Architecting the Code-Switched Data Factory: A Systems Approach to End-to-End Speech Translation for Vietnamese-English

Quang Chien

November 25, 2025

Executive Summary

The engineering of an End-to-End (E2E) Speech Translation (ST) system for Vietnamese-English Code-Switching (CS) represents a formidable "0 to 1" challenge in the domain of computational linguistics and machine learning operations (MLOps). Unlike standard monolingual tasks supported by curated corpora like LibriSpeech or Common Voice, code-switching—the fluid alternation between two languages within a single utterance—lacks standardized, high-volume datasets. Consequently, the primary engineering obstacle is not merely the design of a transformer model, but the construction of a "Data Factory": an industrial-grade, automated pipeline capable of harvesting, refining, and synthesizing the requisite training triplets (Audio, Code-Switched Transcript, and Monolingual Translation) from the chaotic unstructured data of the open web.

This research report provides an exhaustive, low-level engineering roadmap for deploying such a system. It synthesizes state-of-the-art methodologies in data ingestion, signal processing, synthetic generation, and neural architecture design. The analysis moves beyond theoretical abstraction to concrete implementation, detailing a **Dual-Pipeline Strategy** that converges real-world acoustic data from platforms like YouTube with linguistically controlled synthetic data derived from text scraping.

Crucially, this report integrates the latest findings on **"Triangle" network architectures**, which stabilize E2E training by explicitly modeling the interdependence of Automatic Speech Recognition (ASR) and Machine Translation (MT), and explores the utility of **Two-Stage Phoneme-Centric (TSPC)** models for handling the specific phonological challenges of the Vietnamese-English language pair. Furthermore, it evaluates the integration of cutting-edge generative voice tools—specifically **XTTS v2** and **VALL-E X**—to solve the data scarcity problem through high-fidelity cross-lingual synthesis.

By strictly adhering to MLOps best practices—separating storage from state (PostgreSQL) and code (Git/DVC)—this architecture ensures reproducibility, scalability, and long-term maintainability. The resulting system is not just a model, but a sustainable ecosystem for low-resource language processing.

I High-Level Architecture & Data Strategy

The foundational premise of this architectural design is that code-switching data is naturally scarce, noisy, and uncurated. In the specific context of Vietnamese-English CS, the challenge is compounded by the collision of two distinct phonological systems: the tonal, monosyllabic nature of Vietnamese and the stress-timed, polysyllabic nature of English. This phonological interference creates unique acoustic boundaries that standard monolingual models frequently misinterpret. Therefore, a monolithic database approach is insufficient. The system requires a decoupled architecture that separates the "Warehouse" (storage) from the "Ledger" (metadata), facilitating high-throughput ingestion without schema locking.

1.1 The Philosophy of the Data Factory

The transition from "Model-Centric" AI to "Data-Centric" AI necessitates viewing the dataset not as a static artifact downloaded once, but as a programmable product of a continuous manufacturing process. In this "Data Factory," raw inputs (noisy vlogs, blog posts) enter one end, undergo a series of transformative operations (denoising, segmentation, alignment, synthesis), and emerge as refined training tensors. This factory must handle two distinct distributions of data:

1. **The "Real" Distribution (Audio-First):** Captures the messy, spontaneous acoustic reality of code-switching but suffers from imperfect transcription and missing translations.
2. **The "Synthetic" Distribution (Text-First):** Captures the precise linguistic structures and domain-specific vocabulary (e.g., IT jargon) but often lacks prosodic naturalness.

The convergence of these two pipelines creates a robust dataset where the weaknesses of one are offset by the strengths of the other.

1.2 Storage Architecture: The Hybrid Model

A prevalent anti-pattern in early-stage ML engineering is the storage of binary blobs (audio files) directly within relational databases or, conversely, the reliance on file naming conventions to store metadata. Both approaches fail at scale. The proposed architecture strictly separates these concerns to maximize throughput and query flexibility.

1.2.1 Object Store (The Warehouse)

We utilize a **Local Filesystem** approach managed by **DVC (Data Version Control)** as the data warehouse. Unlike complex object stores (MinIO/S3) which incur higher infrastructure overhead, this approach leverages high-speed local disk I/O (bind mounts) for processing while relying on DVC to handle synchronization, versioning, and remote backup to **Google Drive**. This serves as the immutable source of truth for all binary assets.

- **Directory Structure Strategy:** The structure reflects processing stages to ensure idempotency. Processing scripts read from a predictable input path and write to a distinct output path.
  - `data/raw/{video_id}.wav`: The immutable original dump.
  - `data/processed/clean/{video_id}.wav`: The output of the Denoising module.
  - `data/processed/vad/{video_id}/{segment_id}.wav`: The output of Voice Activity Detection.
  - `data/assets/spectrograms/{segment_id}.png`: Pre-computed features for visualization in labeling tools.
- **Rationale:** This "Local-First" strategy eliminates network latency during heavy processing loops (e.g., FFmpeg operations) while DVC provides the "Cloud-Native" benefits of S3 by syncing the final artifacts to Google Drive for team collaboration and backup.

1.2.2 Metadata Store (The Ledger)

A relational database acts as the central ledger, tracking the *state* and *lineage* of every data point. **PostgreSQL** is the mandatory choice here, preferred over SQLite for any dataset exceeding 100,000 clips due to its superior concurrency control (MVCC) and robust JSONB support, which allows for flexible schema evolution.

- **Schema Design Principles:** The schema must support complex querying for dataset balancing (e.g., "Select 50 hours of audio where the speaker is female and the CS ratio is > 20%").
  - `sample_id`: UUID (Primary Key).
  - `file_path`: Text (Relative Local Path).
  - `source_metadata`: JSONB (Contains URL, channel ID, upload date, license).
  - `acoustic_meta`: JSONB (Duration, SNR estimate, speaking rate).
  - `linguistic_meta`: JSONB (CS ratio, vocabulary density, language tags).
  - `transcript_raw`: Text (ASR output).
  - `transcript_corrected`: Text (Human validated).
  - `translation`: Text (Target monolingual output).
  - `processing_state`: Enum (RAW, DENOISED, SEGMENTED, REVIEWED).
  - `split_assignment`: Enum (TRAIN, TEST, VAL).

1.2.3 Data Versioning (DVC)

To maintain reproducibility, we integrate **DVC** with a **Google Drive** remote. While code is versioned in Git, data is too large. DVC bridges this gap by creating small pointer files (*.dvc) that track the checksums of datasets.

- **Workflow:** When an experiment is run, the specific version of the dataset is checked out from Google Drive using `dvc pull`. This ensures that a model trained on a cloud GPU instance (e.g., Vast.ai) sees exactly the same data as the local development environment, solving the "shifting sands" problem in evolving datasets.

1.3 MLOps and Infrastructure Integration

The orchestration of these components is managed via containerization using **Docker Compose**. This ensures that the complex web of dependencies (FFmpeg, PyTorch, CUDA drivers, Python libraries) remains consistent across development and production.

- **Postgres Service (`factory_ledger`):** The database container initialized with a specific schema for tracking dataset lineage.
- **Ingestion Service (`factory_ingestion`):** A lightweight Python container equipped with `yt-dlp`, `FFmpeg`, and `dvc`. It handles the downloading, format standardization (16kHz Mono), and initial database registration of raw assets.
- **Processing Service:** A GPU-enabled container for VAD, Denoising, and Alignment.
- **Labeling Service:** A container hosting the Label Studio instance.
- **Training Service:** A heavy GPU container with Fairseq/HuggingFace libraries.

This modular architecture allows for independent scaling. The ingestion container handles CPU-bound tasks (downloading/encoding) and communicates with the database via a persistent internal network, ensuring robust data integrity checks before any file is committed to storage.

II Low-Level Implementation Steps: Phase 1 - The Crawler & Ingestion Engine

The ingestion engine is the mouth of the Data Factory. Its objective is to ingest raw data from the open web and standardize it immediately, isolating the downstream systems from the variability of external platforms.

2.1 Audio-First Pipeline

YouTube is the largest repository of natural code-switched speech. Vietnamese tech vloggers, gamers, and expatriates living in English-speaking countries frequently produce content that naturally blends both languages. Channels such as **Khoai Lang Thang** (travel), **Coding with Nina** (tech/career), and **Audrey Nguyen** (family/bilingualism) are prime examples of high-density CS sources.

2.1.1 The Tool: yt-dlp

We utilize **yt-dlp**, a highly active fork of `youtube-dl`, for its superior handling of network throttling and metadata extraction.

2.1.2 Intelligent Filtering Strategy

Blind ingestion leads to data swamps. We must implement intelligent filtering to maximize the yield of usable CS data.

- **Duration Gating:** We filter for videos between 2 and 20 minutes. Short videos often lack sufficient context, while videos longer than 20 minutes introduce complexity in alignment and memory management during segmentation.
- **Linguistic Fingerprinting:** We scan video descriptions and titles for bilingual markers (e.g., "EngSub", "Vietnamese/English", "Reaction").
- **Subtitle Nuance:** This is a critical technical detail. YouTube offers both "auto-generated" (ASR) and "auto-translated" subtitles. `yt-dlp` distinguishes these, but the labeling can be misleading.
  - The command `-write-auto-sub` fetches the ASR captions.
  - The command `-write-sub` fetches manual captions.
  - **The Trap:** YouTube often lists translated subtitles as if they were native. For a Vietnamese video, an "English" subtitle track might be a machine translation of the Vietnamese audio, not a transcript of English speech. We must verify the `track_kind` in the metadata to ensure we are downloading the *transcription* of the audio, not a *translation*.

2.1.3 Implementation Command

The ingestion script executes the following `yt-dlp` command to standardize audio at the point of entry:

```
yt-dlp -f 'ba' \
-x --audio-format wav --audio-quality 0 \
--postprocessor-args "-ar 16000 -ac 1" \
--write-auto-sub --sub-lang "vi,en" \
--output "data/raw/%(id)s.%(ext)s" \
{url}
```

- `-f 'ba'`: Downloads "best audio," ignoring video streams to save bandwidth and storage.
- `-ar 16000 -ac 1`: Resamples to 16kHz mono immediately. Most pre-trained speech encoders (Wav2Vec 2.0, Whisper) are trained on 16kHz mono audio. Doing this conversion upstream prevents repeated resampling during training.
- `-write-auto-sub`: Captures the timing data. Even if the text is imperfect, the *timestamps* of speech segments are invaluable for the segmentation phase.

2.2 Text-First Pipeline

To supplement the noisy audio data, we harvest high-quality text from blogs, forums (e.g., Tinhte, Voz), and newsletters (Substack). This text is then converted into synthetic audio.

2.2.1 The Challenge of Detection

Detecting code-switching in short text segments is non-trivial. Standard Language Identification (LID) tools are optimized for long documents and often fail on mixed sentences.

- **Benchmarking LID Tools:**
  - **Langid.py:** Fast, but relies on Naive Bayes and often struggles with short, mixed strings.
  - **FastText:** Extremely fast and efficient, but requires a reasonable character count to form reliable n-gram embeddings. It may misclassify a Vietnamese sentence with one English word as purely Vietnamese.
  - **Lingua-py:** A newer library that outperforms others on short text and mixed-language detection. It is slower (3-20 seconds for large batches vs. milliseconds for FastText) but significantly more accurate for the sentence-level granularity required here.

2.2.2 The "Intersection" Heuristic

Given the scale of web scraping, running a heavy model like Lingua on every sentence is cost-prohibitive. We implement a tiered heuristic filter:

1. **Tier 1 (Dictionary Check):** A fast set intersection check. Does the sentence contain at least one high-frequency Vietnamese particle (*và, là, của, những*) AND one English stop word (*the, and, so, is*)? This operation is O(1) and filters 99% of monolingual text.
2. **Tier 2 (LID Verification):** Passing candidates are then analyzed by Lingua-py or FastText to confirm the language probabilities are split (e.g., $P(vi) > 0.4$ and $P(en) > 0.2$).
3. **Context Windowing:** We must capture the "neighbors" of the CS sentence (S_{i-1}, S_i, S_{i+1}). Code-switching is context-dependent; the translation often relies on the antecedent monolingual sentence. The scraper must store this sliding window.

III Low-Level Implementation Steps: Phase 2 - The Pre-processing Pipeline

Phase 2 transforms the raw, noisy artifacts from Phase 1 into clean, aligned tensors ready for model consumption. This is the most computationally intensive phase, effectively refining the "ore" into "fuel."

3.1 Audio Segmentation: The "Long Video" Problem

Transformers have quadratic memory complexity regarding sequence length ($O(N^2)$). We cannot feed a 10-minute vlog into Wav2Vec 2.0; we must segment it into chunks of 2 to 20 seconds.

3.1.1 Strategy A: Forced Alignment (Transcript-Aware)

When a transcript (manual caption) is available, we use **Montreal Forced Aligner (MFA)**.

- **Mechanism:** MFA uses a pronunciation dictionary and an acoustic model (typically HMM-GMM based) to align phonemes to the audio waveform.
- **Process:**
  1. Convert the transcript into a compatible format (TextGrid).
  2. Run alignment to find precise start/end times for every word.
  3. Merge contiguous words into sentence-level segments.
  4. Cut the audio using ffmpeg based on these timestamps.
- **Advantage:** This method yields the highest quality "Gold" data because the audio and text are strictly synchronized.

3.1.2 Strategy B: Voice Activity Detection (Audio-Only)

For the majority of data (Auto-subs or Raw Audio), we rely on **Voice Activity Detection (VAD)**.

- **Tool Comparison:**
  - **WebRTC VAD:** The traditional standard. It is fast and lightweight but relies on Gaussian Mixture Models (GMMs) and struggles with background noise (music, street sounds), leading to high false positives.
  - **Silero VAD:** A modern, deep-learning-based VAD. It is highly robust to noise, optimized for production (ONNX runtime), and significantly more accurate than WebRTC. It typically has a latency of < 100ms.
  - **Pyannote.audio:** A comprehensive toolkit that includes VAD and **Speaker Diarization**. Diarization is crucial for multi-speaker videos (e.g., interviews). Pyannote can segment audio *and* assign speaker labels (Speaker A vs. Speaker B), preventing the model from learning to associate a voice change with a translation shift.
- **Selection: Silero VAD** is the engineering choice for pure segmentation due to its speed and ease of integration. **Pyannote** is reserved for files tagged as "Interview" or "Podcast" in the metadata.

3.2 Signal Enhancement: The Denoising Module

Real-world code-switching data often comes from vlogs recorded in cafes, on streets, or with background music.

- **Tool Comparison:**
  - **Demucs:** Originally designed for music source separation. It is excellent at isolating vocals from complex musical backgrounds (drums, bass). However, it is computationally heavy and slower than real-time.
  - **DeepFilterNet:** A low-latency, speech-enhancement framework. It uses a two-stage approach (enhancing spectral envelope and then fine structure) to remove non-stationary noise (like wind or traffic) while preserving speech intelligibility. It is highly efficient, capable of running faster than real-time on CPUs.
- **Implementation:** We implement **DeepFilterNet** as the default denoiser for the pipeline due to its balance of quality and throughput. For videos explicitly tagged as "Music" or "Reaction," we route the audio through **Demucs** to strip the background track effectively.

3.3 Synthetic Generation: The TTS Engine

The Text-First pipeline requires converting scraped CS text into audio. This allows us to "inject" specific vocabulary that might be missing from the YouTube dataset.

3.3.1 The "Code-Switching" TTS Challenge

Standard TTS models are monolingual. Feeding English text into a Vietnamese TTS usually results in the English words being read with Vietnamese phonetic rules (e.g., "Facebook" becoming "Phây-buh"), which defeats the purpose of training an ASR model to recognize correct English pronunciation.

3.3.2 Model Selection

1. **MMS-TTS (Meta):** Based on the VITS architecture (Variational Inference with adversarial learning). It supports over 1,000 languages, including Vietnamese. While high quality, it is fundamentally a collection of monolingual

checkpoints. Switching languages mid-sentence requires complex hacking of the inference loop to swap checkpoints or adapters.

2. **XTTS v2 (Coqui):** A diffusion-based model that supports voice cloning and multilingual generation. It allows for "Prompt Conditioning"—feeding a 6-second clip of a real Vietnamese speaker allows the model to generate new text in that voice. Crucially, it supports distinct language codes (e.g., [en], [vi]) within the API, enabling controlled code-switching.
3. **VALL-E X (Microsoft):** A neural codec language model. It treats TTS as a language modeling task, predicting acoustic tokens. It excels at **Zero-Shot Cross-Lingual Synthesis**. It can take a prompt in Vietnamese and synthesize English text *in the same voice*, maintaining the speaker's acoustic environment (e.g., reverb, background ambience). This is a game-changer for CS data, as it preserves speaker identity across the language switch.

3.3.3 The Engineering Choice: VALL-E X & XTTS v2

We employ a mix. **XTTS v2** is used for high-fidelity, clean read speech. **VALL-E X** is used to generate more diverse, "in-the-wild" style synthetic data by prompting it with noisy acoustic tokens from our real dataset.

- **Prompting Strategy:** To force code-switching, we structure the input text with explicit language tags: `[VI]Hôm nay mình sẽ[VI][EN]review[EN][VI]cái sản phẩm này[VI]`. This explicit segmentation guides the model to switch phonemization backends at the correct moment.

IV Low-Level Implementation Steps: Phase 3 - The "Human-in-the-Loop" (Review & Translation)

We cannot rely solely on automation. The "Gold Standard" test set and a training validation subset must be verified. This phase bridges the gap between raw ingestion and model training.

4.1 The Triplet Problem

The objective is to produce triplets: (Audio, CS_Transcript, Monolingual_Translation).

- **Audio:** Derived from ingestion.
- **CS_Transcript:** Derived from ASR (Audio-First) or Scraping (Text-First).
- **Translation:** This is the missing link in the Audio-First pipeline.

4.2 LLM Bootstrapping (Pre-translation)

Before a human sees the data, we use Large Language Models (LLMs) to generate a "Draft Translation" and correct the transcript.

- **Prompt Engineering:** "You are an expert Vietnamese-English translator. The input is a Vietnamese-English code-switched sentence.
  1. Correct any phonetic errors in the transcript (e.g., 'face book' -> 'Facebook').
  2. Translate the sentence into natural, monolingual Vietnamese. Output JSON: {corrected_transcript: '...', translation: '...'}"
- **Model:** GPT-4o or a fine-tuned Llama 3 model. This step reduces human cognitive load by transforming the task from "creation" to "verification."

4.3 The Annotation Interface: Label Studio

Label Studio is the industry standard for this workflow due to its open-source nature and S3 integration.

4.3.1 Configuration

We configure a custom Label Studio interface that displays the audio waveform alongside the two text fields. The configuration uses XML to define the layout and control logic.

XML Configuration Snippet:

```
<View>
<Audio name="audio" value="$audio_url" zoom="true" hotkey="ctrl+enter"/>
<Header value="Transcript (Code-Switched)"/>
<TextArea name="transcript" toName="audio" value="$cs_text" rows="2"
editable="true" maxSubmissions="1"/>
<Header value="Vietnamese Translation"/>
<TextArea name="translation" toName="audio" value="$vi_text" rows="2"
editable="true" maxSubmissions="1"/>
</View>
```

4.3.2 The Review Workflow

1. **Ingest:** A Python script queries the PostgreSQL Ledger for RAW segments, generates pre-signed S3 URLs, and pushes tasks to the Label Studio API.
2. **Annotate:** A human reviewer listens to the audio.
   - *Step 1:* Corrects the ASR errors in the transcript (e.g., fixing "upload" transcribed as "ắp lốt").
   - *Step 2:* Verifies the LLM translation, ensuring the nuance of the English term is captured in Vietnamese.
3. **Export:** A webhook triggers on the "Submit" action, calling an endpoint in our API that updates the PostgreSQL `is_reviewed` flag to TRUE and saves the corrected text.

V Model Training: The "Triangle" Architecture

The core of the "End-to-End Speech Translation" framework is the neural architecture. We adopt the **"Triangle" Architecture**, a Multi-Task Learning (MTL) approach that stabilizes the difficult mapping from Audio to Translation by using the Transcript as an intermediate scaffold.

5.1 Theoretical Foundation

In a direct E2E ST model (Audio $\rightarrow$ Translation), the encoder often struggles to learn robust acoustic features because the semantic gap between the source audio (mixed languages) and the target translation (monolingual) is too large. The model attempts to learn phonetics, semantics, and translation simultaneously. The **Triangle Architecture** solves this by adding an auxiliary path: **Audio** $\rightarrow$ **Transcript**.

1. **Shared Encoder:** Processes Audio inputs to extract acoustic embeddings.
2. **ASR Decoder:** Attends to the Encoder output to generate the **CS Transcript**.
3. **ST Decoder:** Attends to *both* the Encoder output *and* the ASR Decoder's hidden states to generate the **Translation**.

This forces the Shared Encoder to retain high-fidelity phonetic information (driven by the ASR loss), which the ST Decoder can then leverage to disambiguate the translation.

5.2 The Two-Stage Phoneme-Centric (TSPC) Alternative

Recent research specifically on Vietnamese-English CS suggests a refinement to the standard Triangle model: the **Two-Stage Phoneme-Centric (TSPC)** architecture.

- **The Problem:** Vietnamese is tonal; English is not. Direct word-level decoding often confuses English words with Vietnamese tonal homophones.
- **The Solution:** TSPC introduces an intermediate **Phoneme** representation.
  1. **Stage 1 (S2P):** Speech-to-Phone. The model decodes audio into a sequence of IPA phonemes (combining Vietnamese and English phoneme sets).
  2. **Stage 2 (P2T):** Phone-to-Text. The model translates the phoneme sequence into the final text.
- **Relevance:** This approach is particularly robust for "Zero-Shot" scenarios where the model encounters English words not seen in training, as it can rely on phonetic transcription rather than word-level memorization.

5.3 Implementation Strategy with HuggingFace

We leverage the HuggingFace transformers library to implement the Triangle architecture. While no single "TriangleClass" exists out of the box, we can compose it using the `SpeechEncoderDecoderModel` framework.

5.3.1 The Backbone: Wav2Vec 2.0 + mBART

We utilize pre-trained giants to avoid training from scratch (Transfer Learning).

- **Encoder:** `wav2vec2-large-xlsr-53`. This model is pre-trained on 53 languages, including Vietnamese, providing robust multilingual acoustic embeddings.
- **Decoder:** `mbart-large-50`. mBART is a denoising autoencoder pre-trained on many languages. It understands the probability distribution of Vietnamese text, making it an excellent initializer for the translation head.

5.3.2 Implementation Logic

We create a custom `nn.Module` that wraps a shared `Wav2Vec2Model` and two `MBartForConditionalGeneration` decoders (or heads).

1. **Forward Pass 1 (ASR):**
   - Input: `input_values` (Audio).
   - Target: `transcript_tokens`.
   - Loss: CTCLoss (on Encoder intermediate layers for alignment) + CrossEntropyLoss (on ASR Decoder output).
2. **Forward Pass 2 (ST):**
   - Input: `input_values` (Audio).
   - Auxiliary Input: `asr_decoder_hidden_states` (passed via cross-attention).
   - Target: `translation_tokens`.
   - Loss: CrossEntropyLoss (on ST Decoder output).

The total loss is a weighted sum: $L_{total} = \lambda_{ST} L_{ST} + \lambda_{ASR} L_{ASR}$. The λ_{ASR} term acts as a regularizer, ensuring the encoder doesn't "forget" the acoustic reality while trying to satisfy the translation objective.

5.4 Data Manifests

To feed this training loop, we generate JSONL manifests from our Metadata Store. This format acts as the contract between the engineering pipeline and the research code.

Format:

```
{
"id": "10293",
"audio_path": "/data/processed/segments/10293.wav",
"duration": 4.2,
"transcript": "Okay, hôm nay mình sẽ review cái feature mới này.",
"translation": "Được rồi, hôm nay mình sẽ đánh giá tính năng mới này.",
"split": "train"
}
```

VI Engineering Stack & MLOps

To orchestrate this complexity, a robust infrastructure is required. The system is designed to be cloud-agnostic but container-native.

6.1 Containerization & Orchestration

All services are containerized using **Docker**.

- **Service A (Ingest):** Contains `yt-dlp`, `ffmpeg`, and Python scraping scripts. It requires high network I/O and moderate CPU.
- **Service B (Processing):** Contains PyTorch (for VAD/Denoising) and requires NVIDIA Container Runtime for GPU acceleration.
- **Service C (Label Studio):** A web application container backed by Postgres.
- **Service D (Training):** The heavy lifter containing Fairseq/HuggingFace.

Docker Compose is sufficient for single-node development, but **Kubernetes** is the target for production scaling, allowing independent scaling of the Ingestion (CPU) and Training (GPU) layers.

6.2 Data Versioning (DVC)

We use **DVC** to manage the dataset lifecycle.

- **Mechanism:** DVC tracks the JSONL manifest files and the checksums of the audio files in MinIO.
- **Benefit:** This allows us to "time travel." If Model V5 performs worse than Model V4, we can checkout the exact DVC commit used for V4 to investigate if data corruption or a bad ingestion batch caused the regression. It effectively brings "Git-like" semantics to terabytes of audio data.

6.3 Hardware Strategy

- **Ingestion/Processing:** This is CPU-bound. High core counts (e.g., AMD EPYC) are beneficial for parallelizing `ffmpeg` re-encoding and `yt-dlp` downloads.
- **Training:** This is VRAM-bound. The Triangle architecture requires loading a large encoder and two decoders into memory. A minimum of **40GB VRAM (NVIDIA A100)** is recommended. If using smaller cards (A10g, 24GB), techniques like **Gradient Checkpointing** and **DeepSpeed ZeRO-2** are mandatory to fit the model.

Conclusion

Building an End-to-End Speech Translation system for Vietnamese-English code-switching is a discipline of systems engineering, not just model training. The "0 to 1" journey requires a fundamental shift from treating data as a given to treating data as a manufactured product. By architecting a "Data Factory" that decouples storage from metadata, employs a Dual-Pipeline strategy to balance acoustic realism with linguistic control, and integrates a rigorous Human-in-the-Loop verification process, we create a sustainable ecosystem for handling the chaos of code-switched speech.

The "Triangle" and "TSPC" architectures then serve as the sophisticated machinery at the end of this assembly line, capable of synthesizing these refined raw materials into a highly performant model. This engineering plan provides the blueprint for replicating state-of-the-art results, transforming a complex linguistic challenge into a manageable, scalable, and reproducible workflow.

Table 1: Summary of Recommended Tooling Stack

Component	Tool Choice	Alternative	Rationale for Choice
Object Storage	**MinIO**	AWS S3, Ceph	S3-compatible, high-performance, self-hostable, erasure coding.
Metadata DB	**PostgreSQL**	SQLite, MySQL	Robust JSONB support for flexible metadata, MVCC for concurrency.
Ingestion	**yt-dlp**	youtube-dl	Active maintenance, better handling of auto-subs and throttling.
LID (Scraping)	**Lingua-py**	FastText, Langid	Superior accuracy on short/mixed sentences compared to FastText.
Segmentation	**Silero VAD**	WebRTC, Pyannote	Best balance of accuracy, noise robustness, and inference speed.
Denoising	**DeepFilterNet**	Demucs, RNNoise	Optimized for speech enhancement; faster than real-time on CPU.
Synthetic TTS	**XTTS v2**	MMS, VALL-E X	Native support for prompting code-switching and voice cloning.
Annotation	**Label Studio**	CVAT, Doccano	Excellent audio UI, direct S3 integration, XML config flexibility.
Model Arch	**Triangle (MTL)**	Direct E2E	Stabilizes training by using transcripts as an intermediate scaffold.
Versioning	**DVC**	Git LFS	Designed specifically for large datasets; separates code from data.