Voice Note Translator — Architecture

Indic Language STT · Translation · TTS | Sarvam AI Pipeline

STT

What: Audio to Text via Sarvam Saaras v2

Why: Only API with 22 Indic langs + code-mix + auto detect at ~8% WER

Translate

What: Text to Translated Text via Sarvam

Why: Best accuracy for Indic language pairs; handles dialects & informal speech

TTS

What: Text to Speech via Sarvam Bulbul v2

Why: Most natural Indian voice; 10+ voices with pace/pitch control

Why We Use This Approach

Problem 1 — Sarvam Direct STT API limit: https://api.sarvam.ai/speech-to-text only supports audio up to **30 seconds**. For longer voice notes (real-world usage) we must use the **Batch Job API** instead, which uploads the full file to Azure Blob Storage and processes it asynchronously.

Problem 2 — Vercel API payload limit: Vercel serverless functions have a hard limit of **4.5 MB** per incoming request payload (applies to both Hobby and Pro plans). Sending audio files larger than 4 MB through a Next.js API route causes an immediate **413 / Request Entity Too Large** error. We cannot increase this limit on Vercel.

Solution — 2 MB direct-to-server chunked upload: For audio files > 4 MB the browser splits the audio into **2 MB chunks** and uploads each chunk **directly to the backend server** (not through Vercel). The server reassembles the chunks, then triggers the batch job flow.

Audio Size Decision Matrix

File Size	Duration	Flow Used
< 4 MB	≤ 30 s	Direct API (POST /speech-to-text) — sync, instant result
< 4 MB	> 30 s	Batch Job via Vercel API route — upload to Azure → poll for result
> 4 MB	> 30 s	**Chunked upload (2 MB pieces) directly to server → reassemble → Batch Job**

Pipeline Flows — Step by Step

Flow A — Short Audio (< 4 MB, ≤ 30 s)

1. **User Uploads Audio**
   Browser sends file via POST to /api/translate-audio (Next.js)
2. **Direct STT Call**
   API route calls POST https://api.sarvam.ai/speech-to-text — returns transcript instantly
3. **Translate**
   POST /translate (chunks < 2000 chars each) — returns translated text
4. **Text-to-Speech**
   POST /text-to-speech (chunks < 500 chars) — returns base64 WAV audio parts
5. **Merge & Return**
   mergeWavBase64(audioParts) — return { originalText, translatedText, audioBase64 }

Flow B — Long Audio via Batch Job (< 4 MB, > 30 s)

1. **User Uploads Audio**
   Browser sends file to /api/translate-audio (Next.js — < 4.5 MB OK)
2. **Initiate Batch Job**
   POST /speech-to-text/job/v1 (initiateBatchJob) → returns Job ID
3. **Get Azure Upload URL**
   POST /upload-urls for audio.wav → returns secure Azure pre-signed URL
4. **Upload to Azure Blob**
   Browser / server PUTs audio buffer directly to Azure Blob Storage
5. **Start Batch Job**
   POST /start (startBatchJob) → returns Job Started confirmation
6. **Return job ID to Frontend**
   API returns { jobId, status: 'processing' } to UI

■ Phase 2 — Background Polling (every 10 seconds)

Job State	Action

Processing	Return { status: 'processing' } — UI keeps polling
Failed	Return error message to user — stop polling
Completed	POST /download-urls → GET result.json from Azure → extract transcript

7 **Translate + TTS**
Same as Flow A steps 3–4: chunk → translate → TTS → merge WAV

8 **Final Response**
Return { status:'completed', originalText, translatedText, audioBase64 }

Flow C — Large Audio Chunked Upload (> 4 MB) ■ Vercel Limit Bypass

■ **Vercel hard limit = 4.5 MB per API request.** Sending audio > 4 MB through /api/translate-audio will crash with HTTP 413. Solution: browser chunks the file into 2 MB pieces and uploads each piece **directly to the backend server**, bypassing Vercel entirely.

1 **Detect Large File**
Frontend checks file.size > 4 MB before upload

2 **Split into 2 MB Chunks**
Browser slices audio: chunk = file.slice(offset, offset + 2MB)

3 **Upload Chunks to Server**
POST each chunk directly to backend server (NOT via Vercel) Headers: Content-Range, X-Total-Chunks, X-Chunk-Index, X-Upload-ID

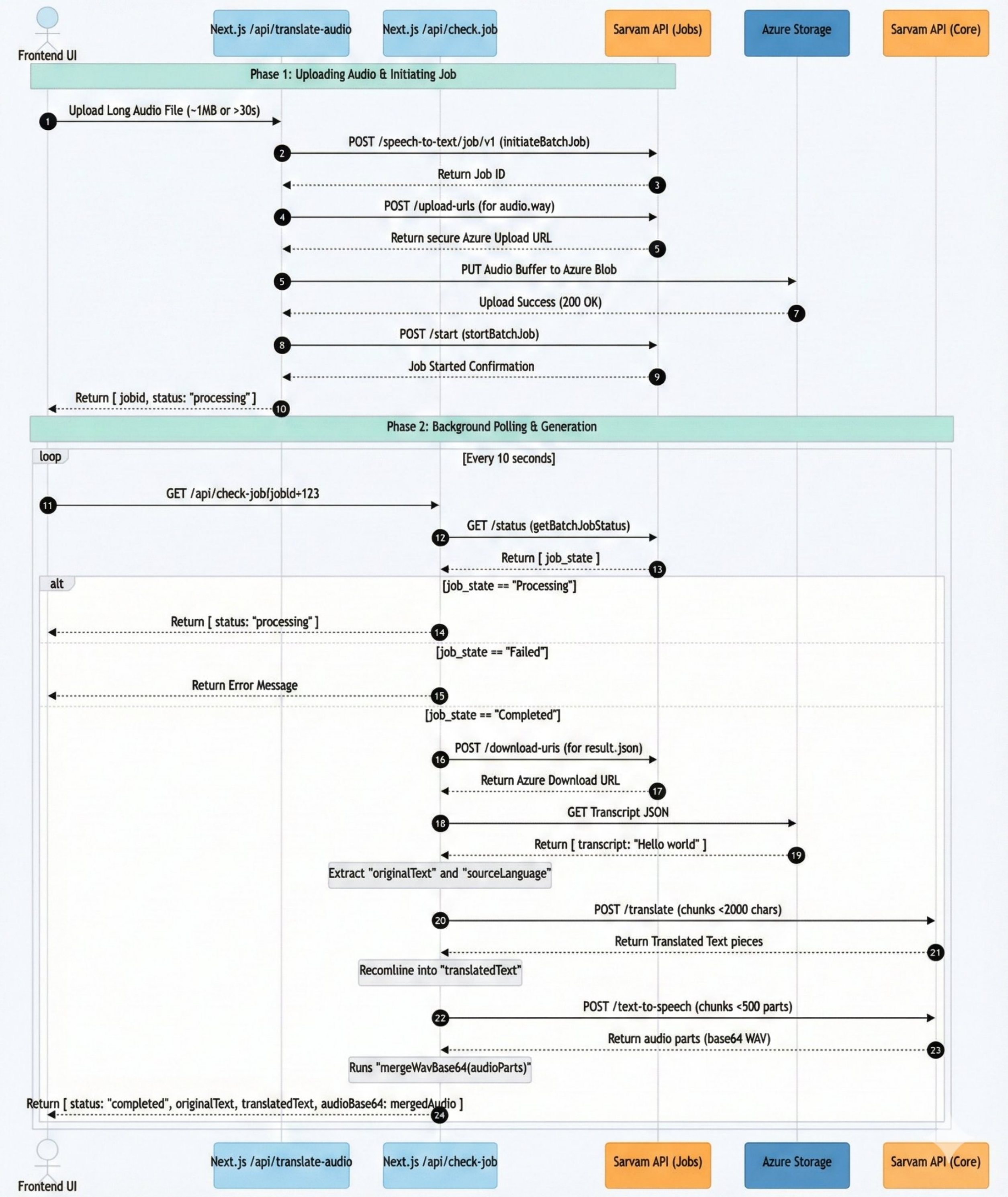
4 **Server Reassembles File**
Server appends chunks using upload ID. Once all received → full audio file ready

5 **Trigger Batch Job**
Server runs Flow B (Batch Job) — initiate → Azure upload → start → poll

6 **Return Result to Frontend**
Same final response: { originalText, translatedText, audioBase64 }

Workflow Diagram Reference

Visual architecture of the Batch Job pipeline


Frontend UI
Next.js /api/translate-audio
Next.js /api/check.job
Sarvam API (Jobs)
Azure Storage
Sarvam API (Core)
Phase 1: Uploading Audio & Initiating Job
1
Upload Long Audio File (~1MB or >30s)
2
POST /speech-to-text/job/v1 (initiateBatchJob)
3
Return Job ID
4
POST /upload-urls (for audio.way)
5
Return secure Azure Upload URL
5
PUT Audio Buffer to Azure Blob
7
Upload Success (200 OK)
8
POST /start (stortBatchJob)
9
Job Started Confirmation
10
Return [jobid, status: "processing"]
Phase 2: Background Polling & Generation
loop
[Every 10 seconds]
11
GET /api/check-job/jobld+123
12
GET /status (getBatchJobStatus)
13
Return [job_state]
alt
[job_state == "Processing"]
14
Return [status: "processing"]
[job_state == "Failed"]
15
Return Error Message
[job_state == "Completed"]
16
POST /download-uris (for result.json)
17
Return Azure Download URL
18
GET Transcript JSON
19
Return [transcript: "Hello world"]
Extract "originalText" and "sourceLanguage"
20
POST /translate (chunks <2000 chars)
21
Return Translated Text pieces
Recomline into "translatedText"
22
POST /text-to-speech (chunks <500 parts)
23
Return audio parts (base64 WAV)
Runs "mergeWavBase64(audioParts)"
24
Return [status: "completed", originalText, translatedText, audioBase64: mergedAudio]
Frontend UI
Next.js /api/translate-audio
Next.js /api/check-job
Sarvam API (Jobs)
Azure Storage
Sarvam API (Core)

The diagram above shows the complete Batch Job flow (Flow B): Phase 1 uploads audio to Azure Blob via Sarvam's presigned URL, Phase 2 polls for completion then runs translate and TTS to produce the final merged audio response.

API Quick Reference & Pricing

1 · Speech-to-Text (STT) — Sarvam Saaras v2

Field	Detail
Endpoint	POST https://api.sarvam.ai/speech-to-text
Batch Endpoint	POST https://api.sarvam.ai/speech-to-text/job/v1
Model	saaras:v2
Languages	22 Indian languages + English (auto-detect)
Max duration	30 sec (direct API) | Unlimited (batch job)
Max file size	25 MB (batch via Azure upload)
Formats	WAV, MP3, OGG, FLAC
WER — Hindi	~8–10% (best among all providers)
Cost	Rs. 15 per 10,000 chars | Free Rs. 1,000 credits
Code-switching	Hinglish, Tanglish, Banglish — excellent

2 · Translation — Sarvam Translate

Field	Detail
Endpoint	POST https://api.sarvam.ai/translate
Language pairs	All 22 Indian languages ↔ English
Max per call	1,000 chars — chunk longer texts before sending
Cost	Rs. 20 per 10,000 chars
Auto-detect	Yes — source_language_code can be 'auto'
Why Sarvam	Best dialect accuracy; handles informal & code-mixed text

3 · Text-to-Speech (TTS) — Sarvam Bulbul v2

Field	Detail
Endpoint	POST https://api.sarvam.ai/text-to-speech
Model	bulbul:v2 (or bulbul:v3 for premium)
Max per call	~500 chars — must chunk longer scripts
Output	Base64-encoded WAV audio — merge with mergeWavBase64()
Voices	10+ Indian voices (meera, arjun, etc.) — pick per language

Parameters	pace (speed), pitch, loudness, temperature
Cost v2	Rs. 15 per 10,000 chars
Cost v3	Rs. 30 per 10,000 chars (premium quality)
Why Sarvam	Most natural Indic intonation; no other provider comes close

Cost Estimate — 1 Minute Voice Note

Step	~Chars	Service	Cost
STT: 1 min audio	~900	Saaras v2	Rs. 1.35
Translation	~900	Sarvam Translate	Rs. 1.80
TTS output	~950	Bulbul v2	Rs. 1.43
Total per voice note	**~2,750**	**Full pipeline**	**Rs. 4.58**

Rs. 1,000 free credits covers ~218 one-minute voice notes through the full pipeline (STT + Translation + TTS) — plenty to build, test and demo the project.

All prices in Rs. (INR) as of 2024. Verify at sarvam.ai before production use. Get free API key at https://console.sarvam.ai — no credit card required.