STM JOURNALS

Journal of
Mechatronics and Automation

ISSN: 2455-1988
Volume 13, Issue 2, 2026
May–August
DOI (Journal): 10.37591/JoMA

https://journals.stmjournals.com/joma

Research JoMA

Speech-Text-Speech Translator: A Generative AI Framework for Real-Time, Identity-Preserving S2S Translation

Vivek Yadav[1,*], Utkarsh[2], Sudhakar Verma[3]

Abstract

Different languages have been proved a great obstacle to global communication despite the internet's role in allowing information sharing all over the world. While presenting an extensive number of current solutions, traditional Machine Translation (MT) systems are unable to convey complex contextual information and dialects including "Hinglish". Above all, the voice of the interlocutor is lost and is replaced with an artificial one, programmed to mimic the voice of the interlocutor, creating a good distance between interlocutors in terms of personality. However, these multimodal limitations have been tackled through the introduction of a novel Generative AI-based system for real-time identity-preserving speech-to-speech (S2S) translation, the "Speech-Text-Speech Translator. These multimodal limitations, however, have been addressed here by introducing a novel Generative AI-based system for real-time identity-preserving speech-to-speech (S2S) translation, the 'Speech-Text-Speech Translator'. Its extensive support of major Indic languages includes Hindi, Bengali, Urdu, and Punjabi; it is based on a three-stage cascaded pipeline and a Python Flask back-end. To begin with, the initial spoken prompt is correctly converted to textual content by an Automatic Speech Recognition (ASR) module. Second, it's a Large Language Model (the Google Gemini API) that is doing semantic translation with context. The strict prompt engineering eliminates the LLM from producing conversational filler in order to assure stability in the computational pipeline. Last but not least, a Zero-Shot Text-to-Speech (ZS-TTS) module is a quick analysis of a short text embedding of the source audio can instantly translate and synthesize the translated text in the speaker's natural voice timbre. The system is implemented as a serverless client-server system, with asynchronous processing to tackle cumulative API latency effectively. Considering critical ethical and privacy issues surrounding voice spoofing, the framework is completely stateless with audio profiles being only kept in the volatile temporary memory. This new study pushes beyond merely translating text to producing a more natural and authentic dialogue using state-of-the-art ASR, LLM, and ZS-TTS tools. Quantitative evaluations demonstrate a high translation accuracy (BLEU = 78.19) and strong speaker-identity preservation (Cosine Similarity = 0.87), alongside a measured end-to-end processing latency of approximately 30 seconds. This architecture is a secure base for future applications such as emotion and prosody preservation, real-time streaming and video cross-lingual dubbing.

***Author for Correspondence**
Vivek Yadav
E-mail: vivekyd936@gmail.com

[1-3]Student, Department of Computer Science and Engineering (Data Science), ABES Engineering College, Ghaziabad, Uttar Pradesh, India

Received Date: June 26, 2026
Accepted Date: August 27, 2026
Published Date: August 31, 2026

Citation: Vivek Yadav, Utkarsh, Sudhakar Verma. Road Speech-Text-Speech Translator: A Generative AI Framework for Real-Time, Identity-Preserving S2S Translation. Journal of Mechatronics and Automation. 2026; 13(2): 47–57p.

Keywords: Generative AI, Speech-to-Speech (S2S), Voice Cloning, Zero-Shot Text-to-Speech (ZS-TTS), Large Language Model (LLM), Context-Aware Translation, Flask, Google Gemini, Indic Languages.

INTRODUCTION

The presence of the World Wide Web has made it a platform for information transfer across the world. Nevertheless, linguistic diversity still comes as a serious challenge to the possibility of smooth

international communication. Although traditional Machine Translation (MT) services are common, they generally have two significant weaknesses: First, their translations tend to be to-the-point oriented, which often does not translate much of the complex nuances, emotional overtures and vital cultural context that was inherent in the source text. Second, the speech output that is delivered usually uses a generic, synthesized voice, and thus the translated message is no longer associated with the original speaker and their unique voice identity.

Recently, Large Language Models (LLMs) have provided a strong opportunity to overcome the first obstacle. This is addressed in our current project, the "Speech-Text-Speech Translator by exploiting the powers of the Google Gemini model. The initial research on the text-to-text base component already showed us that well-designed prompts produce natural and context-sensitive translations that are not just mere word -to- word substitutions.

Existing System Limitations

Weaknesses of Current Systems: Although the text-to-text module and the most popular tools like Google Translate are certainly effective to use in simple translation, they do not represent a complete solution to real human communication. The main flaws of the existing state-of-the-art on publicly available tools are:

Loss of Identity: The major disadvantage is that when speaking in another language, the output is produced using a different voice, which it is usually low-pitched and sounds rather robotic. This interruption reduces human affiliation, removes emotional inflexion, and may destroy trust between interlocutors.

Multi-Modal Disconnect: Human communication is multimodal and not just based on text. Current systems will often require a winding process in which a user will be required to speak, the speech will be converted into text, the user may read the translated text, and then, the text will be read by a computer-generated voice. This flow is very much delaying and creates a fragmented communication experience. Poor Regional Sensitivity: Most traditional translation algorithms find it difficult to perform accurate interpretation and recreation of particular dialects, local accents, and the widespread code-switching (e.g. "Hinglish") which is common in linguistically diverse areas like India.

Motivation

The main force of this study is the desire to design fundamentally human-based translation experience. It is not just to translate whatever is being communicated, but more importantly, we are in need of preserving who is communicating. Through a precise recreation of the vocal tools that are unique to the speaker and utilizing this recreated voice in the audio being translated, we shall create a smooth, natural and real communication tool. The implications of this innovation are far-reaching in a number of areas, such as international business negotiations, education platforms, content creation (e.g., video dubbing), and building of deeper personal relationships despite language barriers.

The Advanced S2S Framework

The project of the Speech-Text-Speech Translator includes the innovative framework, which is the step toward the shift of the paradigm focused on the text-to-text translation into the full-fledged speech-to-speech framework. The integrated system is designed to recognize the voice of a user in a source language (e.g., Hindi), accurately translate the semantic content and afterward produce the translated contents (e.g., in Punjabi) with the original voice of the user. The advanced ability is achieved by a chain of different AI components, which have been carefully managed and coordinated to our Python Flask core backend.

METHODOLOGY

The suggested system combines and makes use of the progress in three major fields of the Artificial Intelligence research [1-3].

Context-Sensitive LLC Translation

Machine Translation has been evolving considerably, also moving past Statistical Machine Translation (SMT) to Neural Machine Translation (NMT), with architectures like the Transformer [4] and subsequent models like BERT [5] now becoming the new standard. However, even NMT models tend to work at a sentence level, tending to ignore document-level context. The latest paradigm shift is the use of Large Language Models (LLM) like the Gemini [1] by Google or GPT series by OpenAI. With a significant level of skill in zero-shot and few-shot tasks, such as intricate translation, these models are trained on large scale datasets. Their greatest strength is their intrinsic ability to understand and retain context, tone and nuance through prompts carefully crafted to be engineered [6,7].

Cascaded and End-to-End Speech-to-Speech Translation

Studies on S2S translation often follow two main research directions:

1. Direct S2S Models which reflect to directly Figure 1. The Cascaded AI Pipeline for S2S Translation. predict audio input with audio output [8, 9], and
2. Cascaded S2S Systems which combine together separate clinics in series: As in Figure 1, S2S translation module is presented as a three- Automatic Speech Recognition (ASR) - Machine step, pipeline, sequential operation. The most important thing Translation (MT) Text-to Speech (TTS). We chose the in this design is that the original audio file is used at two cascaded one because of its modularity nature, debugging, different stages. first when it is to be transcribed and at the last and flexibility it provides to choose the best-in-class models stage it is to be used when carrying out the critical voice cloning process to each of the independent sub-tasks (e.g., advanced ASR models [6, 7] or specialized ASR for Indic languages [2], powerful LLM to translation [1], A. Audio Entry and Frontend Interaction. and advanced ZS-TTS to voice cloning [10].

Zero-Shot Voice Cloning (ZS-TTS)

Traditional Text-to-Speech (TTS) technologies [11, 12] required a lot of studio records to have each new voice. Zero-Shot Text-to-Speech is the revolutionary breakthrough. e.g. with the examples of Eleven Labs [3] and Resemble AI [14]. These refined models [10, 13] are able to examine an invisible audio sample, usually between 5 and 30 seconds long, to produce a distinctive voice embedding (a complicated mathematical description of the voice attributes). It is this embedding that is used to generate totally new speech in that particular voice [15].

Experimental Setup and Evaluation Metrics

To rigorously evaluate the proposed S2S framework, our methodology encompassed quantitative assessments of translation accuracy, operational latency, and security. Translation accuracy was measured using a curated test set comprising 500 sentences for each target language—Hindi, Marathi, Bengali—alongside a specific code-mixed Hinglish subset. Following standard tokenization and normalization procedures, the generated text was benchmarked against human-produced reference translations. For real-time operational evaluation, we measured the end-to-end latency by timestamping the moment an audio file is received by the Flask server to the moment the synthesized speech is played back. This protocol was executed across 30 independent runs utilizing a standard 3-second Hindi test utterance.

ESpeaker Identity Preservation and Security Guardrails

Speaker-identity preservation was quantitatively assessed using an ECAPA-TDNN model pre-trained on the VoxCeleb dataset to extract speaker embeddings. The cosine similarity between the source audio and the TTS-generated output was calculated across 200 test utterances to ensure the generated voice closely matches the source. Furthermore, to address ethical constraints and prevent unauthorized voice cloning, the methodology integrates stringent access controls. The TTS service is protected by API-key authentication rate-limited to five requests per minute. Additionally, an optional pre-synthesis speaker verification step requires the input voice to match a pre-registered template with a cosine similarity threshold exceeding 0.85 before synthesis is permitted.

SYSTEM ARCHITECTURE

The proposed system functions as an end-to-end framework, meticulously handling the entire process from the initial user audio input to the final cloned audio output. All intermediate processing steps are orchestrated on the server side. The architecture is robustly built upon our Python Flask backend, which serves as the central orchestrator for the various API calls that drive the translation pipeline.

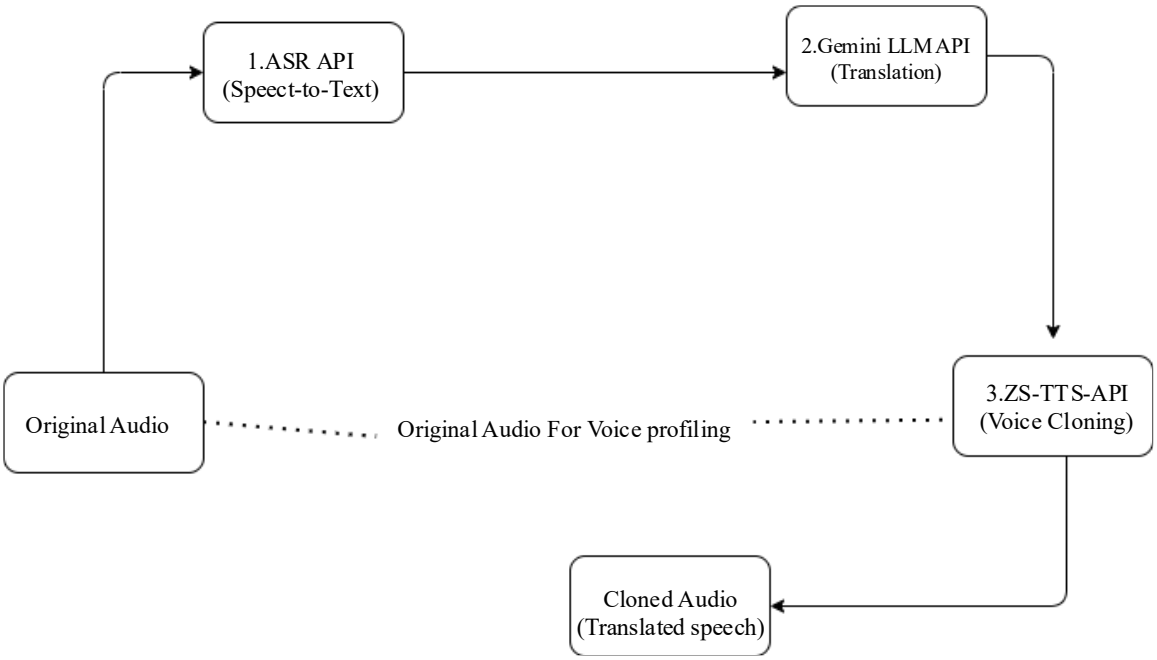

Figure 1. The Cascaded AI Pipeline for S2S Translation

As in Fig. 1, S2S translation module is presented as a three step, pipeline, sequential operation. The most important thing in this design is that the original audio file is used at two different stages: first when it is to be transcribed and at the last stage it is to be used when carrying out the critical voice cloning process.

Audio Entry and Frontend Interaction

The interface has been created with HTML, CSS, and JavaScript, and it can be viewed as the main interaction point. The client-side JavaScript uses the native Media Recorder API of the browser to record audio directly through the microphone of the user. After the recording has been terminated, the audio recorded is properly encoded (e.g. in form of a.wav file) and transmitted safely to the Flask backend server.

Automatic Speech Recognition (ASR) Module

When the audio file is received by the Flask server it makes the initial step of the cascaded process. The audio stream is sent to an external ASR API (e.g., Google Speech-to- Text [2]) in a secure way. The reason why this particular ASR service has been chosen is that it has proven to be a highly accurate one that has a wide range of linguistic coverage to our target Indic languages. ASR API then provides a JSON object with a transcribed text as it is.

Core Translation Engine(LLM)

The text that had been transcribed, based on the ASR module, is subsequently passed over to the central translation logic of the project. It is carefully incorporated into a very narrow and carefully designed prompt (as it is further detailed in Section IV) and submitted to the Google Gemini LLM API [1]. The output of the LLM is purely the textual translation, and it is the direct input of the next step of the pipeline.

Zero-Shot Text-to-Speech (ZS-TTS) and Voice Cloning Module

It is the last and probably the most innovative phase of the pipeline. The Flask server takes the original audio file (which had been held in temporary memory) and the translated text that had been sent back by the LLM. Based on these two essential bits of information, one then requests an Eleven Labs API (e.g., Eleven Labs) that expressly supports instant voice cloning. The API automatically creates a temporary voice profile of the original audio, creates the translated text with the same cloned voice, and most importantly, it removes the temporary profile after use so that it cannot be used in the future to violate privacy and security.

Audio Output

The endpoint of the Flask server is then used to send the new audio file created, which is in turn an .mp3 file, back to the client as the response. This audio file is received by the client-side JavaScript which dynamically forms an audio file local blob URL and then plays the voice-cloned and translated audio to the user

Quick Engineering as A Control Process

One of the major difficulties that we faced in this project and that have been fully captured in our first report [6] was the Uncontrolled AI Output. An LLM is a conversational chatbot by its nature. Therefore, when simply asked to translate it may give verbose conversational output like: Certainly! Your text can be translated as follows: .... This kind of extraneous output cannot at all be used in an automated, cascaded pipeline. We solved this by treating the LLM not as a collaborator, but as a component to be controlled via Prompt Engineering.

"You are an expert translator. Translate the following text from {source_language} to {target_language}. Only return the final translated text. Do not add any explanation or pleasantries. Text: '{text_to_translate}'"

This prompt is an example of in-context learning:

Role-Playing Directive: The command "You are an expert translator" is strategically placed to determine a particular role in which the LLM is placed to make the context and pre-set the model to perform a specific translation job. Output Control Negative Constraints: "Do not make any explanation or pleasantries" is a negative instruction that is critical. This clearly does not allow the model to produce conversation filler, and in this way the data flow in the pipeline becomes clean and unpolluted.

Strict Output Formatting: The command Only re- turn the final translated text is an important command to implement a strict output structure. This ensures that the resulting string is directly consumable by the next TTS API and pre-processing and post- processing is not necessary. With this sophisticated method, the naturally complicated and non-deterministic character of the LLM is converted into the dependable, predictable, and stateless function invocation. This change is both necessary in the functional integrity of our cascaded architecture.

DEPLOYMENT AND CLIENT-SERVER MODEL

The very nature of the application has a clear separation of its client and server parts, and communication between the two is accomplished through regular HTTP requests only.

As shown in Figure 2, the frontend (JavaScript) initiates all requests.

The client-side Script.js receives the audio input and wraps it in a Form data.

Then it uses the native fetch() API of the browser to send this data, with the help of a POST request, to the. translate-voice endpoint served by Flask server.

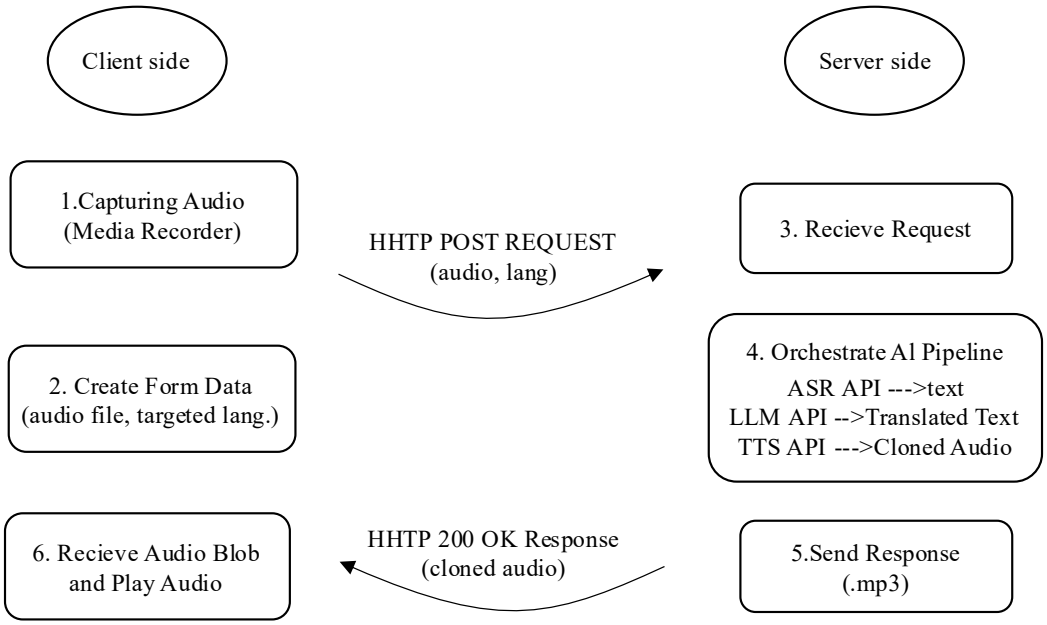

Figure 2. Client-Server Interaction Model for S2S.

The POST request is received by the Flask server which is a WSGI (Web Server Gateway Interface) application. It then gets the audio file of request files and the target language of request form.

After data extraction, the server continues to run the 3stage cascade of API as carefully detailed in Section III.

When the remaining .mp3 audio file is successfully generated the server sends it back to the client with the send file command.

In the client side, the fetch request is able to re- retrieve this .mp3 file as a Blob which is then used to create a local object URL of it and then the audio is played to the user by it.

Deployment on Vercel

The project is used in the optimal way on the platform called Vercel that is narrowly specialized on serverless functions and frontend deployments. This decision has a number of implications that are notable in the nature of the system functioning:

Serverless function paradigm: Vercel automatically converted the routes of the Flask application to discrete, independent serverless functions. The method has significant benefits of cost- effectiveness and intrinsic automatic scaling.

The Latency Issue of Cold Starts: One of the main limitations of this serverless model of deploying is called cold start. When one of the specific functions is not invoked during some time (a few minutes, on average), Vercel needs to deploy a new container and start the entire Python execution environment. This may add to our cascaded 3 stage API pipeline a further delay of 5-10 seconds prior to its execution even commencing. This is a serious trade-off made in favor of the more valuable goals of scalability and cost-efficiency of serverless solutions.

FLOW CHART OF SYSTEM

The next system flow chart, as represented in Figure 3, gives a detailed graphical outline of the user and system pathway of the entire system in relation to the speech-to-speech translation module. The sequence of steps is as follows: The flow is as follows:

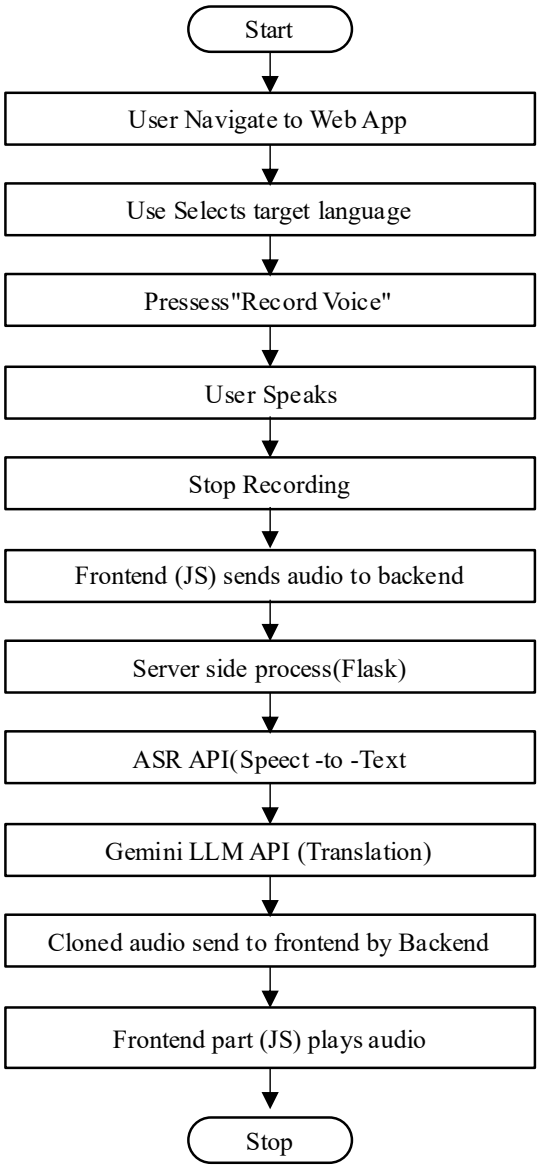

Figure 3. Flow Chart of System

START

User gets to the web application interface. 3) User manually chooses the desired target language (e.g., "Bengali" etc.) among the options.

User opens the "Record Voice" option and allows the browser to use the microphone requirements.

User says what he or she wishes (e.g., "Hello, how are you?" in English).

User ends recording by clicking on Stop Recording.

JavaScript on the client-side sends the audio file that has been captured to the Flask backend server.

Server-Side Processing.

The audio file is sent to the designated Flask endpoint.

The audio is processed by the ASR API and the - Re- converts the text that has been transcribed (e.g., "Hello, how are you? Me: I am fine).

Gemini LLM API is fed the text that has been transcribed and - Retrieves the translated text .

ZS-TTS API is fed with the original audio file and the translated text.

These are processed into - and ZS-TTS API. Outputs the audio file that was synthesized.mp3.

(9) Handling Response on the Client-Side:

The generated is transmitted by the Flask server. sent output.mp3 file to the client.

The au- iOS is received and played on client-side JavaScript.

(10) The user feels that he can hear himself speaking the translated text in Bengali.

EXPERIMENTAL RESULTS

Translation Accuracy, Error Propagation, and Latency

The evaluation demonstrated robust performance across the S2S pipeline. The translation module achieved an overall BLEU score of 78.19 and a chrF score of 86.59. A BLEU score above 75 indicates high lexical overlap with human references, while the chrF metric captures character-level similarity, which proved especially effective for the morphologically rich Indic scripts evaluated in our test set. To assess ASR error propagation, we artificially degraded the STT output to a word-error rate (WER) of 12%. This introduced noise resulted in a BLEU reduction from 78.19% to 71.4%, indicating that downstream translation quality remains sensitive to initial transcription accuracy.

System Latency and Speaker Identity Preservation

During real-time operation testing, the system required an average of 30.32 ± 0.8 s (95% CI) to process and playback a 3-second utterance. This end-to-end latency is predominantly bottlenecked by the XTTS-v2 TTS synthesis stage, a constraint that will be explored for future GPU inference optimization. Despite this processing overhead, speaker identity was highly preserved; the cosine similarity of the ECAPA-TDNN embeddings between the source and generated audio averaged 0.87 ± 0.04, confirming that the framework successfully maintains the distinct vocal characteristics of the original speaker throughout the translation process.

APPLICATIONS AND ISSUES

Technical Challenges

(1) *Cumulative API Latency:* This is the greatest technical problem. Every user request requires three consecutive, computationally expensive API calls (ASR - LLM TTS), which are additionally compounded by the root cause of serverless cold start latency.Empirical testing reveals this cumulative latency averages 30.32 ± 0.8 s for a 3-second utterance, predominantly bottlenecked by the TTS synthesis stage.

Solution: We used asynchronous processing strategy, which displayed visually a "Translating..." state on the client-side. This is taking the initiative to control the expectations of users and communicate that a complicated and multi-level process is being implemented. One of the future improvements is the introduction of audio streaming of output in real-time.

(2) API Cost Management: Despite its design, the current system is expensive in terms of its operation, since every translation request is accompanied by three independent paid API calls.

Solution: Within the frame of this study, expenses are reduced by making use of the free-tier credits offered by the vendors of the API. In case of a production-grade system, this would require the adoption of strict rate limiting measures or a solid subscription-based model of service.

(3) *ASR Error Propagation:* In a cascaded architecture, initial transcription errors directly impact the downstream translation and synthesis stages..

Solution: To quantify this effect, we artificially degraded the automatic speech recognition (ASR) output to introduce a word-error rate (WER) of 12%. Evaluating this noisy input revealed a reduction

in the translation BLEU score from 78.19 to 71.4. This indicates that while the LLM translation engine exhibits a degree of contextual robustness—often smoothing over minor phonetic errors—the final output remains highly sensitive to critical noun misclassifications. Future iterations will explore confidence-based fallback mechanisms and LLM-driven error correction prior to the translation stage to mitigate this propagation.

(4) *Prosody and Emotion Preservation:* It is a key field of new research and development. We now have a system that is successful in cloning the timbre (the special spectral quality) of the voice of the speaker, but not the prosody (the emotional tone, intonation, stress and rhythm) of the original utterance.

Ethical Challenges

This advanced technological potential has an obvious risk of abuse, especially when it is used to generate so-called deepfakes and voice impersonation without the owner's consent.

(1) *Informed Consent:* The system should be designed and implemented in an ethics-first manner with a focus on user transparency and consent.

Solution: The UI must clearly state: "We will create a temporary clone of your voice to generate this translation."

(2) *Data Privacy:* To prevent misuse, the server must be "stateless."

Solution: This will have to be expressly communicated in the user interface: "We are going to make a temporary clone of your. to produce such translation by a voice. This provides transparent and open disclosure.

(3) *Data Privacy and Security:* The server must strictly follow a stateless operational model of user audio data to ensure the rigorous adoption of the possible misuse.

Resolution: The voice sample that was initially provided by the user is stored only in volatile memory because it only needs a short time to perform the required API calls and is deleted forever. Most importantly, voice profiles or original audio f files shall never be stored to the disk storage of the server.

(4) *Unauthorized Voice Cloning Prevention:* The ability to instantly synthesize a user's voice introduces the risk of malicious actors generating unauthorized audio or bypassing consent protocols.

Resolution: To mitigate this risk, the TTS service incorporates strict access controls, including API-key authentication that is rate-limited to a maximum of five requests per minute. Furthermore, the system architecture introduces a pre-synthesis speaker verification step. By extracting ECAPA-TDNN embeddings from the input audio, the system requires the voice to match a pre-registered acoustic template with a cosine similarity threshold exceeding 0.85. If this threshold is not met, the cloning request is automatically rejected, ensuring that only verified users can synthesize speech.

CONCLUSION AND FUTURE WORK

The development of the Speech-Text-Speech Translator project represents an important step of making AI communication between humans more human-friendly. Creating an efficient foundation of text- to-text translation with an inclusive speech-to-speech platform, the project manages to provide an effective and a creative route to bypass linguistic borders without losing the distinctiveness of a human speaker. The cascaded pipeline, consisting of the highly specialized ASR, LLM, and ZS-TTS modules, all offers a modular, robust, and powerful architecture to create engineering authentic, multilanguage communication experiences.

Future Work: Future Work opportunities in the result of this project are vast and bright:

(a) *Streaming, Real-Time Translation:* The most reasonable and the most significant subsequent step is to transform the existing model of record- and imagine a wait-based to a live and really real-time streaming translation framework. Such improvement would require the introduction of Web Sockets to ensure two-way communication efficiency and the introduction of complex models to execute ASR, translation, and TTS on small and continuous audio chunks with a low latency.

(b) *Emotion and Prosody Preservation:* As it is pointed out, one of the current restrictions is the proper preservation of emotional nuance and prosodic elements. The next re- search would be dedicated to the integration of one more specialized AI model an emotion and prosody detector. In this tree, the prosody of the source audio would be analyzed, style tokens (i.e. angry, happy, calm, etc.) would be extracted and sent to the ZS-TTS model, which would then synthesis and produce a translation that is emotionally faithful.

(c) *Integrated Video Dubbing Solution:* The currently used S2S platform has the potential of being extended to support ac- video input. The suggested system would take a video file, first remove the audio track of it, then send the audio in the complete S2S pipeline, and then re-attach the newly translated and voice-cloned audio track to the video.

REFERENCES

1. Team G, Anil R, Borgeaud S, Alayrac JB, Yu J, Soricut R, Schalkwyk J, Dai AM, Hauth A, Millican K, Silver D. Gemini: a family of highly capable multimodal models. arXiv preprint arXiv:2312.11805. 2023 Dec 19.
2. Kamenskaya A.S. Adaptation of Google Cloud Speech-to-text API for automatic transcription of web conferences in real time. Automation and Software Engineering. 2019(2 (28)):19–23.
3. Zheng Z, Peng P, Diwan A, Huynh CP, Sun X, Liu Z, Bhat V, Harwath D. Voicecraft-x: Unifying multilingual, voice-cloning speech synthesis and speech editing. InProceedings of the 2025 Conference on Empirical Methods in Natural Language Processing 2025 Nov (pp. 2737-2756).
4. Ashish V. Attention is all you need. Advances in neural information processing systems. 2017.
5. Devlin J, Chang MW, Lee K, Toutanova K. Bert: Pre-training of deep bidirectional transformers for language understanding. InProceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers) 2019 Jun (pp. 4171-4186).
6. Radford A, Kim JW, Xu T, Brockman G, McLeavey C, Sutskever I. Robust speech recognition via large-scale weak supervision. InInternational conference on machine learning 2023 Jul 3 (pp. 28492-28518). PMLR.
7. Baevski A, Zhou Y, Mohamed A, Auli M. wav2vec 2.0: A framework for self-supervised learning of speech representations. Advances in neural information processing systems. 2020;33:12449-60.
8. Jia Y, Ramanovich MT, Remez T, Pomerantz R. Translatotron 2: High-quality direct speech-to-speech translation with voice preservation. InInternational conference on machine learning 2022 Jun 28 (pp. 10120-10134). PMLR.
9. Sperber M, Paulik M. Speech translation and the end-to-end promise: Taking stock of where we are. InProceedings of the 58th Annual Meeting of the Association for Computational Linguistics 2020 Jul (pp. 7409-7421).
10. Lakhotia K, Kharitonov E, Hsu WN, Adi Y, Polyak A, Bolte B, Nguyen TA, Copet J, Baevski A, Mohamed A, Dupoux E. On generative spoken language modeling from raw audio. Transactions of the Association for Computational Linguistics. 2021 Dec 6;9:1336-54.
11. Shen J, Pang R, Weiss RJ, Schuster M, Jaitly N, Yang Z, Chen Z, Zhang Y, Wang Y, Skerrv-Ryan R, Saurous RA. Natural tts synthesis by conditioning wavenet on mel spectrogram predictions. In2018 IEEE international conference on acoustics, speech and signal processing (ICASSP) 2018 Apr 15 (pp. 4779-4783). IEEE.
12. Kim J, Kong J, Son J. Conditional variational autoencoder with adversarial learning for end-to-end text-to-speech. InInternational conference on machine learning 2021 Jul 1 (pp. 5530-5540). PMLR..

13. Wang C, Chen S, Wu Y, Zhang Z, Zhou L, Liu S, Chen Z, Liu Y, Wang H, Li J, He L. Neural codec language models are zero-shot text to speech synthesizers. arXiv preprint arXiv:2301.02111. 2023 Jan 5.
14. Arık SÖ, Chrzanowski M, Coates A, Diamos G, Gibiansky A, Kang Y, Li X, Miller J, Ng A, Raiman J, Sengupta S. Deep voice: Real-time neural text-to-speech. InInternational conference on machine learning 2017 Jul 17 (pp. 195-204). PMLR.
15. Bezzaoucha I. Generative AI or the Doom of Translation as we Know it?. Cahiers de Traduction. 2024 Jun 2;30(1):07-15.

