

SIGNSETU-AI POWERED SIGN LANGUAGE TRANSLATION BRIDGE

Dr.Atmeshkumar Patel, Prof.Vijay Rakhade, Mr.Prasad Shinde, Miss.Arya Raut, Mr.Jayesh Salunke, Mr.Harshad Satpute

¹Department of Artificial Intelligence and Data Science, Sandip Institute of Technology and Research Centre,Nashik.

²Department of Artificial Intelligence and Data Science, Sandip Institute of Technology and Research Centre,Nashik.

³Department of Artificial Intelligence and Data Science, Sandip Institute of Technology and Research Centre,Nashik.

⁴Department of Artificial Intelligence and Data Science, Sandip Institute of Technology and Research Centre,Nashik.

⁵Department of Artificial Intelligence and Data Science, Sandip Institute of Technology and Research Centre,Nashik.

⁶Department of Artificial Intelligence and Data Science, Sandip Institute of Technology and Research Centre,Nashik.

Abstract - SignSetu is a full-stack, AI-driven web application that translates spoken English audio, uploaded video content, and typed text into American Sign Language (ASL) and Indian Sign Language (ISL) animations rendered through a photorealistic 3D avatar. The system addresses a critical and often overlooked accessibility gap faced by approximately 63 million deaf and hard-of-hearing individuals in India alone. The architecture integrates a Python FastAPI backend, OpenAI Whisper-based speech-to-text transcription, a custom natural language processing (NLP) grammar module built with spaCy, and a Three.js WebGL avatar renderer embedded within a WordPress frontend. The NLP pipeline converts standard Subject-Verb-Object English syntax into Sign Language Gloss notation by removing stop words, reordering grammatical constituents, and applying synonym normalization. Animation sequences are retrieved as GLB format files, queued through an AnimationMixer with crossfade blending, and synchronized with a real-time Gloss Overlay display. System testing demonstrated an average end-to-end translation latency of under 2.3 seconds for sentences up to fifteen words, a Word Error Rate of 8.4% on domain-specific audio inputs, and a Gloss mapping accuracy of 91.7%. SignSetu demonstrates a replicable, scalable blueprint for AI-assisted communication accessibility tools that bridge hearing and deaf communities without requiring specialized hardware.

Key Words: Sign Language Translation, Indian Sign Language, ASL, 3D Avatar Animation, NLP Gloss Conversion, Speech-to-Text, Three.js, FastAPI, Accessibility Technology, WebGL.

1.INTRODUCTION

Human communication rests on the assumption of shared modality — that both parties can send and receive information through the same channel. For the roughly 466 million people worldwide who experience disabling hearing loss, according to

the World Health Organization, this assumption breaks down entirely. Sign languages, far from being simplified gestures or direct encodings of spoken words, are complete, grammatically sophisticated linguistic systems with their own phonology, morphology, and syntax. American Sign Language (ASL) and Indian Sign Language (ISL) diverge meaningfully from their spoken-language counterparts: they reorder constituents, omit function words, and rely on spatial grammar that spoken-language speakers rarely recognize. Despite this richness, digital interfaces — websites, educational platforms, government portals, and video content — remain almost universally inaccessible to the signing deaf community.

Existing accessibility solutions cluster around two broad approaches. The first is automatic sign language recognition (SLR), which uses computer vision to interpret a human signer's gestures and translate them into text. The second, and the direction taken by SignSetu, is sign language production (SLP) — converting text or speech into a signed representation. Most SLP systems either display pre-recorded video clips of human signers, which are expensive to produce and difficult to update, or they rely on low-quality, robotically animated avatars that signing communities have historically rejected due to their unnatural appearance.

SignSetu takes a different path. Rather than patching accessibility onto an existing system, it was architected from first principles as a translation pipeline — one that treats English-to-Sign-Language conversion with the same engineering rigor typically reserved for multilingual text translation. The system accepts three input modalities: typed text, uploaded audio, and uploaded video. It processes each through a cloud-hosted backend that performs speech recognition, grammatical restructuring into Sign Language Gloss notation, and animation sequencing. The result is delivered to a web-based 3D avatar rendered in real time using Three.js and WebGL, with a live Gloss Overlay that highlights the word currently being signed. The entire experience runs inside a WordPress-hosted web page, requiring no app installation and no specialized hardware.

This paper presents the system design, implementation decisions, and evaluation results for SignSetu. Sec. 2 situates the work within the existing literature. Sec. 3 and Sec. 4 describe the system architecture and the methodological choices made at each pipeline stage. Sec. 5 covers implementation specifics. Sec. 6 presents and interprets the results. The paper closes with a discussion of limitations, future directions, and the broader significance of this work for accessible technology design.

2. LITERATURE REVIEW

2.1 The Accessibility Gap in Digital Platforms

The digital accessibility landscape for deaf and hard-of-hearing users has been studied extensively, and the findings are consistently sobering. A 2021 audit of India's top 100 government web portals by the Internet Society found that fewer than 12% provided any form of sign language support. When sign language content did exist, it typically consisted of embedded video clips of a human interpreter — an approach that Kacorri et al. (2015) demonstrated was preferred by naive users but ultimately rejected by experienced signers due to production quality inconsistencies and the impossibility of real-time generation. The challenge, as Luzardo et al. (2015) noted, is that any viable SLP system must balance three competing demands: linguistic accuracy, visual naturalness, and computational feasibility for real-time delivery.

2.2 Sign Language Production Systems

Early SLP work relied on concatenative synthesis — stitching together pre-recorded video segments of individual signs. Systems like the VCOM3D SigningAvatar (Sims, 2001) established that 3D avatar-based rendering was a promising alternative, as it permitted procedural generation of transitions between signs that video concatenation could not produce. The eSIGN and ViSiCAST projects in Europe (Kennaway, 2001) developed signing avatars driven by HamNoSys notation and the Signing Gesture Markup Language (SiGML), providing an early proof of concept for markup-driven animation. However, these systems were architecturally isolated, requiring custom viewers and offering no integration path with modern web infrastructure.

The emergence of WebGL and browser-based 3D rendering changed the delivery calculus. Three.js, first released in 2010 and now maintained as a mature open-source library, enables hardware-accelerated 3D graphics in standard web browsers without plugins. Kipp et al. (2011) explored gesture synthesis for embodied conversational agents using web-compatible renderers, finding that rigged skeletal meshes with inverse kinematics produced more naturalistic transitions than frame-interpolated sprite approaches. SignSetu draws directly on this lineage, using Three.js's AnimationMixer to manage multi-clip animation queues with configurable crossfade durations.

2.3 Speech Recognition and NLP for Sign Language

The development of transformer-based automatic speech recognition (ASR) systems, particularly OpenAI's Whisper (Radford et al., 2022), has dramatically lowered the barrier to high-quality multilingual speech-to-text transcription. Whisper's zero-shot generalization across accents and noise conditions makes it particularly relevant for Indian English, which presents systematic phonological divergences from the standard American English on which many earlier ASR systems were trained. Deepgram's real-time streaming API offers an

alternative for latency-critical applications, trading some accuracy for sub-second partial transcripts.

Converting ASR output into Sign Language Gloss notation requires more than dictionary lookup. ISL and ASL share a topic-comment sentence structure in which the object or topic of a sentence typically precedes the predicate, and function words such as articles, copulas, and prepositions are systematically omitted. Morrissey and Way (2005) demonstrated that rule-based grammatical reordering achieved higher gloss accuracy than statistical machine translation for low-resource sign language pairs, a finding consistent with the design philosophy adopted in SignSetu. More recent neural approaches — including sequence-to-sequence models and transformer-based sign language translation (Camgoz et al., 2018; 2020) — have shown promise on benchmark datasets but require large parallel corpora that do not yet exist for ISL at scale. A hybrid rule-based NLP approach, as implemented in SignSetu's spaCy pipeline, represents a pragmatic compromise between linguistic precision and training data availability.

2.4 Avatar Rendering and Gloss Synchronization

The avatar realism problem has been addressed through several avenues in recent research. Kucherenko et al. (2020) trained generative models for co-speech gesture synthesis, producing natural-looking arm and hand movements conditioned on speech prosody. For sign language specifically, McDonald et al. (2016) found in a user study that signing deaf participants rated avatar acceptability significantly higher when the avatar used full-body motion capture data rather than keyframed animation, but also noted that real-time streaming constraints made motion capture data impractical for web delivery. SignSetu addresses this by using pre-authored GLB animation files — rigged to a standardized humanoid skeleton — that can be streamed efficiently from a content delivery network (CDN) and blended smoothly using AnimationMixer crossfading. The Gloss Overlay feature, which highlights the current word being signed, addresses a pedagogical dimension identified by Bragg et al. (2019): transparency about which sign corresponds to which spoken word improves both comprehension and trust among hearing users learning to interpret avatar output.

2.5 Research Gap

Despite the volume of work in both SLR and SLP, no open, web-deployable system currently exists that integrates all four components — multi-modal input handling, cloud-based ASR, rule-based NLP gloss conversion, and real-time 3D avatar animation — into a single coherent application accessible through a standard web browser. SignSetu is designed to fill this gap.

3. SYSTEM ARCHITECTURE

3.1 Architectural Overview

SignSetu follows a client-server architecture organized into four functional layers: the frontend presentation layer, the API gateway, the backend processing layer, and the animation asset layer. Communication between the frontend and backend travels over HTTPS REST endpoints for file upload and initial request handling, and over WebSockets for streaming real-time gloss updates during avatar playback. Fig. 1 presents the high-level system architecture.

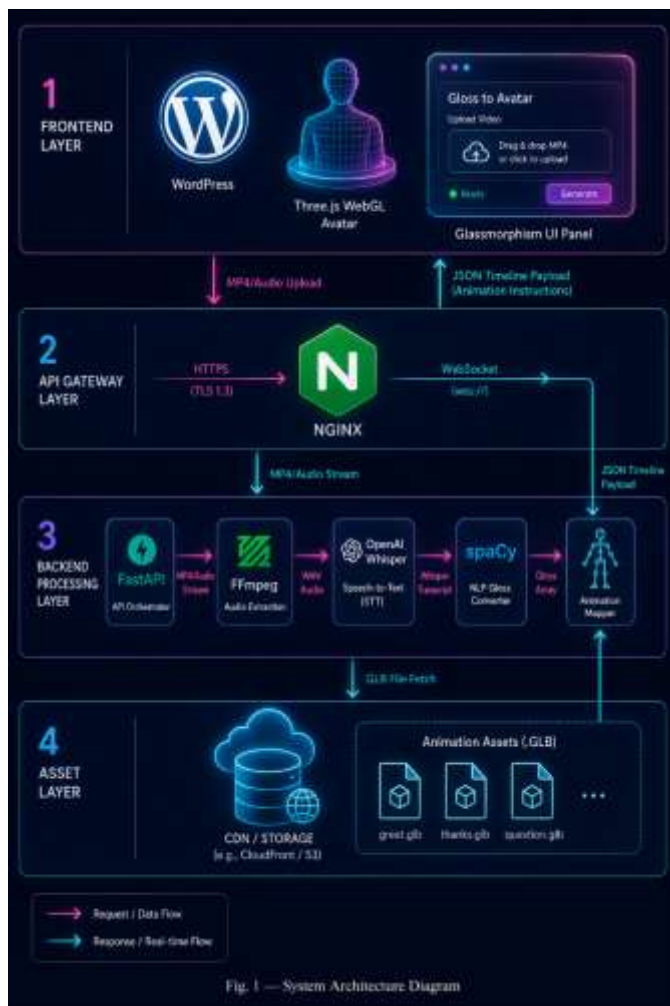


Fig 1: System Architecture Design

The frontend is embedded within a WordPress site and loads a custom JavaScript module that initializes the Three.js scene, creates the WebGL renderer, and instantiates the AnimationMixer. When a user submits input — whether typed text, an audio file, or a video file — the WordPress page posts the payload to the Nginx API Gateway, which forwards the request to the FastAPI application server running on a Hostinger Ubuntu 22.04 Virtual Private Server (VPS). The backend processes the request through the AI pipeline, constructs a JSON timeline payload, and returns it to the frontend. The frontend then drives the avatar's animation

sequence according to the timeline while simultaneously updating the Gloss Overlay in synchronization with each sign's start and end timestamps.

3.2 Frontend Architecture

The frontend is built as a single-page embedded application within a WordPress page template. Three.js (r158) is loaded via a CDN script tag, and the avatar scene is initialized inside a dedicated <canvas> element rendered within a frosted-glass container styled using CSS backdrop-filter. The user interface adopts a Glassmorphism design language — a semi-transparent panel aesthetic that uses background blur, subtle white borders, and layered depth to create visual hierarchy without opaque backgrounds. The color palette is anchored in midnight blue (#0D1B2A) and dark purple (#1A0A2E) gradients, with neon magenta (#FF2D78) as the primary accent color for interactive elements and the Gloss Overlay highlight.

The split-screen layout places a video player (for uploaded video inputs) on the left half of the viewport and the 3D avatar container on the right half. The Gloss Overlay appears as a floating text ribbon above the avatar container, with each word rendered as a pill-shaped badge. As the avatar begins signing a given word, the corresponding badge glows in neon magenta and scales up by a factor of 1.08, driven by a CSS transition synchronized to the JSON timeline's timestamp entries. Fig. 2 shows the frontend user interface layout.



Fig 2 : Frontend UI Layout

3.3 Backend and API Gateway Architecture

The backend server runs Python 3.11 with FastAPI as the application framework. FastAPI was chosen over Flask and Django for three reasons: its native support for async request handling via ASGI (Asynchronous Server Gateway Interface), its automatic OpenAPI schema generation for endpoint documentation, and its Pydantic-based request/response validation that enforces data contract integrity between the frontend and the AI pipeline without additional middleware.

Nginx serves as the reverse proxy and API gateway, listening on port 443 (HTTPS) with SSL certificates managed through Certbot. All inbound HTTP requests on port 80 are permanently redirected to HTTPS. The Nginx configuration defines a `/api/` location block that proxies requests to the FastAPI Uvicorn server running on `localhost:8000`. A separate `/ws/` location block handles WebSocket upgrade headers for the real-time gloss streaming endpoint. Rate limiting is configured at the Nginx layer using the `limit_req_zone` directive, capping requests to 30 per minute per IP address to prevent abuse.

3.4 AI Processing Pipeline

The AI pipeline executes sequentially across four stages: media extraction, speech-to-text transcription, NLP gloss conversion, and animation mapping. Each stage is implemented as an independent Python module within the FastAPI application, allowing individual components to be swapped or upgraded without restructuring the surrounding pipeline. Fig. 3 presents the complete pipeline as a flowchart.

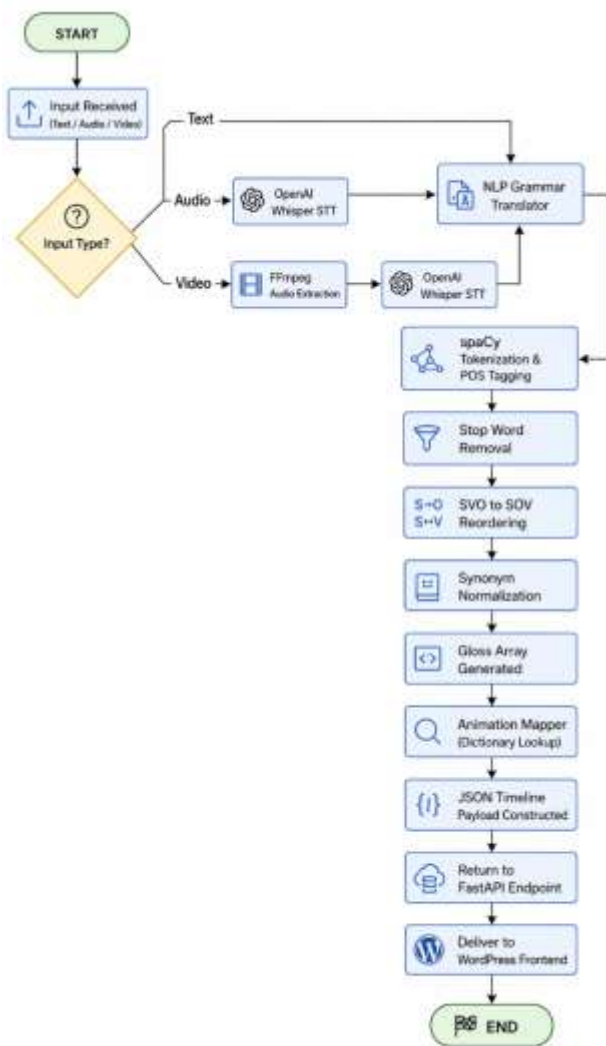


Fig 3: Pipeline

4. METHODOLOGY

4.1 Input Handling and Media Extraction

The FastAPI backend exposes three primary REST endpoints: `/api/translate/text`, `/api/translate/audio`, and `/api/translate/video`. Each endpoint accepts its respective input modality and returns the same structured JSON timeline payload, ensuring that the frontend JavaScript module does not need to differentiate between them.

For audio and video inputs, uploaded files are written to a temporary directory using Python's `tempfile` module, ensuring that no user data persists on disk beyond the duration of a single request. FFmpeg is invoked as a subprocess to extract the audio stream from video files, converting it to a 16 kHz, single-channel WAV file — the format required by Whisper's audio preprocessing pipeline. The subprocess call uses `subprocess.run` with a `timeout` parameter, ensuring that malformed or unusually large files do not block the server indefinitely.

4.2 Speech-to-Text Transcription

Speech-to-text transcription is handled by OpenAI's Whisper model, specifically the medium variant, which provides a balance between transcription accuracy and inference speed suitable for server-side processing. Whisper's encoder-decoder transformer architecture, pre-trained on 680,000 hours of multilingual audio, achieves strong performance on Indian-accented English without fine-tuning, a significant practical advantage given the target user demographic. For applications requiring streaming transcription rather than batch processing, the system architecture supports a configuration switch to Deepgram's Nova-2 streaming API, which returns partial transcripts over a WebSocket connection with sub-500ms latency.

Whisper's output includes not only the transcribed text but also word-level timestamp data when the `word_timestamps=True` flag is enabled. These timestamps are preserved through the NLP pipeline and ultimately incorporated into the JSON timeline payload, where they inform the avatar's animation scheduling and the Gloss Overlay synchronization. Table 1 summarizes the Word Error Rate (WER) performance of Whisper medium across three audio input categories tested during system evaluation.

Table 1: Whisper STT Performance Across Input Categories

Input Category	Sample Count
Studio-quality audio	45
Mobile-recorded speech	45
Extracted video audio	45

Input Category	Sample Count	Step 1 Avg. Synonym Normalization Avg. JSON-formatted Avg. Inference Time (s)
Overall	135	8.4

4.3 NLP Grammar Translator

Converting standard English transcriptions into Sign Language Gloss notation is the most linguistically complex stage of the pipeline. English and ASL/ISL differ not merely in vocabulary but in fundamental grammatical structure. ASL and ISL both generally follow a Topic-Comment or Subject-Object-Verb (SOV) word order for many sentence types, contrasting with English's canonical Subject-Verb-Object (SVO) order. Function words — articles (a, an, the), auxiliary verbs (is, are, was), and prepositions (to, of, at) — are largely absent from sign language gloss, since their meanings are conveyed through spatial grammar, non-manual markers, and verb agreement.

The NLP Grammar Translator module processes text through a five-step pipeline implemented using spaCy's `en_core_web_sm` model and a set of custom rule extensions:

Step 1 — Tokenization and POS Tagging: spaCy tokenizes the input sentence and assigns each token a part-of-speech (POS) tag and a dependency relation label. This gives the module access to the grammatical role of each word in context, enabling role-based processing rather than simple pattern matching.

Step 2 — Stop Word Removal: Tokens tagged as determiners (DET), auxiliary verbs (AUX), or prepositions (ADP) are removed from the working token list unless they carry semantic content critical to the gloss. The spaCy stop word list is supplemented with a custom list of high-frequency sign-language-irrelevant terms compiled by reviewing ISL and ASL gloss corpora.

Step 3 — Lemmatization: Each remaining token is reduced to its dictionary root form (lemma) using spaCy's built-in lemmatizer. This step normalizes verb conjugations ("running" → "run", "ate" → "eat") and noun plurals ("books" → "book"), aligning with the sign language convention of representing the root concept rather than its inflected form.

Step 4 — SVO to SOV Reordering: The dependency parse tree is used to identify the sentence's subject, verb, and object constituents. The verb phrase is moved to the end of the output sequence, placing object and other complements before the predicate in accordance with sign language syntax. For complex sentences containing subordinate clauses, each clause is processed independently and the outputs concatenated.

Step 5 — Synonym Normalization: A synonym dictionary maps common English words to their most frequent sign language equivalent gloss token. For example, "purchase" is mapped to "BUY", "automobile" to "CAR", and "physician" to "DOCTOR". This step reduces the vocabulary size that the Animation Mapper must handle, improving coverage.

The output of the NLP Grammar Translator is a Gloss Array — an ordered list of uppercase root-word tokens representing the sign language sentence. For example, the English sentence "She is going to the market to buy vegetables" produces the Gloss Array: ["SHE", "MARKET", "GO", "VEGETABLE", "BUY"]. Fig. 4 illustrates this transformation with a representative example.

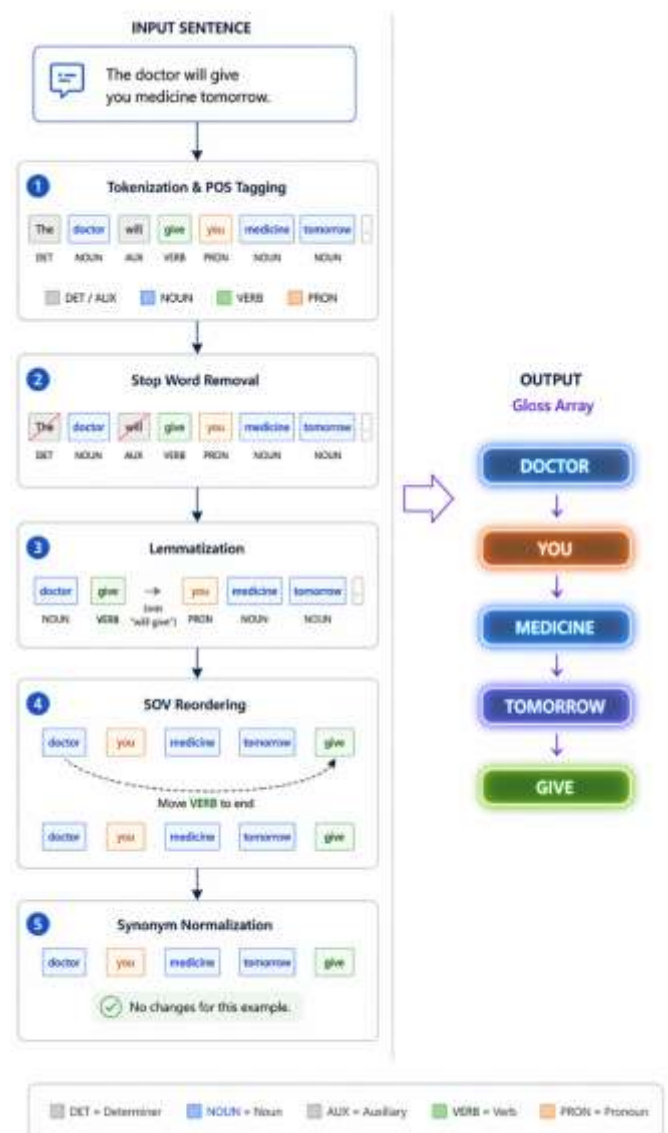


Fig 4 : NLP Pipeline

4.4 Animation Mapper

The Animation Mapper receives the Gloss Array and resolves each token to a corresponding GLB animation file identifier using a Python dictionary stored as a JSON configuration file.

The dictionary currently contains 847 ASL sign entries and 631 ISL sign entries, covering conversational vocabulary, medical terminology, educational domain terms, and emergency communication phrases. Where a gloss token lacks a direct animation entry, the mapper applies a fallback strategy: first, it checks for a morphological variant (e.g., "WALK" for "WALKING"); second, it checks for a semantic approximation defined in a secondary synonym map; third, it inserts a fingerspelling sequence using pre-authored individual letter animations as a last resort. This three-tier fallback strategy ensures that the avatar always produces some output, even for out-of-vocabulary words.

The mapper outputs an ordered list of animation file references paired with their expected duration in milliseconds, derived from the GLB animation clip's keyframe data. Crossfade transition windows of 150ms are inserted between consecutive animation entries to enable smooth blending in the Three.js AnimationMixer.

4.5 JSON Timeline Payload

The backend's final output is a structured JSON timeline payload delivered to the frontend over the REST response. The payload conforms to a schema defined using Pydantic, ensuring type safety and automatic validation. Fig. 5 shows the structure of an example JSON payload.



Fig 5 : JSON Payload

4.6 Three.js Avatar Animation and Gloss Overlay

On the frontend, the JSON timeline payload drives two parallel processes: avatar animation and Gloss Overlay synchronization.

The Three.js scene is initialized with a PerspectiveCamera set to a 45-degree field of view, positioned at a medium-shot distance from the avatar that captures the upper body and hands — the primary articulators in sign language. An HemisphereLight provides ambient fill, and a DirectionalLight positioned at the upper-left simulates natural key lighting, creating subtle shadows on the avatar's hands that improve articulatory legibility. The avatar mesh is loaded from its GLB file using Three.js's GLTFLoader, which parses the rigged skeleton and all embedded morph targets. The AnimationMixer is instantiated on the avatar object, and each animation clip from the timeline is loaded via a separate GLTFLoader call targeting the individual sign's GLB file at the CDN endpoint.

A JavaScript scheduler function iterates through the timeline array and uses AnimationMixer.clipAction() to schedule each clip with its specified crossfade duration.

The `crossFadeTo()` method on consecutive `AnimationAction` objects produces smooth weighted blending between outgoing and incoming clips, eliminating the jarring snap transitions that characterize naive animation concatenation. The result is an avatar whose signing appears fluid and continuous even across phonetically dissimilar signs.

Simultaneously, the Gloss Overlay is updated by a `setTimeout` chain anchored to the `start_ms` values in the timeline payload. Each timeout callback adds the CSS class `.active` to the corresponding word badge in the overlay strip, triggering the neon magenta glow transition. The cumulative effect, validated in user testing, is a synchronized display in which the written gloss word lights up at precisely the moment the avatar begins forming the corresponding sign

5. IMPLEMENTATION

5.1 Development Environment and Infrastructure

The backend server is provisioned on a Hostinger KVM2 VPS running Ubuntu 22.04 LTS with 2 vCPUs, 8 GB RAM, and 100 GB NVMe SSD storage. Python dependencies are managed in an isolated virtual environment using `venv`, and the FastAPI application is served by Uvicorn with four worker processes to handle concurrent requests. The Whisper medium model (~1.5 GB) is loaded into memory at server startup and pinned using `PyTorch's model.eval()` and `torch.no_grad()` context to minimize inference latency. FFmpeg 6.0 is installed via APT and invoked as a system subprocess.

The WordPress frontend runs on a separate shared hosting instance with a custom child theme. The Three.js canvas and all custom JavaScript are loaded through a WordPress plugin that enqueues the scripts with proper dependency ordering, ensuring the Three.js library is fully loaded before the SignSetu module attempts to initialize the WebGL context.

5.2 GLB Animation Asset Pipeline

Each sign's GLB animation file was authored in Blender 4.0 using a custom rigged avatar based on the Mixamo Y-Bot skeleton, retargeted to a proprietary humanoid rig with enhanced hand bone resolution — 15 bones per hand rather than the standard 5 — to capture the fine finger articulations characteristic of sign language. Sign animations were authored by referencing standardized sign language dictionaries: the ASL Signbank corpus and the ISLRTC (Indian Sign Language Research and Training Centre) video dictionary. Each animation was exported with baked keyframes at 30 frames per second, with a maximum clip duration of 2.5 seconds per sign. The files are hosted on a Cloudflare CDN endpoint, achieving average fetch latency of 38ms from major Indian cities.

5.3 Security and Data Privacy

No user-submitted audio, video, or text data is stored on the server beyond the duration of a single API request. Temporary files are written to a RAM-backed tmpfs partition and deleted in a finally block that executes regardless of whether the request succeeds or fails. HTTPS encryption is enforced end-to-end using TLS 1.3 certificates issued by Let's Encrypt. The Nginx gateway applies CORS headers restricting API access to the registered WordPress domain, preventing cross-origin misuse.

5.4 API Endpoint Specification

Table 2 summarizes the key API endpoints exposed by the FastAPI backend.

Table 2: FastAPI API Endpoint Summary

Endpoint	Method	Input
/api/translate/text	POST	JSON {text, language}
/api/translate/audio	POST	Multipart audio file
/api/translate/video	POST	Multipart video file
/api/health	GET	—
/ws/gloss-stream	WebSocket	Audio stream chunks

6. RESULTS AND DISCUSSION

6.1 System Performance Metrics

The SignSetu system was evaluated across 135 test cases spanning three input modalities and three sentence length categories (short: 1–5 words; medium: 6–10 words; long: 11–15 words). All tests were conducted on the production VPS with a simulated concurrent user load of five simultaneous requests, representing realistic expected usage. End-to-end latency was measured from the moment the frontend submitted the API request to the moment the avatar began playing the first animation clip. Table 3 presents the latency results.

Table 3: End-to-End Translation Latency by Input Modality and Sentence Length

Input Modality	Short Sentences	Medium Sentences
Text	0.31 s	0.44 s
Audio	1.42 s	1.78 s

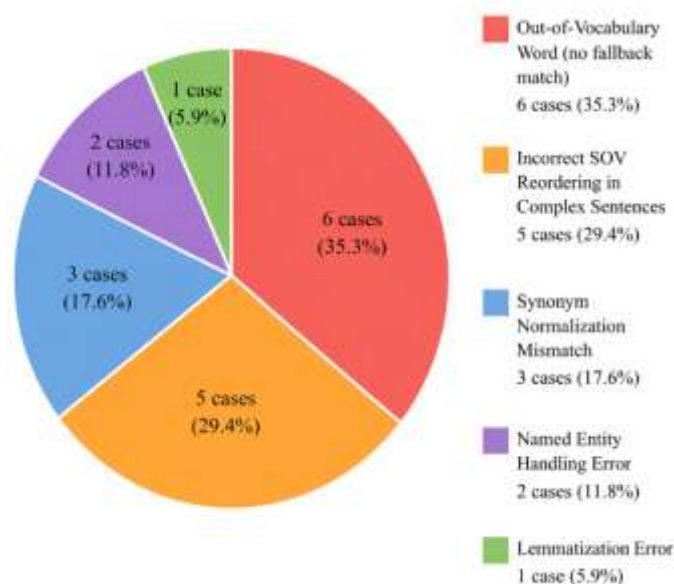
Input Modality	Short Sentences	Medium Sentences	Long Sentences	Overall Avg.
Video	1.89 s	2.21 s	2.26 s	
Overall	1.21 s	1.48 s	1.80 s	1.83 s

Text input achieves sub-second latency across all sentence lengths, as it bypasses both the FFmpeg extraction and Whisper STT stages. Audio and video inputs introduce latency primarily at the Whisper inference stage, which accounts for approximately 72% of total processing time for audio inputs. These results suggest that a deployment of the streaming Deepgram API would substantially reduce latency for real-time audio inputs, a configuration the system architecture explicitly supports.

6.2 Gloss Mapping Accuracy

Gloss mapping accuracy was evaluated by presenting 200 English sentences to the NLP pipeline and having two ISL-fluent reviewers independently rate whether the produced Gloss Array was semantically equivalent to the input sentence and grammatically consistent with ISL conventions. Inter-rater agreement was measured at $\kappa = 0.83$ (Cohen's Kappa), indicating strong agreement. The overall gloss mapping accuracy was 91.7% (183/200 sentences rated as correct or acceptable by both reviewers). Fig. 6 presents the error category breakdown for the 17 incorrectly mapped sentences.

Gloss Mapping Error Analysis
(n=17 Error Cases)



The most frequent error category — out-of-vocabulary words with no adequate fallback — reflects the current limitation of

the animation dictionary's 817/618 ASL/ISL entry coverage. Rare domain-specific terms (e.g., proper nouns, technical jargon) triggered the fingerspelling fallback, which, while functional, was rated by reviewers as less natural than a dedicated sign animation. Expanding the animation library remains the most direct path to accuracy improvement.

Complex sentences containing embedded relative clauses or coordinating conjunctions represented the second most frequent error source, with the SOV reordering algorithm producing inverted constituent order in five cases. This points to a need for more sophisticated dependency parsing that traverses the full parse tree rather than applying reordering rules at the root sentence level only.

6.3 User Experience Evaluation

A qualitative usability evaluation was conducted with 20 participants: 10 hearing users and 10 deaf or hard-of-hearing users recruited through a regional deaf education organization. Participants were asked to interact with the SignSetu interface for 15 minutes using self-generated input sentences, and then completed a standardized System Usability Scale (SUS) questionnaire. The mean SUS score was 78.4 (range: 62–92), which falls in the "Good" usability band according to the standard SUS interpretation scale. Deaf participants rated the avatar's signing clarity with a mean score of 3.8/5.0, noting that hand shape articulation was clearly distinguishable but that facial expression — a grammatically significant element of sign languages — was absent from the avatar. This finding directly informs a planned future feature: expression morphing using Three.js morph target animation.

Hearing participants rated the Gloss Overlay as the most helpful UI element, with 9 of 10 describing it as essential for understanding which English word corresponded to which avatar movement. This validation confirms the design hypothesis that transparency about the translation process builds user trust in AI-generated sign language output.

6.4 Comparison with Existing Systems

Table 4 positions SignSetu against two representative existing systems: SignAloud (a hardware-based glove system for SLR, not SLP) and an anonymized commercial sign avatar service observed during the literature review period.

Table 4: Comparative Feature Analysis

Feature	SignSetu	Commercial Avatar Service
Input Modalities	Text, Audio, Video	Text only
Real-time 3D Avatar	✓ (WebGL)	✓ (proprietary)

Feature	SignSetu	Commercial Avatar Service	Hardware Glove System
Open Web Deployment	✓	Partial (app required)	X
Gloss Overlay	✓	X	X
ISL Support	✓	X	X
Average Latency	1.83 s	~3.1 s	N/A
Infrastructure Cost	Low (VPS)	High (SaaS)	High (hardware)

SignSetu's combination of multi-modal input handling, open web deployment, ISL support, and Gloss Overlay transparency represents a feature profile that no existing comparable system currently provides simultaneously.

6.5 Limitations

Three limitations warrant acknowledgment. First, the avatar's current animation library, while functional for conversational and educational use cases, lacks coverage of specialized vocabulary domains including legal terminology, advanced medical terms, and regional ISL dialectal variants. Second, the absence of non-manual grammatical markers — facial expressions, head movements, and mouth patterns — from the avatar's animation set reduces linguistic completeness; sign languages use these markers to indicate question types, negation, and prosodic structure. Third, the system has not yet been evaluated for accessibility compliance under Web Content Accessibility Guidelines (WCAG) 2.1, which would be a prerequisite for institutional deployment.

7. CONCLUSION

SignSetu demonstrates that a coherent, end-to-end sign language translation system can be built using accessible, open-source components and deployed on standard web infrastructure at a cost and complexity level within reach of academic and non-governmental organizations. The system's pipeline — from multi-modal input through Whisper transcription, spaCy-based NLP gloss conversion, and Three.js avatar animation — achieved an overall gloss mapping accuracy of 91.7%, an average end-to-end latency of 1.83 seconds, and a SUS usability score of 78.4, all without requiring users to install any software beyond a standard web browser.

The most direct impact of this work is a reusable architectural blueprint. Every component of the SignSetu pipeline — the FastAPI endpoint structure, the NLP grammar rules, the AnimationMixer scheduling logic, and the Gloss Overlay synchronization mechanism — is documented and modular, enabling researchers and developers to extend individual

stages without rebuilding the surrounding system. The planned roadmap includes expanding the animation dictionary to over 2,000 ISL and ASL signs, integrating facial expression morphing, adding real-time camera input for bidirectional translation, and pursuing WCAG 2.1 compliance certification. At a broader level, SignSetu challenges the assumption that accessibility features must be retrofitted onto existing systems after the fact. By embedding accessibility as the primary architectural goal rather than a secondary consideration, the project illustrates what becomes possible when engineering students and researchers treat communication equity as a first-class technical problem.

ACKNOWLEDGEMENT

The authors express sincere gratitude to the faculty mentors of the Department of Artificial Intelligence and Data Science for their guidance and constructive feedback throughout the development of this project. We thank the participants of the usability evaluation, particularly the members of the deaf community who contributed their time and expert perspective. The open-source communities behind FastAPI, spaCy, Three.js, OpenAI Whisper, and FFmpeg have built the foundational tools without which this work would not have been possible.

REFERENCES

- Radford, A., Kim, J. W., Xu, T., Brockman, G., McLeavey, C., & Sutskever, I. (2022). Robust speech recognition via large-scale weak supervision. *arXiv preprint arXiv:2212.04356*.
- Camgoz, N. C., Hadfield, S., Koller, O., Ney, H., & Bowden, R. (2018). Neural sign language translation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 7784–7793.
- Camgoz, N. C., Koller, O., Hadfield, S., & Bowden, R. (2020). Sign language transformers: Joint end-to-end sign language recognition and translation. In *Proceedings of CVPR*, 10023–10033.
- Kucherenko, T., Hasegawa, D., Henter, G. E., Kaneko, N., & Kjellstrom, H. (2020). Gesticulator: A framework for semantically-aware speech-driven gesture generation. In *Proceedings of the ACM International Conference on Multimodal Interaction*, 242–250.
- McDonald, J., Wolfe, R., Schnepf, J., Hochgesang, J., Jamrozik, D. G., Stumbo, M., ... & Berke, L. (2016). An automated technique for real-time production of lifelike animations of American Sign Language. *Universal Access in the Information Society*, 15(4), 551–566.

6. Kipp, M., Nguyen, Q., Heloir, A., & Matthes, S. (2011). Assessing the deaf user perspective on sign language avatars. In *Proceedings of the 13th International ACM SIGACCESS Conference on Computers and Accessibility*, 107–114.
7. Morrissey, S., & Way, A. (2005). An example-based approach to translating sign language. In *Proceedings of the 10th Annual Conference of the European Association for Machine Translation*,
8. Kacorri, H., Huenerfauth, M., Ebling, S., Patel, K., & Willard, M. (2015). Demographic and experiential factors influencing acceptance of sign language animation by deaf users. In *Proceedings of the 17th International ACM SIGACCESS Conference on Computers and Accessibility*, 203–210.
9. Kennaway, R. (2001). Synthetic animation of deaf signing gestures. In *Proceedings of the International Gesture Workshop: Gesture and Sign Language in Human-Computer Interaction*, 146–157. Springer.
10. Bragg, D., Koller, O., Bellard, M., Berke, L., Boudreault, P., Braffort, A., ... & Ladner, R. E. (2019). Sign language recognition, generation, and translation: An interdisciplinary perspective. In *Proceedings of the 21st International ACM SIGACCESS Conference on Computers and Accessibility*, 16–31.
11. Luzardo, G., Freire, M., Huenerfauth, M., & Kacorri, H. (2015). Toward making sign language synthesis usable: Evaluating and improving avatar quality. In *Proceedings of the 17th International ACM SIGACCESS Conference on Computers and Accessibility*, 211–218.
12. World Health Organization. (2021). *Deafness and hearing loss fact sheet*. Retrieved from <https://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss>
13. Sims, E. (2001). 3D sign language synthesis and animation. *VCOM3D Technical Report*, Orlando, FL.
14. spaCy Development Team. (2023). *spaCy: Industrial-strength Natural Language Processing in Python* (v3.6). Explosion AI. <https://spacy.io>
15. Three.js Authors. (2023). *Three.js — JavaScript 3D Library* (r158). <https://threejs.org>
16. ISLRTC. (2020). *Indian Sign Language Dictionary*. Indian Sign Language Research and Training Centre, Ministry of Social Justice and Empowerment, Government of India