Anuvadx: A Translator for All 22 Scheduled Indian Languages

Prof. Iffat Kazi[1], Huda Qazi Syed[2], Sneha Mishra[3], Shravani Khamkar[4]
[1,2,3,4]*Department of Computer Engineering, Usha Mittal Institute of Technology, SNDT University, Mumbai, India*

***Abstract*—India's linguistic diversity, which encompasses 22 constitutionally recognized languages, presents persistent challenges for inclusive digital communication. Existing translation systems often support limited languages or modalities of single interaction, restricting the applicability of multilingual systems in the real-world. This paper presents AnuvadX, a production-grade full-stack multilingual translation platform designed to support all 22 scheduled Indian languages across multiple input and output modalities. The system delivers text-to-text translation, PDF document translation, real-time voice recognition, and neural text-to-speech synthesis within a unified architecture. The frontend is built on Next.js with React.js, while the backend is powered by FastAPI with asynchronous concurrency controls. The core translation engine employs the facebook/nllb- 200-distilled-600M model via HuggingFace Transformers and PyTorch. Speech-to-text is handled by OpenAI Whisper and speech synthesis by Microsoft Edge TTS (edgetts). PDF pro- cessing relies on PyMuPDF (fitz) for spatial geometry analysis, Indic font re-injection, and layout preservation. A multilingual virtual keyboard enables accurate native-script input. Named- entity transliteration via the indic-transliteration library pre- serves proper nouns phonetically, and a gibberish-detection preprocessing layer prevents misleading outputs. Performance is evaluated using BLEU scores, Word Error Rate (WER), and Mean Opinion Score (MOS). The architecture emphasizes modularity, low-resource optimization, and offline adaptability, with ongoing work targeting real-time voice-to-voice translation.**

***Index Terms*—Multilingual Translation, Neural Machine Translation, NLLB-200, FastAPI, Next.js, Text-to-Speech, Whis- per ASR, PDF Translation, Indian Languages, Indic Transliteration.**

I. INTRODUCTION

India is one of the most linguistically diverse nations in the world, with 22 constitutionally recognized scheduled languages and hundreds of regional dialects spoken across its states and union territories. While this diversity reflects profound cultural richness, it also generates significant communication barriers across sectors including education, health- care, governance, tourism, and public administration. Citizens migrating across linguistic zones frequently encounter difficulties in comprehending official documents, interacting with government portals, or accessing digital services rendered exclusively in English or Hindi.

Despite the rapid digitization of public services under initiatives such as Digital India, multilingual accessibility remains an unresolved challenge. Most commercial translation plat- forms focus on a narrow subset of Indian languages, prioritize text-only modalities, and lack robust handling of native scripts, code-mixed input, or low-resource languages such as Dogri, Maithili, Santali, or Bodo. Furthermore, existing tools rarely address the practical concerns of rural or semi-literate users who rely on spoken interaction rather than typed input.

Recent advancements in transformer-based Neural Machine Translation (NMT), automatic speech recognition (ASR), and neural text-to-speech (TTS) synthesis have substantially im- proved the quality of multilingual AI systems. Models such as Meta's No Language Left Behind (NLLB) and OpenAI's Whisper offer broad language coverage and strong performance even for low-resource languages. However, most aca- demic and commercial implementations remain fragmented, delivering isolated capabilities rather than a coherent, full- stack multilingual platform.

This paper presents AnuvadX, a comprehensive full-stack multilingual translation system engineered to

support all 22 scheduled Indian languages across text, document, and speech modalities. The platform integrates a Next.js and React.js frontend with a FastAPI backend, employs the facebook/nllb- 200-distilled-600M model as the core translation engine, and incorporates OpenAI Whisper for STT and Microsoft Edge TTS for speech synthesis. PDF documents are processed through PyMuPDF with UTF-8 Indic font re-injection to pre- serve layout and script fidelity. The system further integrates indic-transliteration for named-entity phonetic preservation, langdetect for automatic source language identification, and asyncio-based semaphore controls to prevent out-of-memory failures during concurrent AI inference.

The scope of this work covers the design, implementation, and evaluation of a functional production prototype demonstrating end-to-end multimodal translation capabilities. The study highlights how a carefully integrated full-stack architecture can bridge the gap between raw AI model capability and practical, user-centered multilingual accessibility across India's diverse socio-cultural landscape.

II. LITERATURE SURVEY

A. Unified Neural Machine Translation

The historical absence of parallel training data for all 22 scheduled languages was addressed by Gala et al. [1], who released the Bharat Parallel Corpus Collection (BPCC). This dataset contains 230 million bitext pairs, including 644,000 manually translated pairs that serve as ground truth for low-resource languages. Utilizing this data, they developed IndicTrans2, the first open-source model supporting all 22 languages through a Transformer architecture. To enhance cross-lingual transfer, researchers employed script unification, mapping Brahmi-derived scripts into a single Devanagari pivot.

B. Preprocessing and Tokenization Effects

Linguistic intricacies, such as the *hasanta* in Assamese, present significant challenges for standard tokenizers. As investigated by Ahmed et al. [2], Western-centric tokenizers like Moses create unwanted segregations that degrade translation quality. Conversely, research demonstrated that the IndicNLP tokenizer significantly improves performance by preserving these morphological markers.

C. Dialectal Variations and Benchmark Datasets

Standard MT systems often struggle with regional dialects. Faria et al. [3] introduced the Vashantor dataset, the first large-scale benchmark for translating regional Bangla dialects into standard Bangla. Their findings using BanglaT5 models showed high performance for the Mymensingh dialect (BLEU 69.06) but significant challenges for the Chittagong dialect (BLEU 36.75). In specialized classification tasks, Sawant and Govilkar [4] utilized Modified K-Nearest Neighbor (MKNN) to achieve 99.66% accuracy for Marathi text.

D. Multimodal and Speech-to-Speech Integration

The integration of voice and image translation is essential for real-world accessibility. Mujadia et al. [6] proposed a cascaded Speech-to-Speech Machine Translation (SSMT) pipeline. Additionally, Pabba et al. [5] developed the Live Multimodal Language Translation System (LMLTS), which integrates Tesseract OCR and Google Translate APIs.

E. Code-Mixing and Accent Adaptation

Dey et al. [9] introduced BharatBhasaNet, a framework that performs word-level language identification for 12 languages. For speech recognition, Singh et al. [8] employed Model-Agnostic Meta-Learning (MAML) to enable fast adaptation to regional accents in Hinglish ASR. The MAML approach optimized the model parameters θ:

$$\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta) \quad (1)$$

F. Advanced Fine-Tuning and Evaluation

Recent research has shifted toward preference-based optimization. Nair et al. [11] introduced a framework combining Direct Preference Optimization (DPO) and Hypergeometric Gamma Reward (HGR) to fine-tune mT5-large models. Finally, the Krutrim AI Team [7] established BharatBench, the first comprehensive framework for evaluating vision-language models on Indian cultural contexts.

III. METHODOLOGY

A. Input Processing and Normalization

All textual inputs pass through a preprocessing

pipeline before NMT inference. This pipeline performs punctuation normalization, whitespace correction, and mixed-script standardization. A gibberish-detection layer evaluates token entropy and known-word ratios to identify and reject nonsensical inputs, preventing the NMT engine from generating misleading translations from invalid queries.

Named-entity recognition is applied to identify proper nouns such as personal names and geographic locations. These entities are routed to the indic-transliteration module for phonetic script conversion rather than semantic translation, preserving their identity in the target language output.

B. Translation Pipeline

The translation request lifecycle proceeds as follows. The frontend submits a text payload or audio blob to the FastAPI backend via the api.js abstraction. For text requests, the backend acquires a semaphore slot, performs language detection if no source language is specified, applies the pre- processing pipeline, invokes the NLLB-200 model, applies post-processing and transliteration corrections, and returns the translated text. For voice input, Whisper transcribes the audio payload to text, which is then forwarded through the same text translation pipeline.

C. PDF Translation Pipeline

Uploaded PDF files are processed by PyMuPDF, which extracts text blocks with their spatial bounding-box coordinates and font metadata. Extracted text segments are translated through the NMT engine in batch. The translated segments are then re-rendered into the PDF at their original spatial positions using the appropriate cached Indic TrueType font, reconstituting the document with native-script output while preserving the original layout geometry.

D. Speech Synthesis Pipeline

Translated text is passed to the edgetts engine with the appropriate target-language voice identifier. The engine returns an audio stream that is delivered to the frontend for playback. For video translation tasks, FFmpeg extracts the original audio track, the translated audio is synthesized by edgetts, and FFmpeg recombines the translated audio with the original video frames, optionally generating aligned subtitle tracks.

E. Multilingual Virtual Keyboard

A software virtual keyboard supporting native script input for all 22 scheduled languages is integrated into the frontend. The keyboard eliminates dependency on OS-level input method editors (IME), allowing users to compose text in Devanagari, Bengali, Tamil, Telugu, Odia, and other scripts directly within the browser interface. Key mappings are dynamically loaded based on the selected source language.

F. Text-to-Speech Integration

The text-to-speech module converts translated text into natural speech output. This feature enhances accessibility for users who prefer auditory feedback or face difficulties with reading.

Speech synthesis is integrated as an output layer within the translation pipeline, ensuring smooth transition from translated text to audio generation. The modular design allows future integration of voice-to-voice functionality without restructuring the system architecture.

G. Backend Orchestration

The backend layer manages communication between frontend components and translation modules. It handles model execution, request routing, preprocessing, and output format- ting.

By separating backend orchestration from user interaction layers, the system ensures scalability and independent mod- ule upgrades. This structure allows additional features such as real-time speech recognition or offline processing to be incorporated in future iterations.

H. Testing and Validation

The system was evaluated through functional and performance-based testing. Translation quality was assessed using BLEU scores, while intelligibility of speech output was evaluated through qualitative review.

Module-level testing ensured proper handling of text input, document extraction, transliteration accuracy, and gibberish filtering. Interface responsiveness and multilingual rendering were also validated across repeated usage scenarios.

IV. SYSTEM ARCHITECTURE

A. Frontend Architecture

The web frontend is built on Next.js utilizing the modern App Router and Turbopack compiler for optimized hot-reload development and production build performance. React.js com- ponents manage state and render UI elements, while a unified CSS design system based on custom CSS variables (e.g., var (– primary-gradient)) ensures visual consistency across all views. Lucide React provides lightweight, dynamically rendered SVG iconography.

Client-server communication is abstracted through a custom api.js module that wraps the native browser fetch API with automatic JWT insertion and standardized RESTful interaction patterns. Authentication state is maintained via Local Storage tokenization, protected by a custom React Protected Route component that intercepts unauthenticated navigation attempts.

B. Backend Architecture

The backend is implemented in FastAPI, selected for its native async/await request handling, Pydantic-based schema validation, and automatic OpenAPI documentation generation. User configuration data, translation history, and authentication records are persisted in an SQLite database (anuvadx.db), pro- viding a lightweight relational store without external database dependencies.

Security is enforced through passlib with bcrypt password hashing, PyJWT for JSON Web Token generation and verification, and Pydantic for strict request body validation. A critical engineering concern during concurrent AI inference is GPU/CPU memory exhaustion. AnuvadX addresses this through Python asyncio semaphores that limit the number of simultaneous NMT, STT, and TTS inference calls, preventing out-of-memory crashes under peak load.

C. AI and NLP Engine

The core translation engine employs the facebook/nllb-200- distilled-600M model loaded via HuggingFace Transformers with PyTorch as the computation backend. NLLB-200 sup- ports all 22 scheduled Indian languages through a unified multilingual architecture, with the 600M distilled variant offering practical inference speed on CPU or modest GPU hardware. Named-entity transliteration is performed by the indic-transliteration library, which converts proper nouns into phonetically equivalent Indic scripts without altering their semantic meaning. Source language detection, when not user- specified, is performed by langdetect prior to NMT inference. Speech-to-text recognition is handled by OpenAI Whisper (base model), which transcribes spoken input across Indian languages with robust handling of accent variation. Text-to- speech output is synthesized by edgetts, which interfaces with Microsoft's Edge cloud neural TTS infrastructure to produce natural, multi-dialect audio output with rapid response times.

D. Media and Document Processing

PDF translation is implemented using PyMuPDF (fitz), which performs block-level document decomposition, spatial geometry bounding-box measurement, and whitespace nor- malization. Translated text is re-injected into the PDF using dynamically cached TrueType (ttf) fonts for Indic scripts such as Devanagari, Odia, and Tamil, ensuring correct character rendering in the output document. UTF-8 mapping is applied throughout to maintain cross-platform compatibility.

Audio and video media processing is delegated to FFmpeg, which runs asynchronously within the FastAPI pipeline to down sample, extract, or combine audio signals from uploaded MP4 or WAV files. This supports AnuvadX's voice translation and audio captioning features without blocking the main request thread.

V. IMPLEMENTATION

AnuvadX is built using a modern full-stack architecture that ensures efficiency, scalability, and smooth user experience across multiple translation modes. On the frontend, the Next.js App Router enables server-side rendering for faster initial loads and improved SEO, while client-side navigation ensures responsive interactions. The backend, powered by FastAPI, uses asynchronous processing to handle requests efficiently, with SQLite managing user data and translation history. Ad- vanced language processing is achieved using the NLLB-200 model, enabling accurate multilingual translations. The system also supports complex functionalities like PDF translation using PyMuPDF and voice/video

processing through FFmpeg and Whisper, ensuring seamless handling of text, documents, and media. Overall, the architecture integrates optimized frontend rendering, secure API handling, and powerful AI models to deliver a comprehensive, high-performance multilingual translation platform.

A. Frontend Implementation

The Next.js App Router enables server-side rendering (SSR) for initial page loads and client-side navigation for subsequent interactions, optimizing both SEO discoverability and runtime performance. The Turbopack compiler reduces hot-module replacement latency during development. The React component hierarchy is organized around three primary views: Text Translator, PDF Translator, and Voice Translator, with shared state managed through React Context. The api.js abstraction layer intercepts all outbound fetch calls, appends the stored JWT from Local Storage as an Authorization Bearer token, and handles 401 response codes by redirecting unauthenticated users to the login page. The Protected Route component wraps all authenticated views and checks for the presence of a valid token on mount.

B. Backend Implementation

FastAPI route handlers are defined as async functions, enabling non-blocking I/O for database queries (via aiosqlite) and file I/O. SQLite stores user records (hashed passwords, profile data) and translation history with timestamps. Pydantic models enforce strict type validation on all incoming request bodies, rejecting malformed payloads before they reach business logic. The NLLB-200 model is loaded once at application startup and held in memory for the process lifetime, avoiding per-request model loading overhead. The HuggingFace tokenizer is initialized with the source language FLORES-200 code, and the models generate () method is called with beam search decoding and a maximum token budget of 512. Semaphore acquisition is wrapped in an async context manager to guarantee release even on exception.

C. Language Identification and Code-Mixed Handling

For scenarios where the source language is not user-selected, the system integrates the BharatBhasaNet frame- work. This module performs granular identification at both the word and character levels, allowing it to interpret com- plex code-mixed sentences. It utilizes two distinct models: *Roberta-native* for high-accuracy identification of native scripts (99.54%) and *Roberta-Romanized* for transliterated text (60.90%). By shifting to word-wise identification, the model avoids the common bias of predicting a single language for a sentence that contains multiple linguistic influences.

D. Neural Machine Translation and Script Unification

The core translation engine is built upon the IndicTrans2 transformer model, which utilizes a specific transformer_18_18 configuration with 1.1 billion parameters. To maximize performance for low-resource languages, the implementation adopts a "Script Unification" strategy. Languages derived from the Brahmi family are mapped into a single Devanagari pivot during processing to encourage lexical sharing and transfer learning. This allows the model to leverage similarities between Indo-Aryan and Dravidian language families, ensuring high-quality output for resource- poor languages like Maithili or Dogri.

E. Text-to-Speech Synthesis and Output

The output module converts translated text into natural, intelligible speech using synthesis engines like iitmTTS. To ensure accessibility for users with varying literacy levels, the output is delivered as both real-time audio and synchronized text display. For video-based inputs, a synchronization module aligns the synthesized target audio with the original video frames using frame interpolation.

F. PDF Processing Implementation

PyMuPDF's fitz open () loads the uploaded PDF into memory. The document is iterated page-by-page; each page's get text (" dict") call returns structured block and span data including bounding boxes, font names, font sizes, and raw text content. Whitespace normalization is applied per-span before translation. The translated span text is inserted into a new PDF page using page. Insert text () with the registered Indic font, positioned at the original bounding-box coordinates scaled for the output page dimensions. TrueType font files for all major Indic scripts are cached in a local fonts directory and registered with the fitz font registry at startup.

G. Testing, Validation, and Deployment

The implementation underwent rigorous testing using met- rics such as BLEU, chrF++, and COMET. Performance was validated through Mean Opinion Scores (MOS), confirming that human post-editing can raise translation quality from 3.2 to 4.1. For deployment, the system is containerized using Docker to ensure portability. It is hosted on sovereign AI infrastructure, such as the *Yotta Shakti Cloud*, ensuring data residency and a 40% performance boost.

H. User Interface Screenshots

Fig. 1: AnuvadX Home Interface

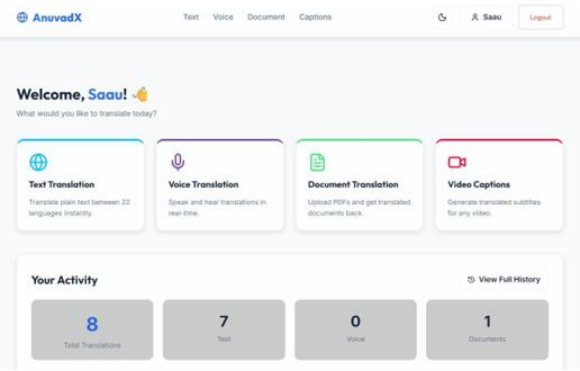

Fig. 2: User Dashboard

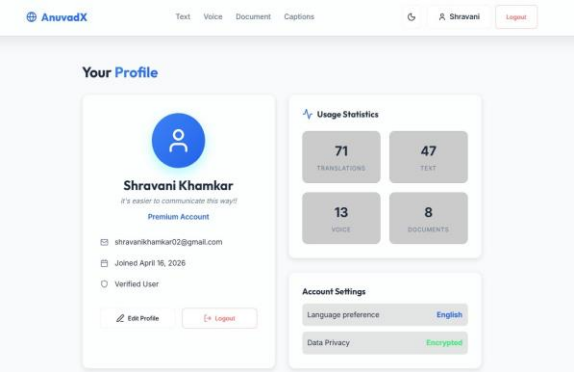

Fig. 3: User Profile

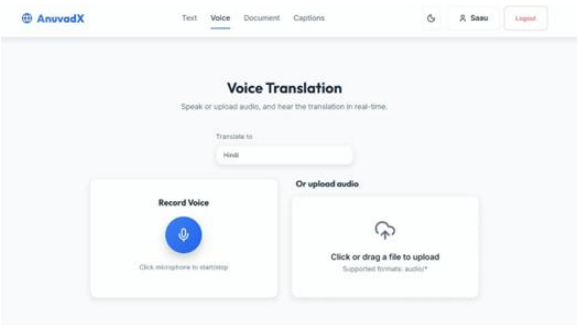

Fig. 4: Voice Translation

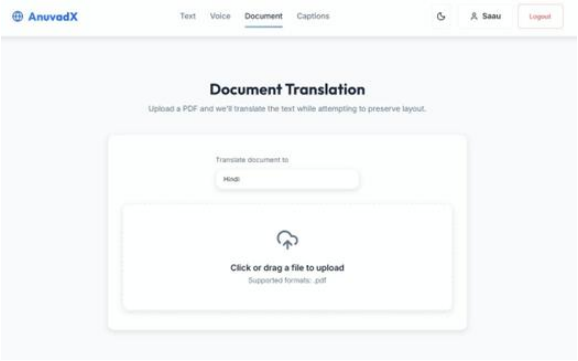

Fig. 5: Document Translation

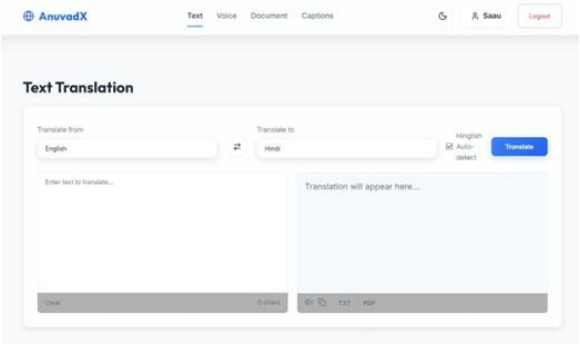

Fig. 6: Text Translation

VI. CONCLUSION AND FUTURE SCOPE

A. Conclusion

The *AnuvadX* system demonstrates how a unified, multi- lingual translation platform can make cross-linguistic communication more accessible across India's diverse linguistic landscape. The project focused on developing a practical and scalable solution that supports text-to-text translation, PDF document translation, and text-to-speech synthesis for all 22 scheduled Indian languages. By integrating multilingual virtual keyboards, named-entity transliteration, and gibberish detection mechanisms,

the system addresses common challenges such as improper translation of proper nouns, invalid inputs, and native-script accessibility.

The implementation highlights that combining transformer- based translation models with structured preprocessing and a clean user interface can significantly improve translation reliability and usability without requiring overly complex infrastructure. The modular architecture ensures that components such as translation, document processing, speech synthesis, and language identification can evolve independently. This flexibility supports continuous enhancement, including the ongoing development of real-time voice-to-voice translation.

While the current system operates as a functional prototype, it validates the feasibility of building a comprehensive multimodal translation framework tailored specifically for Indian languages. The design prioritizes inclusivity, scalability, and practical deployment over isolated algorithmic experimentation. By focusing on meaningful integration and user-centered design, AnuvadX contributes toward reducing linguistic barriers and promoting digital accessibility across sectors such as education, governance, healthcare, and public services.

B. Future Scope

Although AnuvadX delivers a robust modular framework for Indic translation, several strategic enhancements can further extend its real-world impact and technical sophistication. One of the primary future directions is the expansion of the system beyond the 22 scheduled languages to include endangered tribal languages and regional dialects. By integrating datasets like *Vashantor* for Bangla dialects [3] and frameworks like *Adi-Vaani* for tribal scripts such as Santali or Gondi [7], [11], the system can achieve a higher degree of linguistic inclusivity, serving communities that have traditionally lacked digital representation.

Another key enhancement involves the transition from the current cascaded architecture (ASR → MT → TTS) toward end-to-end speech-to-speech translation (S2ST) models. As discussed in [6], cascaded systems are prone to error propagation between modules. Future work will explore fused

Speech-Text architectures that leverage prosodic cues—such as tone and emphasis—to produce more natural, expressive translations while reducing latency to sub-second levels for real-time conversation. This will be complemented by the integration of "Adaptive MT," allowing the system to continuously learn from user feedback and human post-editing to improve accuracy in specialized domains like law or healthcare over time.

From a platform perspective, a major priority will be the implementation of real-time "Website Translation" capabilities. This feature will allow users to browse and translate entire web domains into any of the 22 scheduled languages while maintaining the original layout and interactive elements. This extends the system's utility into document-level translation, where context from across a page is used to resolve linguistic ambiguities that sentence-level models often miss.

Technically, optimizing the system for "Edge-based" or on- device processing will be essential for deployment in rural regions with low internet connectivity. By utilizing distilled, compact versions of models like *IndicTrans2* [1], the system can provide secure, private translation without relying on cloud infrastructure. Furthermore, future iterations will focus on improving robustness in "In-the-Wild" environments, employing meta-learning techniques [8] to maintain high speech recognition accuracy despite background noise and diverse regional accents. Ultimately, through large-scale user studies and the creation of "Indic-original" benchmarks [7], [10], AnuvadX has the potential to evolve into a comprehensive digital public infrastructure. By bridging the gap between English-centric digital content and India's vast multilingual population, the project aims to foster a more equitable and participatory digital ecosystem for over 1.3 billion citizens.

REFERENCES

[1] Gala *et al.*, "IndicTrans2: Towards high-quality and accessible machine translation models for all 22 scheduled Indian languages," *Trans. Mach. Learn. Res.*, Dec. 2023.

[2] M. A. Ahmed, K. Kashyap, and S. K. Sarma, "Tokenization effect on neural machine translation: An experimental investigation for English-Assamese," in *Proc. 2023 14th Int. Conf. Comput. Commun. Netw. Technol. (ICCCNT)*, 2023.

[3] T. J. Faria *et al*., "VASHANTOR: A large-scale multilingual benchmark dataset for automated translation of Bangla regional dialects to Bangla language," *arXiv preprint* arXiv:2311.11142, Nov. 2023.
[4] S. Sawant and S. Govilkar, "Text classification in Marathi language," *Int. J. Sci. Res. Eng. Manag. (IJSREM)*, 2024.
[5] P. Pabba *et al*., "Live multimodal language translation system: Integrating real-time text, voice, image, and document translation," *Telematique*, vol. 23, no. 1, pp. 776–786, 2024.
[6] V. Mujadia *et al*., "Towards speech-to-speech machine translation focusing on Indian languages," in *Proc. 17th Conf. Eur. Chapter Assoc. Comput. Linguist. (EACL)*, May 2023.
[7] Krutrim AI Team, "BharatBench: Comprehensive multilingual multimodal evaluations of foundation AI models for Indian languages," *preprint*, 2024.
[8] S. Singh *et al*., "Hinglish cross-accent model agnostic meta-learning automatic speech recognition," *ACM Trans. Asian Low-Resour. Lang. Inf. Process.*, vol. 24, no. 9, Sep. 2025.
[9] S. Dey *et al*., "BharatBhasaNet—A unified framework to identify Indian code mix languages," *IEEE Access*, vol. 12, pp. 68893–68904, 2024.
[10] L. Madanbhavi *et al*., "An efficient multilingual text classification using IndicCorp dataset," in *Proc. 2024 5th IEEE Glob. Conf. Adv. Technol. (GCAT)*, Oct. 2024.
[11] R. Nair *et al*., "Enhancing low-resource Indian language MT using LLMs," *IEEE*, 2026.