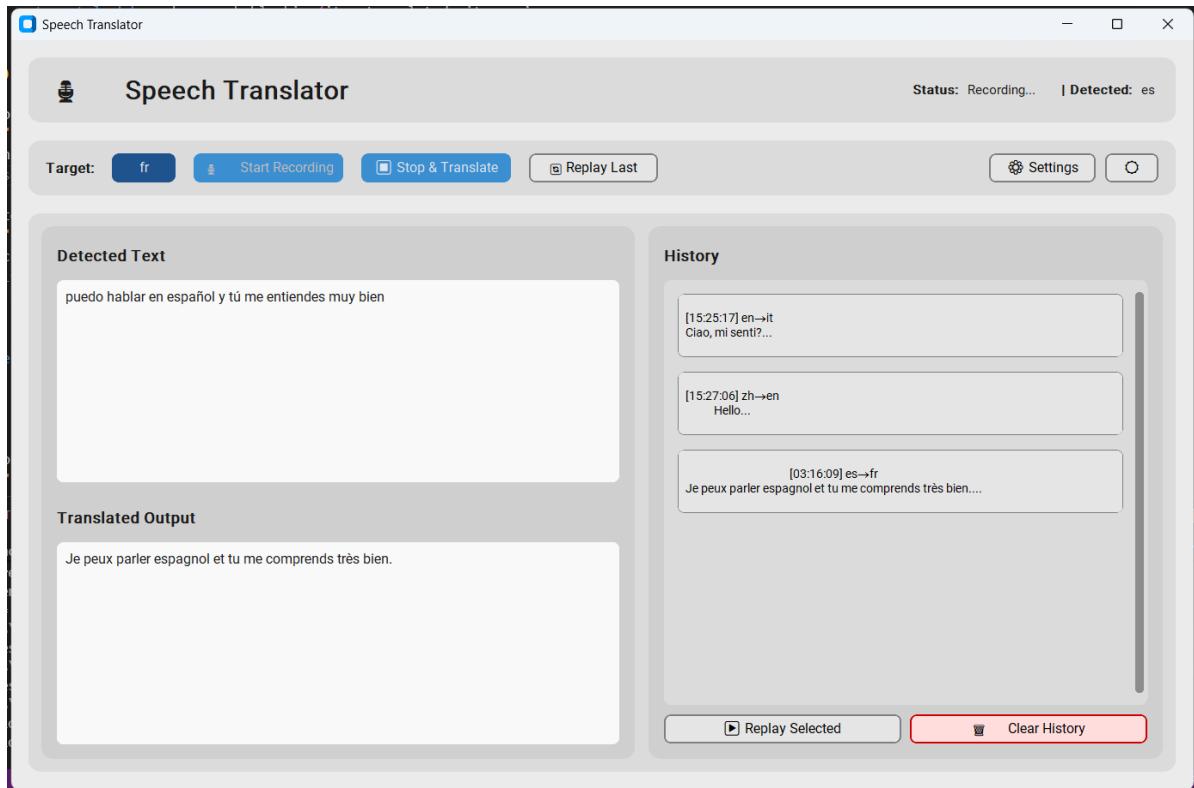
```
def set_status(self, status: str):
    """Actualizează status-ul aplicației"""
    self.status_var.set(status)

def ui_set_text(self, widget: ctk.CTkTextbox, value: str):
    """Setează text în textbox"""
    widget.delete("1.0", "end")
    widget.insert("1.0", value)
```

3. Implementarea Sistemului Adaptiv (Dark/Light Mode)

Am dezvoltat un mecanism de tip toggle care permite schimbarea temei vizuale în timp real. Această funcție nu schimbă doar fundalul, ci și stilul tuturor widget-urilor (butoane, meniuri, textbox-uri), asigurând o lizibilitate optimă indiferent de preferințele utilizatorului.



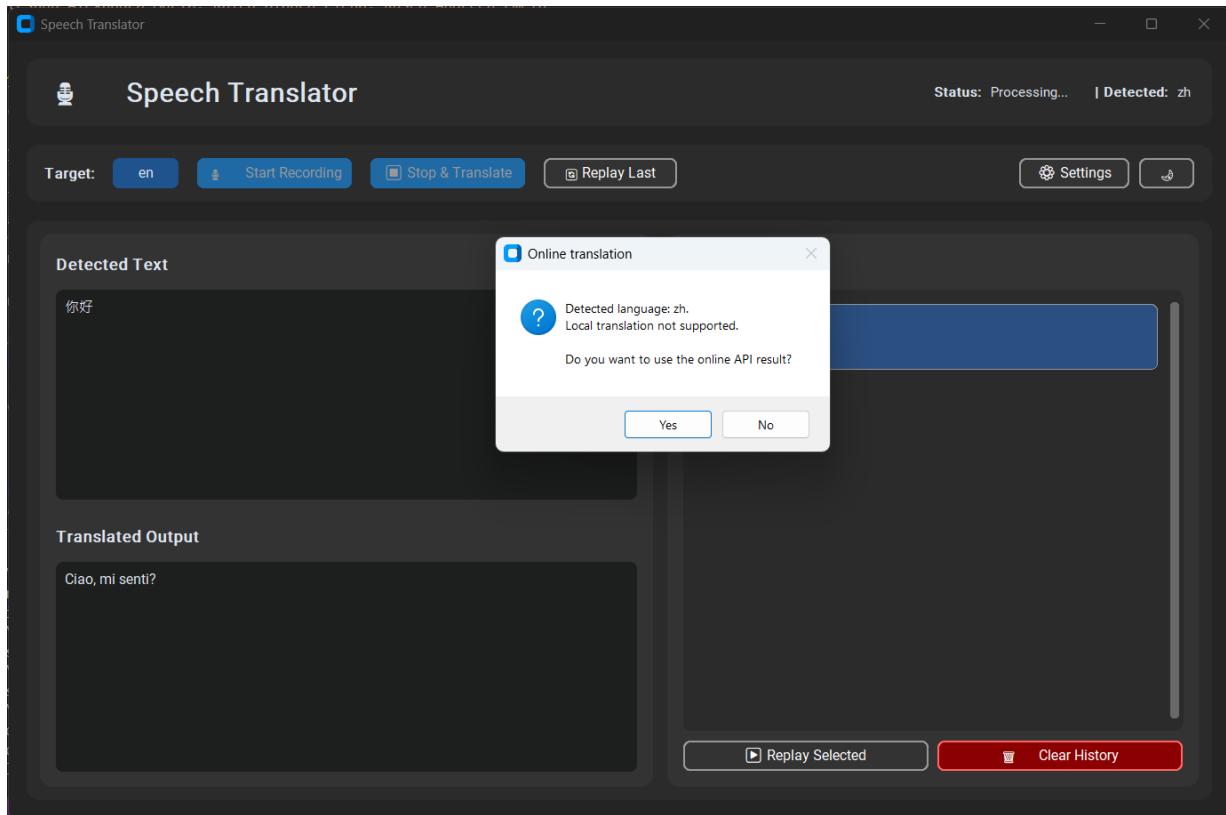


```
def toggle_theme(self):
    """Schimbă între light și dark mode"""
    current = self.appearance_mode.get()
    if current == "dark":
        ctk.set_appearance_mode("light")
        self.appearance_mode.set("light")
        self.theme_btn.configure(text="☀️")
    else:
        ctk.set_appearance_mode("dark")
        self.appearance_mode.set("dark")
        self.theme_btn.configure(text="🌙")
```

5. Gestionaarea Erorilor

5.1. Validarea Limbilor și Prevenirea Traducerilor Eronate

Cea mai importantă eroare gestionată este detectarea unei limbi care nu este prezentă în modelele locale. În loc să oprim aplicația, am implementat o structură logică ce redirecționează textul către un API extern. Acest lucru previne blocarea pipeline-ului și asigură utilizatorului o traducere validă indiferent de context.



```
if used_fallback:
    # Întrebăm utilizatorul
    use_api = messagebox.askyesno(
        "Online translation",
        f"Detected language: {detected_lang}.\nLocal translation not supported.\n\n"
        "Do you want to use the online API result?"
    )
```

5.2. Tratarea Erorilor la Inițializarea Resurselor

Deoarece procesarea audio și redarea TTS sunt sarcini grele, am implementat blocuri try...except în interiorul thread-urilor. De exemplu, în funcția edge_speak_blocking, am tratat eroarea de RuntimeError care apare atunci când un "event loop" este deja activ, creând un loop nou pentru a garanta redarea sunetului.

```
try:
    asyncio.run(_edge_synthetize_to_wav(text, voice, tmp_name, rate, volume))
except RuntimeError:
    loop = asyncio.new_event_loop()
    asyncio.set_event_loop(loop)
    loop.run_until_complete(edge_tts.communicate(text, voice).save(tmp_name))
    loop.close()
```

5.3. Siguranță Fluxului de Date (Cleanup)

Pentru a evita coruperea memoriei sau umplerea spațiului de stocare, utilizăm blocul `finally` la finalul fiecărei redări audio. Acest lucru garantează că fișierul temporar .wav este șters chiar dacă redarea a fost întreruptă de o eroare de sistem sau de închiderea bruscă a aplicației.

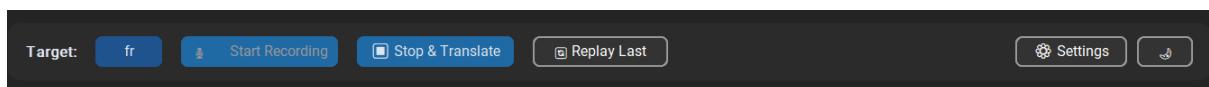
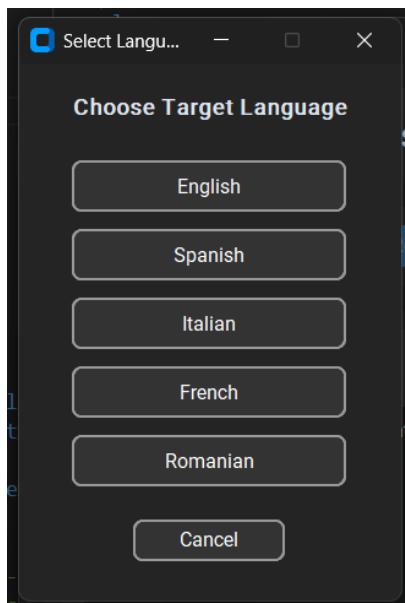
```
try:  
    if os.path.exists(conv_tmp):  
        os.remove(conv_tmp)  
except Exception:  
    pass
```

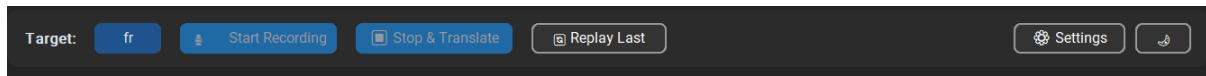
6. Instrucțiuni de Utilizare

6.1. Fluxul Principal de Traducere

Pentru a realiza o traducere, utilizatorul trebuie să urmeze patru pași simpli, marcați prin schimbări vizuale la nivelul butoanelor:

- ❖ Selectarea Limbii: Se alege limba țintă din meniul dropdown (ex: Italiană).
- ❖ Înregistrarea: Se apasă butonul "Start Recording". Acesta își schimbă starea, iar bara de status afișează "Status: Recording...".
- ❖ Procesarea: La apăsarea butonului "Stop + Translate", butoanele devin inactive (disabled) pentru a preveni comenzi multiple, iar statusul devine "Status: Processing (AI)...".
- ❖ Finalizarea: Textul apare în boxele dedicate, iar traducerea este redată automat.

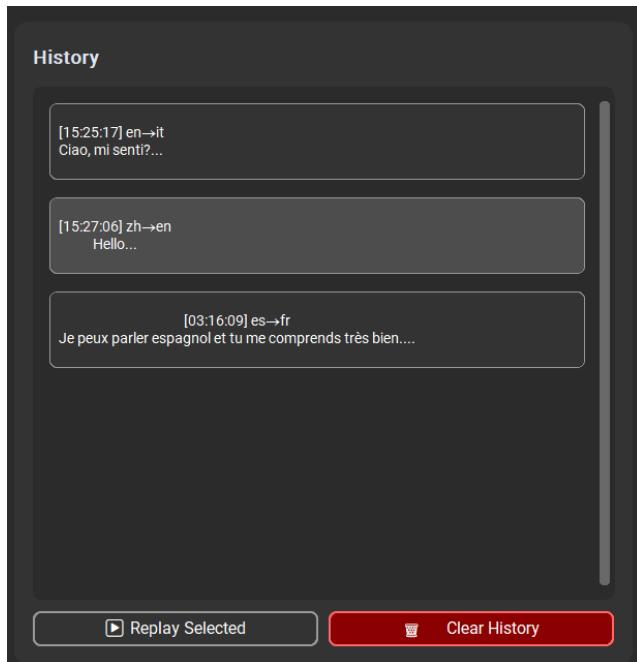




6.2. Gestionarea Iсторичули și Reascultarea (Replay)

Aplicația oferă funcționalități avansate pentru managementul sesiunii curente, localizate în panoul din partea dreaptă a interfeței:

- ❖ Replay Last: Redă instantaneu ultima traducere.
- ❖ Selecție din Listă: Utilizatorul poate face click pe orice intrare din panoul din dreapta.
- ❖ Replay Selected: După selecție, acest buton declanșează un fir de execuție separat care redă sunetul fără a bloca interfața.
- ❖ Schimbarea Temei: Butonul ☰ / ⚡ permite adaptarea vizuală instantanee.



7. Concluzii

Proiectul „AI-Driven Multilingual Voice Translator” demonstrează cu succes integrarea unor modele neuronale complexe într-o interfață modernă și intuitivă. Prin combinarea modelului Faster-Whisper pentru recunoaștere vocală robustă cu arhitectura MarianMT pentru traduceri în cascadă, am reușit să dezvoltăm un sistem care echilibrează performanța locală cu flexibilitatea lingvistică globală.

Dezvoltarea acestei aplicații a evidențiat importanța principiilor de Interfață Om-Mașină (HMI), precum feedback-ul vizual în timp real, procesarea asincronă pentru menținerea responsivității sistemului și designul adaptiv prin teme personalizabile. Rezultatul final nu reprezintă doar o soluție tehnică pentru barierele lingvistice, ci și o bază solidă pentru implementări viitoare în sisteme integrate (embedded) sau tehnologii de asistență mobilă.

8. Bibliografie

- ❖ <https://github.com/openai/whisper>
- ❖ <https://huggingface.co/Helsinki-NLP>
- ❖ <https://github.com/rany2/edge-tts>
- ❖ <https://github.com/SYSTRAN/faster-whisper>
- ❖ <https://python-sounddevice.readthedocs.io/>
- ❖ <https://docs.python.org/3/library/tkinter.html>
- ❖ <https://huggingface.co/docs/transformers/index>
- ❖ <https://ffmpeg.org/documentation.html>
- ❖ <https://customtkinter.tomschimansky.com/>
- ❖ <https://customtkinter.tomschimansky.com/documentation/>