Requirements Specification for Live Call Translation System

Introduction

This document outlines the functional and non-functional requirements for a live call translation system designed to support real-time, bidirectional translation between all international languages and specific African languages, with optional voice cloning, live billing, AI conversation capabilities, and live translation captions for videos and video call bots for platforms like Zoom and Google Meet. The system targets tourists, travelers to Africa, and global users, leveraging existing open-source models and modular development across three teams: Data Science (Machine Translation, LLM Integration, ASR, TTS), Backend, and Mobile App. The system uses a pay-as-you-use billing model and is designed for scalability to support a billion users.

Customer Segments

The system caters to the following customer segments:

- **Tourists and Travelers to Africa**: Need real-time translation for communication with locals in African languages (e.g., Swahili, Yoruba, Amharic) and international languages (e.g., English, Mandarin, Spanish).
- **Business Professionals**: Require translation for multilingual customer support, meetings, or negotiations, especially in African markets.
- **Humanitarian Organizations**: Use the system for aid helplines to communicate with diverse populations in Africa.
- **Content Creators**: Utilize live translation captions for videos to reach multilingual audiences.
- **Remote Teams**: Integrate translation bots into Zoom/Google Meet for seamless multilingual meetings.
- **General Public**: Engage with AI in their native language for customer service or personal assistance.

Functional Requirements

The system is divided into modules assigned to each team, leveraging existing open-source models (e.g., Whisper, NLLB-200, Coqui XTTS) and external voice cloning technologies.

Data Science Team Requirements

The Data Science team is responsible for integrating and optimizing open-source models for Automatic Speech Recognition (ASR), Machine Translation (MT), Text-to-Speech (TTS), and LLM integration for AI conversations and text refinement. Voice cloning can be handled by external models like Coqui XTTS.

1. Automatic Speech Recognition (ASR)

- **Functionality**: Transcribe audio from live calls or video meetings in real time, supporting all international languages and African languages (e.g., Swahili, Yoruba, Amharic, Hausa, Igbo, Zulu, Luganda, Xhosa, Kinyarwanda).
- **Model**: Use Whisper (large-v3) for its support of over 100 languages, including African languages, and streaming mode for low-latency transcription.
    - Example: `whisper --model large-v3 --language auto --streaming`
- **Tasks**:
    - Integrate Whisper for real-time audio transcription from Asterisk or video platforms.
    - Fine-tune Whisper for African languages with limited support (e.g., Luganda, Xhosa) using datasets like Mozilla Common Voice or Masakhane.
    - Ensure language detection with Whisper or VoxLingua107 for automatic source language identification.
    - Optimize for latency (<0.3s per transcription) using streaming APIs and GPU acceleration.
- **Output**: Transcribed text with identified source language, sent to the MT module.

2. Machine Translation (MT)

- **Functionality**: Translate text between any pair of international and African languages (e.g., English to Yoruba, Swahili to Mandarin) in real time.
- **Model**: Use NLLB-200-distilled-600M for its support of over 200 languages, including low-resource African languages, optimized for efficiency.
    - Example: `python translate.py --source_lang yo --target_lang en --text "Hello, how are you?"`
- **Tasks**:
    - Integrate NLLB-200-distilled-600M for text translation, using CTranslate2 for quantization (int8) to reduce latency (~0.125s).
    - Fine-tune NLLB-200 for unsupported African languages (e.g., Akan) using FLORES-200 or Masakhane datasets.
    - Support bidirectional translation for live calls (e.g., Yoruba to English and English to Yoruba simultaneously).
    - Ensure compatibility with live captioning for video content.
- **Output**: Translated text sent to the TTS module or captioning system.
-

3. Text-to-Speech (TTS)

- **Functionality**: Synthesize translated text into speech with emotional expression, supporting all international and African languages.
- **Model**: Use Coqui TTS with Fairseq models (e.g., Tacotron 2, FastSpeech 2) for 1100+ languages, including African languages like Akan, Hausa, Swahili, Yoruba.
    - Example: `tts --model_name yo/fairseq/vits --text "Hello, how are you?" --out_path output.wav`
- **Tasks**:
    - Integrate Coqui TTS for real-time speech synthesis with streaming inference (<0.2s latency).
    - Fine-tune TTS models for African languages with limited support using datasets like BibleTTS or AfroDigits.
    - Adjust prosody parameters to convey emotional nuances (e.g., tone, intonation).
    - Integrate with Asterisk for call audio playback and with video platforms for bot integration.
- **Output**: Synthesized audio in the target language, preserving emotional context.

4. Voice Cloning

- **Functionality**: Clone the original speaker's voice for personalized TTS output, using external voice cloning technology.
- **Model**:
- **Tasks**:
    - Capture a short audio sample (3-6s) at call initiation for each speaker.
    - Integrate Coqui XTTS to clone voices, caching profiles for reuse to minimize latency.
    - Optimize XTTS for real-time use (<0.2s latency), balancing quality and speed.
    - Provide an option to toggle voice cloning on/off to manage latency.
- **Output**: Cloned voice audio for TTS, enhancing user experience.

5. LLM Integration for AI Conversations

- **Functionality**: Enable users to interact with an AI agent in their native language for customer service or assistance, with text refinement for natural translations.
- **Model**: Use Llama 3 (fine-tuned for low-resource African languages) for translation enhancement and Grok (or equivalent open-source LLM) for conversational responses.
- **Tasks**:
    - Fine-tune Llama 3 for African language translation using Masakhane or FLORES-200 datasets to improve NLLB-200 output.
    - Integrate Grok for generating conversational responses (e.g., "I see, let me check that for you") in the user's language.
    - Use Grok to refine translated text for fluency before TTS synthesis.

    - Support real-time AI conversations via Asterisk or mobile app interfaces.
- **Output**: Conversational text or refined translations for TTS or captioning.

Backend Team Requirements

The Backend team is responsible for telephony, infrastructure, billing, and integration with video platforms (Zoom, Google Meet) and captioning services.

1. Telephony System

- **Functionality**: Manage incoming/outgoing calls, route audio streams, and integrate with the translation pipeline.
- **Tool**: Use Asterisk for call handling, with Kamailio/OpenSIPS for SIP proxying to scale call volumes.
- **Tasks**:
    - Configure Asterisk to handle live calls, using AEAP to stream audio to the translation service via RTP.
    - Set up Kamailio/OpenSIPS to distribute calls across multiple Asterisk instances for scalability.
    - Implement IVR or automatic language detection to initiate translation.
    - Ensure bidirectional audio streaming for real-time communication.
- **Output**: Audio streams sent to the translation pipeline and translated audio played back to callers.

2. Distributed Infrastructure

- **Functionality**: Support scalability for millions of concurrent calls and global deployment.
- **Tools**: Use Docker for containerization, Kubernetes for orchestration, and Kafka for audio stream distribution.
- **Tasks**:
    - Containerize the translation service (ASR, MT, TTS) using Docker.
    - Deploy with Kubernetes for auto-scaling based on call volume and CPU usage.
    - Configure Kafka to distribute audio streams across translation services.
    - Deploy edge servers in multiple regions (e.g., Africa, Asia, Europe) with geo-routing for low latency.
- **Output**: Scalable infrastructure supporting a billion users with minimal latency.

3. Live Billing System

- **Functionality**: Implement a pay-as-you-use billing model based on call duration or translation usage.
- **Tasks**:
    - Integrate a billing service (e.g., Stripe, Paystack for African markets) to track usage and charge credits.

    - Monitor call duration and translation requests in real time, notifying users of credit usage.
    - Provide APIs for the mobile app to display billing status and allow credit purchases.
    - Ensure secure payment processing and compliance with regional regulations (e.g., GDPR, African data laws).
- **Output**: Real-time billing updates and payment notifications.

4. Video Call Bot Integration

- **Functionality**: Develop bots for Zoom and Google Meet to provide real-time translation during video calls.
- **Tasks**:
    - Create bots using Zoom SDK or Google Meet API to intercept audio streams.
    - Integrate with the translation pipeline (ASR, MT, TTS) for real-time speech-to-speech translation.
    - Support voice-to-chat RTT, where agents receive translated text in their preferred language.
    - Ensure compatibility with WebRTC for low-latency audio/video streaming.
- **Output**: Translated audio or text overlays in Zoom/Google Meet meetings.

5. Live Translation Captions

- **Functionality**: Generate real-time captions for videos or live calls, supporting all languages.
- **Tasks**:
    - Integrate the translation pipeline to generate text captions from transcribed audio.
    - Provide APIs to overlay captions on videos or live streams (e.g., using FFmpeg or WebVTT).
    - Support customizable caption styles (font, color, position) for accessibility.
    - Ensure captions are synchronized with audio (<0.5s delay).
- **Output**: Real-time captions in the target language for videos or calls.

Mobile App Team Requirements

The Mobile App team is responsible for developing a user-friendly app for tourists, travelers, and other users, supporting live calls, AI conversations, video call integration, and billing.

1. Live Call Interface

- **Functionality**: Allow users to make live calls with real-time translation in their chosen languages.
- **Tasks**:
  - Develop a UI for initiating calls, selecting source/target languages, and toggling voice cloning.
  - Integrate with Asterisk via WebRTC for audio streaming.
  - Display real-time call status and translation progress.
  - Support offline language detection for poor network conditions.
- **Output**: A seamless call experience with translated audio playback.

2. AI Conversation Interface

- **Functionality**: Enable users to interact with an AI agent in their native language for assistance (e.g., travel tips, customer support).
- **Tasks**:
  - Create a chatbot-like UI for voice or text input/output, integrated with Grok for responses.
  - Support voice input/output using the ASR and TTS pipeline.
  - Allow language selection for AI interactions.
  - Provide pre-rendered filler responses (e.g., "Just a moment") to mask latency.
- **Output**: Natural, multilingual AI conversations.

3. Video Call Integration

- **Functionality**: Allow users to join Zoom/Google Meet calls with translation support via the app.
- **Tasks**:
  - Integrate Zoom/Google Meet SDKs for joining video calls.
  - Display translated captions or play translated audio during calls.
  - Provide controls to toggle between voice-to-voice and voice-to-chat RTT.
  - Ensure compatibility with mobile device hardware (e.g., microphones, speakers).
- **Output**: Translated video call experience with captions or audio.

4. Billing and Credit Management

- **Functionality**: Allow users to manage credits, view usage, and make payments.
- **Tasks**:
  - Develop a UI for purchasing credits and viewing billing history.
  - Integrate with the backend billing API for real-time updates.

    - Support multiple payment methods (e.g., mobile money for African users, credit cards).
    - Provide notifications for low credit balances.
- **Output**: Transparent billing and payment management.

5. Accessibility and Localization

- **Functionality**: Ensure the app is usable by diverse audiences, including tourists and African users.
- **Tasks**:
    - Support multilingual UI (e.g., English, Swahili, French, Yoruba).
    - Optimize for low-bandwidth environments common in African regions.
    - Include accessibility features (e.g., screen reader support, high-contrast mode).
    - Provide tutorials for first-time users (e.g., tourists unfamiliar with African languages).
- **Output**: A user-friendly, accessible app for global users.

Non-Functional Requirements

1. **Latency**: Achieve sub-1-second end-to-end latency for live call translation (ASR: ~0.3s, MT: ~0.125s, TTS: ~0.2s, network: ~0.375s).
2. **Scalability**: Support millions of concurrent calls for a billion users using Kubernetes and edge computing.
3. **Reliability**: Ensure 99.9% uptime with monitoring tools like Prometheus and Grafana.
4. **Security**: Secure audio streams and billing data with encryption (e.g., TLS, HTTPS) and compliance with data protection regulations.
5. **Language Coverage**: Support all international languages and key African languages (e.g., Swahili, Yoruba, Amharic, Hausa, Igbo, Zulu, Luganda, Xhosa, Kinyarwanda).
6. **Compatibility**: Ensure compatibility with iOS, Android, Zoom, Google Meet, and web platforms.

Implementation Steps

1. **Proof of Concept**:
    - Test with language pairs (e.g., Swahili to English, Yoruba to French) using sample audio or live calls.
    - Measure latency, translation accuracy, and voice quality using tools like Wandb or cProfile.
2. **Integration**:
    - Data Science: Integrate Whisper, NLLB-200, Coqui TTS/XTTS, and fine-tuned Llama 3/Grok.
    - Backend: Set up Asterisk, Kubernetes, Kafka, and billing APIs.
    - Mobile App: Develop UI/UX for calls, AI interactions, and video integration.

3. **Scaling**:
    - Deploy edge servers in Africa, Asia, Europe, etc., using Kubernetes for auto-scaling.
    - Test with high call volumes to ensure scalability.
4. **Monitoring**:
    - Use Prometheus/Grafana to track latency, call quality, and system performance.
    - Implement user feedback mechanisms to improve accuracy and usability.

Challenges and Mitigations

- **Latency**: Use streaming APIs, model quantization (CTranslate2), and edge computing to achieve sub-1-second latency.
- **Language Coverage**: Fine-tune models for unsupported languages using Common Voice or FLORES-200 datasets.
- **Voice Cloning**: Optimize Coqui XTTS for speed and test with diverse audio samples to ensure quality.
- **Cost**: Use cloud providers (AWS, GCP) or on-premises servers with cost monitoring to manage expenses.
- **User Experience**: Provide clear instructions and filler responses to mask latency and enhance usability.