Indian loan management app: multilingual voice architecture guide

A hybrid approach combining Azure Speech Services for 13 major languages with AI4Bharat's IndicConformer for all 22 scheduled languages is the only viable path to day-1 full coverage. No single commercial provider supports all 22 Indian languages — Azure covers 13, Google covers 14, and 8 languages (Kashmiri, Konkani, Maithili, Bodo, Dogri, Manipuri, Santali, Sanskrit) lack any commercial speech API. The zero-knowledge encryption requirement creates a fundamental tension with server-side RAG that demands a hybrid architecture: encrypted storage by default with explicit user-initiated "unlock" sessions for document Q&A. Total infrastructure cost for 1,000 MAU runs **$1,800–$4,800/month** depending on optimization choices, with Azure Speech STT ($2,500/month) dominating spend.

Speech services: no single provider covers all 22 languages

Azure Speech Services capabilities

Azure supports **13 of 22 scheduled Indian languages** for speech-to-text, making it the best single commercial option but far from complete:

Language	STT Locale	TTS Voices	Custom Speech	Fast Transcription
Hindi	hi-IN	**9** (5M, 4F)	Full	✅
Bengali	bn-IN	2 (1M, 1F)	Plain text	✅
Telugu	te-IN	2 (1M, 1F)	Audio+transcript	❌
Tamil	ta-IN	2 (1M, 1F)	Audio+transcript	❌
Marathi	mr-IN	2 (1M, 1F)	Audio+transcript	✅
Gujarati	gu-IN	2 (1M, 1F)	Plain text	❌
Kannada	kn-IN	2 (1M, 1F)	Plain text	❌
Malayalam	ml-IN	2 (1M, 1F)	Plain text	✅
Odia	or-IN	2 (1M, 1F)	Audio+transcript	❌
Punjabi	pa-IN	2 (1M, 1F)	Audio+transcript	❌
Assamese	as-IN	2 (1M, 1F)	Audio+transcript	❌
Urdu	ur-IN	2 (1M, 1F)	Audio+transcript	✅
Nepali	ne-NP	2 (1M, 1F)	Plain text	❌

Hindi stands out with 9 neural TTS voices, speaking style support (cheerful, newscast, empathetic via SSML), and the deepest customization options. Telugu and Tamil have solid STT with custom speech training capability microsoft but only 2 TTS voices each. The 9 unsupported languages — Kashmiri, Konkani, Maithili, Sanskrit, Sindhi, Bodo, Dogri, Manipuri, Santali — have **zero Azure speech coverage**.

Azure's language identification works in two modes: **at-start** (detects once in first 5 seconds, max 4 candidates, free) and **continuous** (detects throughout audio, max 10 candidates, Microsoft Learn $0.30/hour). Azure Docs GitHub The 10-candidate limit in continuous mode means you cannot detect all 22 languages simultaneously Microsoft Learn — you must pre-filter candidates, perhaps using text-based language detection or user profile data.

The comparison matrix across providers

Capability	Azure Speech	Google Cloud	Bhashini	AI4Bharat IndicConformer	OpenAI Whisper
Indian languages (of 22)	13	14	**22**	**22**	14
Hindi WER (benchmark)	~20%	~24%	Varies	**~13.6%**	~17-37%
Real-time streaming	✅	✅	⚠️ Limited	✅ (RNNT)	❌ (API only)
Auto language detection	✅ (10 max)	✅	⚠️	❌	✅ (unreliable)
SLA guarantee	99.9%	99.9%	❌ None	Self-managed	✅ (API)
Production readiness	Enterprise	Enterprise	PoC only	Needs GPU infra	Enterprise
Pricing (per minute)	$0.017	$0.016	Free/PoC	GPU cost only	$0.006
Code-mixing (Hinglish)	⚠️ Poor	⚠️ Poor	⚠️ Limited	⚠️ Limited	⚠️ Moderate

AI4Bharat's IndicConformer (600M parameters, MIT license) is the only model covering all 22 scheduled languages. AI4Bharat GitHub On the Vistaar benchmark, **IndicWhisper achieves 13.6% WER for Hindi** — outperforming Azure (20%), Google (24%), and vanilla Whisper (17%). GitHub These models require NVIDIA GPU infrastructure (minimum T4, recommended A100) and the NeMo framework. AI4Bharat's **IndicParler-TTS** provides the first open-source TTS for all 22 languages with style-controllable synthesis. X

Bhashini (Government of India's NLTM platform) aggregates ASR, TTS, NMT, and OCR models from IIT Madras, IISC, and CDAC covering all 22 languages. Gitbook API access is free for proof-of-concept. Gitbook Production use requires a paid arrangement with the Bhashini team. Real-world deployments exist (Federal Bank voice banking, NPCI voice payments, eShram portal), but **no formal SLA or uptime guarantee** exists for commercial use. The API documentation explicitly states usage "shall be for the purposes of PoC only." Gitbook

Recommended hybrid architecture for speech

The practical three-tier routing strategy:

Tier 1 — Azure Speech (13 languages, production-grade): Hindi, Bengali, Telugu, Tamil, Marathi, Gujarati, Kannada, Malayalam, Odia, Punjabi, Assamese, Urdu, Nepali. These get enterprise SLA, streaming support, and the JavaScript SDK for direct browser integration.

Tier 2 — Self-hosted IndicConformer on Azure GPU VM (9 remaining languages): Kashmiri, Konkani, Maithili, Sanskrit, Sindhi, Bodo, Dogri, Manipuri, Santali. Deploy on an Azure NC-series VM (NC6s_v3 with T4 at ~$0.90/hour or NCasT4_v3 at ~$660/month). The model supports streaming via RNNT decoder.

Tier 3 — Fallback path: OpenAI Whisper via Azure OpenAI ($0.006/minute) for batch processing or when primary services are unavailable. Brasstranscripts DevRain Not suitable for real-time due to lack of streaming. Vocafuse

For **TTS**, use Azure Neural TTS for the 14 supported languages (excellent quality, especially Hindi with style support), Microsoft Community Hub and IndicParler-TTS (self-hosted) for the remaining languages. Hugging Face

---

Zero-knowledge encryption meets RAG: the fundamental tradeoff

Client-side encryption architecture

The encryption stack should combine **Web Crypto API** (native, hardware-accelerated AES-256-GCM) with **libsodium.js** (Argon2id key derivation, streaming XChaCha20-Poly1305 for large files). Both have broad browser support. The key derivation follows the Bitwarden model:

```
User Password + Email (salt)
  → Argon2id (time=3, mem=64MB, parallelism=4) → 256-bit Master Key
  → HKDF expansion → 512-bit Stretched Master Key
  → Encrypts randomly-generated Symmetric Key (stored server-side, wrapped)
```

Envelope encryption is essential: each document gets a random 256-bit Document Encryption Key (DEK), the document is encrypted with AES-256-GCM using the DEK, and the DEK is wrapped with the user's master key using AES-KW. When a user changes their password, only DEKs need re-wrapping — documents remain untouched. Microsoft Learn

Key storage follows the **session-only** pattern: derive the master key from the password at login, hold it in memory (never persist to IndexedDB or localStorage), and clear all key references on logout or timeout. The extractable: false flag on CryptoKey objects prevents JavaScript-level extraction, MDN Web Docs but an attacker with filesystem access could still read IndexedDB data Thales — hence the session-only approach.

Recovery uses a **mandatory recovery key** generated at signup (displayed as Base32 groups like ABCDE-FGHIJ-KLMNO), which encrypts the user's master symmetric key. The encrypted-by-recovery-key blob is stored server-side. Optionally, **Shamir's Secret Sharing** (via @privy-io/shamir-secret-sharing, audited) can split

the recovery key into N-of-M shares for distributed backup. The server never sees any plaintext key material. Bitwarden

Resolving the RAG paradox

If documents are truly encrypted at rest with client-only keys, the server cannot perform text extraction, embedding generation, or vector search — the core operations RAG requires. Four approaches exist, ranked by practicality:

Approach A — Hybrid unlock (recommended for Phase 1): Zero-knowledge storage by default. When the user wants Q&A, they click "Enable Document Q&A," the browser decrypts documents locally, extracts text client-side using pdf.js, and sends plaintext to the backend over TLS for RAG processing. The server discards all plaintext after the session ends. This is how most encrypted-storage products handle search/AI features — Tresorit intentionally doesn't offer server-side content search, and Standard Notes performs all search client-side.

Approach B — Azure Confidential Computing (recommended for Phase 2): Deploy RAG processing on **Azure Confidential VMs** (DCasv5 series with AMD SEV-SNP). The user sends encrypted documents + a session key encrypted to the TEE's attestation key. The TEE decrypts, processes, and returns results — data is encrypted even in memory, and Azure operators cannot access it. Confidential GPU VMs (NCCadsH100v5 with H100) can run LLM inference within the TEE boundary. Azure Attestation cryptographically verifies TEE integrity before the client releases keys.

Approach C — Client-side RAG (future): Extract text in browser, generate embeddings using Transformers.js or ONNX Runtime Web, store encrypted embeddings. Feasible for small document collections (<100 docs) but impractical at scale due to browser compute limits and large model sizes (~100MB+ for embedding models).

The practical document upload flow: user selects PDF → browser reads as ArrayBuffer → generates random DEK via crypto.getRandomValues() → encrypts with AES-256-GCM → wraps DEK with master key → requests write-only SAS token from FastAPI → uploads encrypted blob directly to Azure Blob Storage Medium → stores wrapped DEK + encrypted metadata in Cosmos DB.

Multilingual document processing and RAG pipeline

OCR and text extraction

Azure Document Intelligence v4.0 LinkedIn supports **23+ Indic languages** for printed text OCR, auto-detecting languages in mixed-script documents (Hindi text + English numbers). The **Layout model** ($10/1,000 pages) is the right choice for loan documents — it extracts text, tables (EMI schedules), and document structure. For 1,000 users uploading ~5 pages/month, OCR costs approximately **$50/month**.

Key limitation: **handwritten Indian text is not supported** — only printed text works. For Telugu specifically, complex conjunct characters may reduce accuracy on low-quality scans. **PaddleOCR** (open-source, 94.5% on

OmniDocBench) GitHub serves as an excellent cost-effective backup for high-volume processing.

Embedding and retrieval strategy

Azure OpenAI text-embedding-3-large ($0.143/1M tokens) is the recommended embedding model. It consistently outperforms open-source alternatives in cross-lingual retrieval benchmarks, handles all major Indian scripts, and integrates directly with Azure AI Search. For budget-conscious deployments, **text-embedding-3-small** at $0.022/1M tokens provides strong performance at 85% lower cost.

GPT-4o's **o200k_base tokenizer** delivers dramatic improvements for Indian languages: **76.6% fewer tokens for Telugu, 74%+ fewer for Tamil, ~71% fewer for Hindi** compared to GPT-4. This makes Indian language processing 2.9–4.4x cheaper. microsoft GPT-4o-mini at $0.15/$0.60 per million input/output tokens Microsoft Learn handles loan Q&A excellently and should be the default model.

The retrieval architecture uses **Azure AI Search (Basic tier, ~$75/month)** Microsoft Learn with hybrid search combining BM25 keyword matching, vector search, and semantic re-ranking. Keyword search catches exact loan numbers, amounts, and dates that pure vector search misses. Azure AI Search includes built-in Indian language analyzers for Hindi and other languages.

For chunking multilingual documents, **paragraph-level splitting (300–500 tokens)** with 50–100 token overlap works best — sentence-boundary detection is unreliable across Indian languages due to varying punctuation conventions. Unicode NFC normalization must be applied before chunking. Avoid ML-based semantic splitters; they perform poorly on multilingual content. Use section-aware chunking aligned with document structure from the Layout model.

The complete RAG pipeline

Ingestion: PDF → Azure Document Intelligence (Layout) → text + tables + structure → GPT-4o-mini extracts structured loan fields (EMI, interest rate, tenure) into Cosmos DB → paragraph chunking with NFC normalization → text-embedding-3-large → Azure AI Search (vector + text index).

Query: User question → Azure AI Language detects language → query classification (structured vs. unstructured) → structured queries route directly to Cosmos DB metadata → unstructured queries go through hybrid retrieval (BM25 + vector + semantic reranker) → top 3–5 chunks assembled with structured metadata → GPT-4o-mini generates response in user's language → response cached in Redis (keyed on user_id + normalized_query_hash, 24-hour TTL).

The system prompt instructs GPT-4o-mini to always respond in the user's language, translate cross-language loan terms, include both numbers and their meaning for financial values, and cite specific document sections.

Browser voice recording and React architecture

Audio capture and streaming

The Azure Speech JavaScript SDK (microsoft-cognitiveservices-speech-sdk) Microsoft Learn handles microphone access internally via AudioConfig.fromDefaultMicrophoneInput(), manages WebSocket connections to Azure Speech Service, and provides real-time partial results through Recognizing events. Microsoft Learn For production security, generate temporary auth tokens server-side and pass them to the frontend — never expose Azure subscription keys in browser code. Microsoft Learn

Cross-browser audio format detection is critical. Build with Matija Safari iOS does not support audio/webm — it uses audio/mp4. Runtime detection at initialization ensures compatibility:

- Chrome/Firefox/Edge: audio/webm;codecs=opus ✅
- Safari iOS/macOS: audio/mp4 ✅
- WebM on Safari: ❌ Not supported

Azure Speech SDK requires **16 kHz, 16-bit, mono PCM** input. Vocafuse The SDK handles conversion when using its built-in microphone input. For custom MediaRecorder streams, the backend needs GStreamer for format conversion. Medium Microsoft Learn

React component structure

```
src/
├── components/
│   ├── chat/          # ChatInterface, MessageBubble, MessageList, TextInput
│   ├── voice/         # VoiceRecorder, AudioPlayer, WaveformVisualizer, MicPermissionPrompt
│   ├── language/      # LanguageIndicator, LanguageSelector
│   ├── documents/     # DocumentUploader (with encryption), DocumentList
│   └── common/        # LoadingSpinner, ErrorBoundary, OfflineIndicator
├── hooks/             # useVoiceRecording, useAudioPlayback, useSpeechRecognition, useWebSocket
├── services/
│   ├── api/           # speechService, chatService, documentService, authService
│   ├── websocket/     # wsClient, audioStreamHandler
│   └── encryption/    # documentEncryption (Web Crypto + libsodium.js)
├── stores/            # chatStore, voiceStore, authStore (Zustand)
├── types/             # TypeScript interfaces
└── utils/             # audioFormat detection, retryWithBackoff, serviceWorker
```

Zustand is the clear state management choice — no Provider wrapper needed Zustand (critical for persistent voice sessions), ~1KB bundle vs Redux Toolkit's ~11KB, and selective re-rendering prevents unnecessary

updates (Edstem) during real-time audio streaming. The FastAPI backend mirrors this with route modules for speech, chat, documents, and auth, plus a WebSocket endpoint for real-time speech streaming.

Achieving sub-3-second voice-to-voice latency

The end-to-end latency budget breaks down as:

Stage	Sequential	Streaming Optimized
Voice recording + upload	100–200ms	50–100ms
STT processing	100–350ms	90–200ms
Language detection	0ms (bundled with STT)	0ms
LLM processing (TTFB)	200–500ms	100–300ms
TTS processing (TTFB)	100–300ms	75–150ms
Audio download + playback start	50–100ms	30–50ms
Total	**550–1,450ms**	**345–800ms**

Sub-3 seconds is achievable and with streaming optimizations, the system can respond in under 1.5 seconds. The critical optimization is **pipeline streaming**: stream audio chunks to STT during recording, begin LLM inference on partial transcription, start TTS as soon as the first sentence is generated, and begin playback as the first audio chunk arrives. Deploy the backend in **Azure Central India region** to minimize network latency (Microsoft Learn) (~40–80ms round trip vs ~200–300ms for US regions).

Pre-cache common TTS responses (greetings, error messages, confirmations) as static audio files on Azure CDN. Use Voice Activity Detection to detect when the user stops speaking rather than relying on arbitrary timeouts.

Language detection: text is solved, speech is harder

Text language detection is fully solved for all 22 scheduled Indian languages. Azure AI Language Service detects all 22 scheduled languages in both native scripts and romanized form (for 12 languages). It also detects the specific script (ISO 15924) for texts of 12+ characters (microsoft) and returns confidence scores. For specialized Indian language detection, **AI4Bharat's IndicLID** supports all 22 languages in both native and romanized scripts (47 classes total), (GitHub) runs 10x faster than NLLB, (GitHub) and handles code-mixed text.

For **speech language detection**, Azure's continuous LID (max 10 candidates) combined with text-based detection creates a robust pipeline: use at-start LID with the 4 most probable languages (based on user profile or

region), then refine with text-based detection on the transcription output. Azure Docs GitHub The system cannot detect mid-sentence code-switching GitHub — Hinglish and Tenglish remain challenging for all commercial providers. Qxf2 BLOG **Shunya Labs' "Zero Codeswitch"** is the most promising specialized solution for code-mixed Indian speech, claiming sub-100ms latency on CPUs and support for 40+ Indian languages including Hinglish. CXO VOICE

Cost projection for 1,000 monthly active users

Assumptions

- 20 voice interactions per user per day, average 15 seconds each
- Monthly audio volume: **2,500 hours**
- Document uploads: ~5 pages per user per month

Monthly cost breakdown

Component	Service	Unoptimized	Optimized
Speech-to-text	Azure Speech (real-time)	**$2,500**	$1,250 (commitment tier)
Text-to-speech	Azure Neural TTS	$900	$630 (30% cached)
LLM (Q&A + chat)	Azure OpenAI GPT-4o	$1,200	$175 (GPT-4o-mini)
Document OCR	Azure Doc Intelligence	$50	$50
Embeddings	text-embedding-3-large	$15	$15
Vector search	Azure AI Search (Basic)	$75	$75
Database	Cosmos DB (serverless)	$5	$5
Blob storage	Azure Blob	$5	$5
Compute	Azure Container Apps	$150	$150
GPU VM (Tier 2 languages)	NC6s_v3	$660	$660
CDN + bandwidth	Azure CDN	$60	$40
Total		**~$5,620**	**~$3,055**

Azure Speech STT dominates cost. The single biggest optimization is the commitment tier (50K hours/year at $0.50/hour, saving 50%). (Naitive) Switching to GPT-4o-mini saves ~$1,000/month with minimal quality loss for loan Q&A. (Microsoft Learn) If the GPU VM for Tier 2 languages isn't needed immediately (defer the 9 low-resource languages), optimized cost drops to approximately **$2,400/month**.

An alternative approach using **Deepgram** for STT ($0.0043/minute (Naitive) = ~$645/month) instead of Azure would cut the total to approximately **$1,800/month**, though it supports fewer Indian languages.

UX patterns for voice-first Indian users

Voice-first design for low-literacy users requires the app to **speak first, listen second**. The system should proactively guide users with audio prompts in their detected language: "Apna sawaal boliye" (Speak your question). Key UX principles: large touch targets (minimum 64×64px for the mic button), bottom-positioned controls in the thumb-friendly zone, pulsing waveform animation during recording (using (react-audio-visualize) which accepts MediaRecorder directly), (npm) color-coded status (green = recording, amber = processing, blue = playing), and minimal text with maximum audio.

For poor connectivity (common in rural India), implement retry with exponential backoff (1s → 2s → 4s → max 30s with jitter), an offline message queue persisted to IndexedDB via service workers, compressed Opus audio format to minimize bandwidth, and optimistic UI that shows "sending..." immediately. Target **<500KB initial bundle** and design for 320px minimum width (low-end Android phones). Test on 2GB RAM devices with 3G connections.

The onboarding flow should include: welcome screen with auto-playing local language greeting → interactive microphone permission request with voice explanation → test recording where the app repeats the user's name back → visual confirmation of detected language → brief audio tutorial.

Phase-wise implementation plan

Phase 1 — Text chat MVP (4 weeks): React+Vite+TS scaffold, FastAPI backend, JWT auth, Azure Cosmos DB. Text chat in Hindi, English, Telugu with GPT-4o-mini. Basic document upload with client-side AES-256-GCM encryption. Deploy on Azure Container Apps in Central India. *Deliverable: working encrypted document storage + text Q&A in 3 languages.*

Phase 2 — Voice I/O + RAG (6 weeks): Voice recording component with waveform visualization. Azure Speech STT/TTS integration (streaming via WebSocket). Language auto-detection. RAG pipeline: Azure Document Intelligence → chunking → embeddings → Azure AI Search → GPT-4o-mini. End-to-end voice-question-to-voice-answer flow. *Deliverable: full voice+text chat with document Q&A for Hindi, English, Telugu.*

Phase 3 — Major language expansion (4 weeks): Add Tamil, Bengali, Marathi, Kannada, Gujarati, Malayalam, Punjabi. Performance optimization (streaming pipeline, TTS caching, CDN). Offline resilience (service workers, retry queue). *Deliverable: 13 Azure-supported languages, production-hardened.*

Phase 4 — Full 22-language coverage (4 weeks): Deploy IndicConformer on GPU VM for 9 remaining languages. IndicParler-TTS for Tier 2 language synthesis. Code-mixing support evaluation. Load testing, security audit, A/B testing framework. *Deliverable: all 22 languages, production-ready.*

If "all 22 languages on day 1" proves too ambitious, **prioritize Hindi + English + Telugu in Phase 1–2**, expand to the 10 most-spoken remaining languages in Phase 3, and treat the 9 low-resource languages as Phase 4. The 13 Azure-supported languages cover approximately **95% of India's population** (Microsoft Community Hub) — the 9 remaining languages, while constitutionally scheduled, have significantly smaller speaker populations.

Testing strategy for multilingual voice accuracy

Use **AI4Bharat's IndicVoices-R** dataset (1,704 hours across all 22 languages) (Hugging Face) as the primary benchmark. (AI4Bharat) (GitHub) For each language, measure Word Error Rate (WER) against held-out test sets. Set CI/CD thresholds: **fail the build if Hindi WER exceeds 15%, Telugu exceeds 20%, or English exceeds 10%**. Run automated WER tests on ~100 samples per language on every deployment using pytest fixtures that call the STT API with known audio and compare transcriptions.

Manual testing requires 2–3 native speakers per language covering accent variation, code-mixing (Hinglish, Tenglish), noisy environments (street noise, crowd), and domain-specific vocabulary (loan terms like EMI, foreclosure, collateral). Implement an A/B testing framework using feature flags to route percentages of users to different STT providers, comparing WER, latency, and user satisfaction (thumbs up/down feedback).

Conclusion

This architecture resolves the three hardest constraints through pragmatic hybrid approaches. For **language coverage**, Azure handles 13 major languages with enterprise SLA while self-hosted IndicConformer fills the 9-language gap — the only path to genuine 22-language coverage today. (Hugging Face) For **zero-knowledge encryption**, the envelope encryption pattern with Argon2id key derivation provides Bitwarden-grade security, with explicit user-initiated Q&A sessions bridging the encryption-RAG paradox until Azure Confidential Computing matures into the standard processing layer. For **latency**, the streaming pipeline (concurrent STT → LLM → TTS) achieves sub-1.5-second voice-to-voice response times, well within the 3-second target. The most impactful cost decision is choosing GPT-4o-mini over GPT-4o — it saves ~$1,000/month with negligible quality loss (Microsoft Learn) for extractive loan Q&A. The most impactful architectural decision is deploying in Azure Central India region, which cuts network latency by 60–75% for the target user base.