Survey Report

In the evolving field of AI, Sarvam-M is an Indian Large Language Model (LLM) built on top of the open-source Mistral Small. It offers bilingual and code-mixed support, logical reasoning, and long-context understanding tailored for Indian languages. Alongside Sarvam-M, the team has developed two major voice-based tools:

- Saarika: A multilingual Speech-to-Text model optimized for real-time transcription.

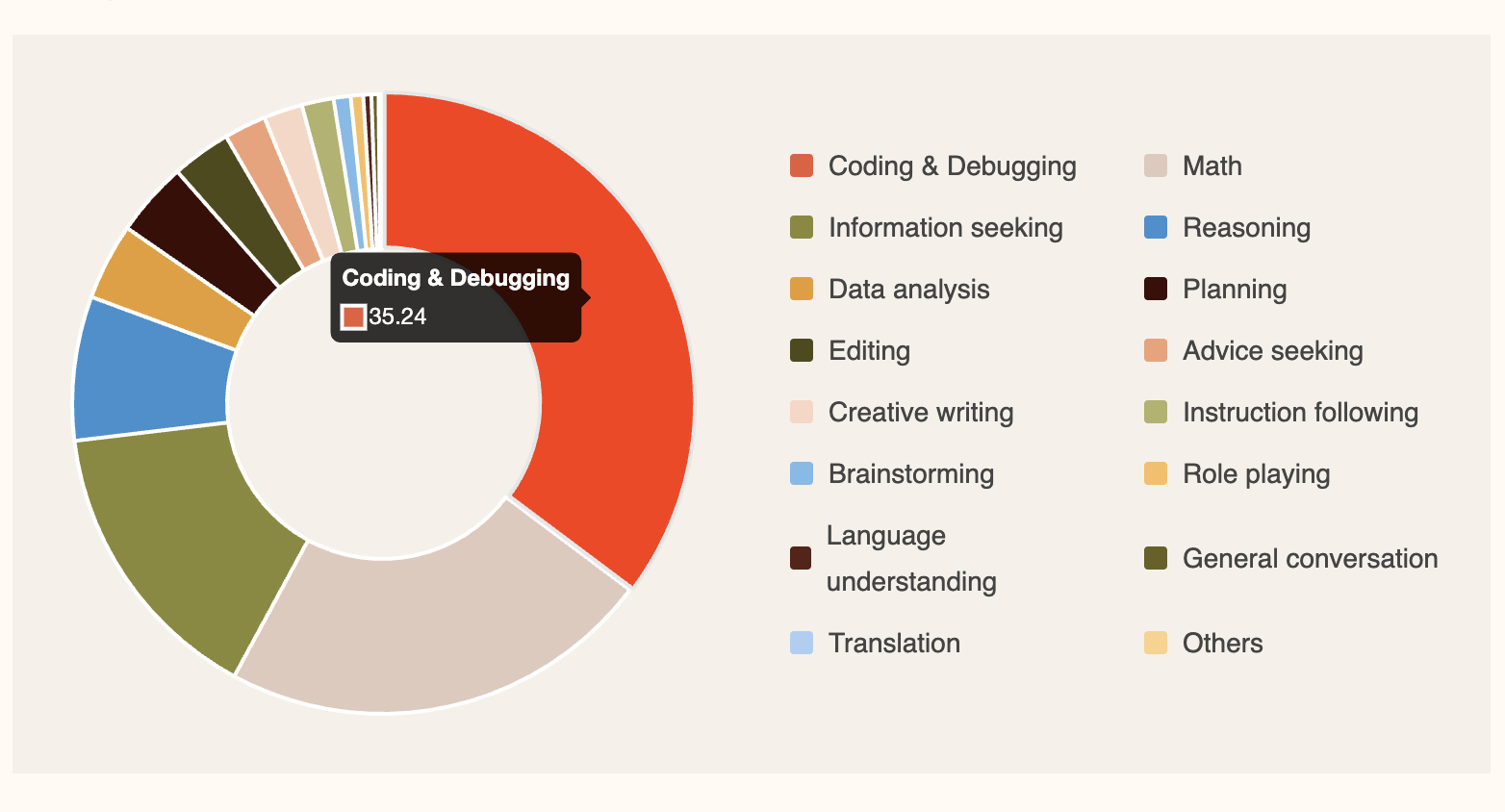
- Bulbul: A Text-to-Speech model with natural Indian voices and prosody control.

This survey was conducted to evaluate awareness, usage experience, and interest .

Sarvam M TTS and STT Model of sarvam

One of the key players selected for this effort is **Sarvam AI,** which recently launched **Sarvam-M** a **24-billion parameter hybrid language model** built on top of **Mistral Small** and optimized for Indian languages, coding, and mathematics.

Sarvam-M is open-weight, API-accessible, and represents a milestone in creating a **multilingual LLM** for India. The model is trained in **10 Indian languages** covering over 70% of the population’s mother tongues and supports use cases like conversational AI, educational tools, and machine translation.

Sarvam AI has also released **Bulbul**, a multilingual **Text-to-Speech (TTS)** model that supports **11 Indian languages** with authentic regional accents and **Saarika**, a Speech-to-Text model for real-time applications. 

Reference:https://analyticsindiamag.com/ai-news-updates/sarvam-launches-sarvam-m-a-24b-open-weights-model-on-top-of-mistral/

**Sarvam** **offers 4 AI** models designed to serve diverse Indian language needs:

**Mayura:** A multilingual translation model that supports English and 11 Indian languages with automatic language detection, preserving meaning and context.

**Saras:** A speech-to-text model that transcribes audio and translates between Indian languages in a single pipeline.

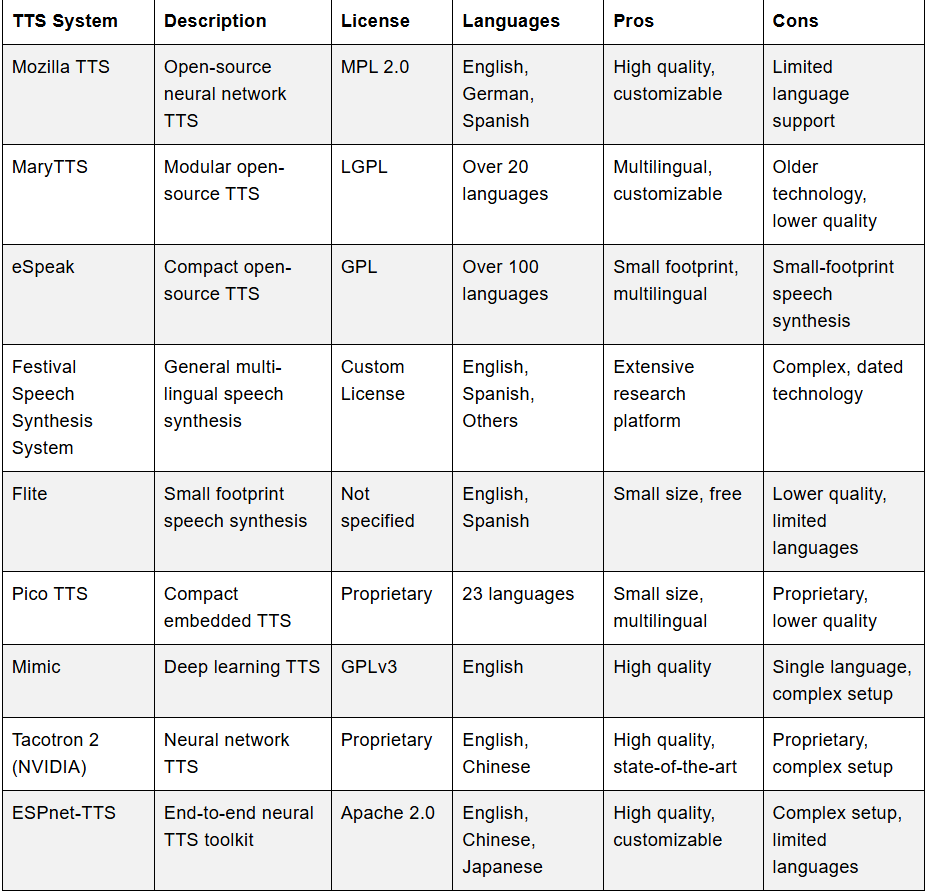
**Saarika:** A high-accuracy text-to-speech model for multiple Indian languages, offering clear and intelligible output.

**Bulbul:** The TTS backbone of Sarvam, Bulbul offers human-like prosody, multiple voice personalities, and real-time synthesis tailored for Indian accents and languages.

**Text-to-Speech (TTS)** Technology

TTS engines work by processing text input and generating synthetic speech output that resembles human speech.

**10 open source:**



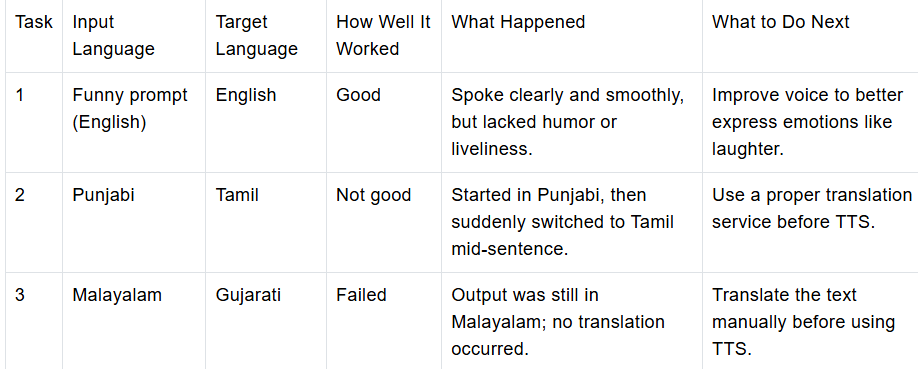
**What is Special About Bulbul-V2?**

**Voice Control**: Fine-grained control over pitch (-1 to 1), pace (0.3 to 3), and loudness (0.1 to 3)

**Sample Rate Options**: Multiple sample rates: 8kHz, 16kHz, 22.05kHz, 24kHz.

**Text Preprocessing**: Smart normalization of numbers, dates, and mixed-language text

**Language Support**: Support for 11 Indian languages with BCP-47 codes.



Reference: [1]

Bulbul-V2’s fast and natural text-to-speech capabilities make it a good fit in many real-world cases where inter-language conversion is not involved.

**Assistive Technology:** TTS transforms text into speech for visually impaired users

**E-Learning and Content Creation:** TTS models can be used to make audiobooks, other educational materials, and voice-over for videos.

**Language Translation & Localization:** TTS technology supports the creation of localized content. It enables real-time translation for applications

**Speech-to-Text (STT) Model**

the Saarika Model for Speech-to-Text (STT) .It covers both short and long audio transcription, including how to split large files into chunks and transcribe them using the real-time API.

**Usage for STT**

The Saarika model can be used for converting speech to text across different scenarios.

It supports basic transcription, code-mixed speech, and automatic language detection for Indian languages.

**1. Basic Usage**

Basic transcription with a specified language code.

Ideal for single-language audio with clear speech and minimal noise.

if audio\_file\_path:

with open(audio\_file\_path, "rb") as audio\_file:

response = client.speech\_to\_text.transcribe(

file=audio\_file,

model="saarika:v2.5",

language\_code="en-IN" )

print("Transcription Response:")

print(response)

else:

print("No audio file found. Transcription aborted.")

Reference:[2]

**2.Code-Mixed Speech**

Handles mid-sentence language switches intelligently.

Perfect for conversational speech in Indian multilingual settings.

if audio\_file\_path:

with open(audio\_file\_path, "rb") as audio\_file:

response = client.speech\_to\_text.transcribe(

file=audio\_file,

model="saarika:v2.5" )

print(response)

else:

print("No valid audio file found.")

Reference:[3]

**3. Automatic Language Detection**

It detect the spoken language automatically.

Useful when input language is unknown or for multilingual speech.

if audio\_file\_path:

with open(audio\_file\_path, "rb") as audio\_file:

response = client.speech\_to\_text.transcribe(

file=audio\_file,

model="saarika:v2.5",

language\_code="unknown" )

print(response)

else:

print("No valid audio file found.")

Reference:[4]

**4. Handling Long Audio Files**

If audio file exceeds the 30-second limit supported by the real-time transcription API, you must split it into smaller chunks for accurate and successful transcription. These smaller segments are then transcribed individually using the real-time API, and the results are stitched back together to form the final transcript. This function splits a long .mp3 or .wav audio file into smaller chunks (default: 29 seconds) using FFmpeg. It ensures each segment remains within the real-time API’s 30-second limit and stores them in the specified output directory.

Define the **transcribe\_audio\_chunks** Function This function takes the list of chunked audio file paths and uses the Saarika real-time API to transcribe each one individually. It collects all partial transcriptions and combines them into a single, complete transcript.

Putting It All Together Call the split\_audio\_ffmpeg() function first to break the audio into chunks, and then pass those chunks to transcribe\_audio\_chunks() for transcription. This two-step process ensures large audio files are handled smoothly using the real-time API. ref:[5]

**How to use sarvam AI API**

1. Create an API Key

2. Set Up Your Environment

**Windows**: $env:SARVAM\_API\_KEY="your\_api\_key\_here"

**macOS/linux**: export SARVAM\_API\_KEY="your\_api\_key\_here"

3.Install the SDK

$ pip install sarvamai

4.To **make a first API** call in python

from sarvamai import SarvamAI

client = SarvamAI(

api\_subscription\_key="YOUR\_API\_KEY",)

response = client.text.translate(

input="Hi, My Name is Vinayak.",

source\_language\_code="auto",

target\_language\_code="gu-IN",

speaker\_gender="Male")

print(response)

5. sarvam AI API

**Speech to text translate**

from sarvamai import SarvamAI

client = SarvamAI(

api\_subscription\_key="YOUR\_SARVAM\_API\_KEY",

)

response = client.speech\_to\_text.translate(

file=open("audio.wav", "rb"),

model="saaras:v2.5"

)

print(response)

reference:[6]

**Text to speech translation**

from sarvamai import SarvamAI

client = SarvamAI(

api\_subscription\_key="YOUR\_SARVAM\_API\_KEY",

)

response = client.text\_to\_speech.convert(

text="Hello, how are you?",

target\_language\_code="hi-IN",

)

print(response)

reference:[7]

Reference

[1] <https://www.analyticsvidhya.com/blog/2025/05/bulbul-v2-by-sarvam/#h-exploring-sarvam-s-model>

[2] <https://docs.sarvam.ai/api-reference-docs/cookbook/starter-notebooks/stt-api-tutorial#4-saarika-v25-usage-for-stt>

[3] <https://docs.sarvam.ai/api-reference-docs/cookbook/starter-notebooks/stt-api-tutorial#42-code-mixed-speech>

[4] :<https://docs.sarvam.ai/api-reference-docs/cookbook/starter-notebooks/stt-api-tutorial#43-automatic-language-detection>

[5] <https://docs.sarvam.ai/api-reference-docs/cookbook/starter-notebooks/stt-api-tutorial#53-putting-it-all-together>

[6] <https://docs.sarvam.ai/api-reference-docs/getting-started/quickstart#sarvam-ai-apis>

[7] <https://docs.sarvam.ai/api-reference-docs/getting-started/quickstart#sarvam-ai-apis>