End-to-End Speech Translation for Low-Resource Code-Switching: System Design and Model Evaluation

La Quang Chien Tran Thi Vy Anh Vo Huyen Khanh May
National Economics University
Hanoi, Vietnam

December 13, 2025

Abstract

Code-switching (CS) presents a profound challenge for modern speech systems, particularly for the Vietnamese-English pair where tonal complexities invalidate traditional linguistic frameworks. To address the critical scarcity of high-quality data in this domain, we propose a dual-pronged solution. First, we engineered the "Data Factory," a rigorous full-stack pipeline designed to standardize the ingestion of YouTube content and facilitate high-precision human verification. Second, we benchmarked a End-to-End (E2E) architecture against a Whisper baseline. Experimental results reveal a critical divergence: while the Data Factory successfully processed over 60 hours of verified audio, the modeling results highlight the fragility of low-resource adaptation. The E2E model stalled at a Word Error Rate (WER) of 97.5%, and the Whisper baseline suffered from catastrophic hallucination loops (399% WER). This report, therefore, serves as a blueprint for robust CS data engineering while providing a candid failure analysis of standard architectures in noisy, low-resource settings.

Contents

Contributions

- **La Quang Chien - 11221106:** System Design, Model Design, Report Auditing.
- **Vo Huyen Khanh May - 11224145:** Data Crawling, Annotation, Model Training.
- **Tran Thi Vy Anh - 11220646:** Data Crawling, Annotation, Report Writing.

1 Introduction

1.1 Motivation

In modern Vietnam, the linguistic landscape has shifted dramatically. Vietnamese-English Code-Switching (CS) has become the norm in academic, professional, and media environments. Unlike monolingual speech, CS involves rapid, unpredictable shifts between languages, often breaking the grammatical continuity expected by traditional models. Despite its ubiquity, current speech technologies struggle to process this mixed-modality data, failing to distinguish between the tonal precision of Vietnamese and the stress-timed nature of English.

1.2 Problem Definition

Progress in Automatic Speech Recognition (ASR) and Speech Translation (ST) for this pair is impeded by two major barriers. The first is the **Data Gap**: while monolingual datasets abound, there is a critical shortage of high-quality, timestamped, real-world Vietnamese-English CS data. Existing corpora are often synthetic or too small for deep learning. The second is **Architectural Fragility**: traditional cascaded systems (ASR $\rightarrow$ MT) amplify errors, where a single phonetic hallucination can corrupt the entire translation downstream. Conversely, End-to-End models promise to solve this but historically require massive datasets to converge.

1.3 Objective

This research aims to bridge these gaps through a holistic approach.

1. **Engineering Objective (The Data Factory):** To construct a scalable, fault-tolerant software pipeline capable of synthesizing a dataset from raw audio. This involves orchestrating ingestion, AI-assisted pre-labeling, and distributed human verification into a seamless workflow.
2. **Scientific Objective (The Experiment):** To benchmark the "Triangle" E2E architecture against the Whisper baseline, evaluating whether a custom encoder-decoder arrangement can outperform a massive pre-trained model in the low-resource CS domain.

2 Methodology I: System Design

To address the scarcity of training data, we moved beyond static corpus collection and engineered a dynamic "Data Factory." Unlike traditional datasets aggregated from disparate sources, our approach necessitated a rigorous, full-stack intervention to standardize the ingestion, transcription, and verification of raw audio at scale.

2.1 Architecture Overview and Data Integrity

We employed a three-tier architecture centered on data integrity. Given the complexity of managing relationships between channels, videos, chunks, and segments, we selected **PostgreSQL** as our persistence layer. This choice allowed us to enforce strict foreign key constraints and ACID compliance, preventing the metadata corruption often observed in flat-file storage systems. The backend logic was served via **FastAPI**, which acted as the single source of truth for concurrent requests. By utilizing **SQLModel**, we established a unified type system that validated data both at the API boundary and within the database schema, ensuring that malformed inputs were rejected before they could taint the dataset.

2.2 Ingestion and Automated Processing

The pipeline begins with the ingestion module, which utilizes **yt-dlp** to normalize diverse YouTube source material into a uniform 16kHz mono AAC format. This standardization is critical to prevent acoustic heterogeneity from introducing noise into the training signal.

Following ingestion, we automated the initial transcription using a multi-model cascade powered by

Gemini 2.5. Recognizing the volatility of API rate limits during bulk processing, we implemented a state-machine-based "Key Manager." This module treats API quotas as a pooled resource; upon encountering a `RESOURCE_EXHAUSTED` error, the system automatically cycles through available keys and escalates from the Flash model to the Pro tier. This resilience ensures that ingestion jobs can run unattended overnight, processing hundreds of videos without operator intervention.

To mitigate the hallucination risks inherent in long-context audio processing, we partitioned every video into fixed 5-minute chunks before transmission. These segments were processed using a rigorously engineered prompt strategy designed to enforce linguistic fidelity. We conditioned the model with the persona of a "Senior Linguistic Data Specialist," strictly defining its operating principles: millisecond-precision timestamping, verbatim integrity without censorship, and "tonality preservation." This latter constraint was particularly critical; the model was instructed to translate English terms into natural Vietnamese while strictly preserving the speaker's original register and sentence-final particles (e.g., *á, nè, nhỉ*). By enforcing a structured JSON output schema, we ensured that the resulting dataset captured both the semantic content and the emotional nuance of the code-switched speech.

2.3 The Annotation Workbench

The core of our dataset creation involved human verification, which introduces challenges in concurrency and user experience. We developed a custom **React**-based frontend designed to maximize annotator throughput.

2.3.1 Dashboard and Hierarchy

The system organizes data into a strict hierarchy: Channels contain Videos, and Videos are split into 5-minute Chunks. The **Dashboard**, Figure 1 provides a high-level view of system health, tracking metrics such as total verified hours (currently 63.48 hours) and pending reviews. Navigating to a specific channel reveals a granular status view, allowing administrators to monitor progress at the video level — Figure 2, 3.

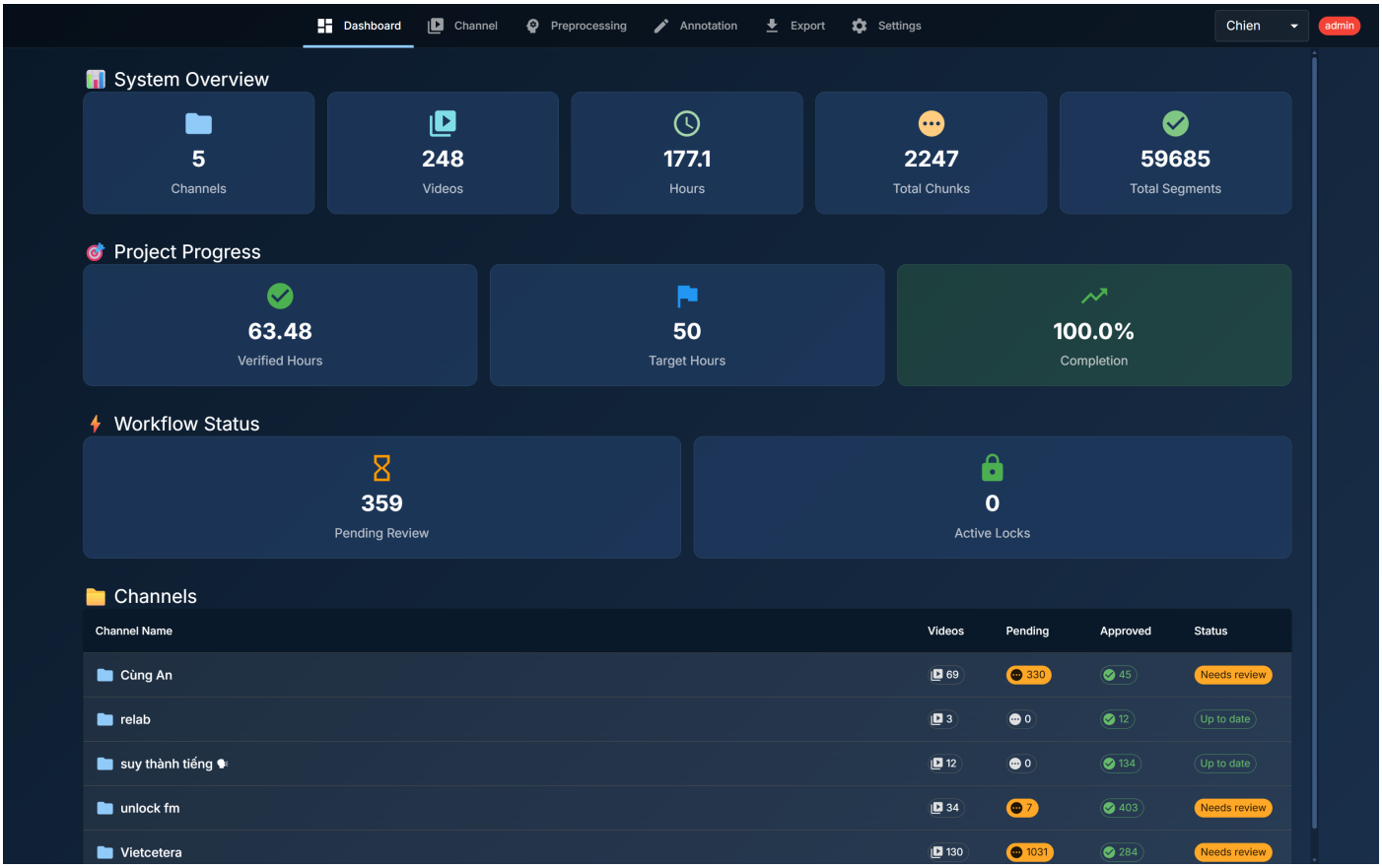

Figure 1: Dashboard Interface

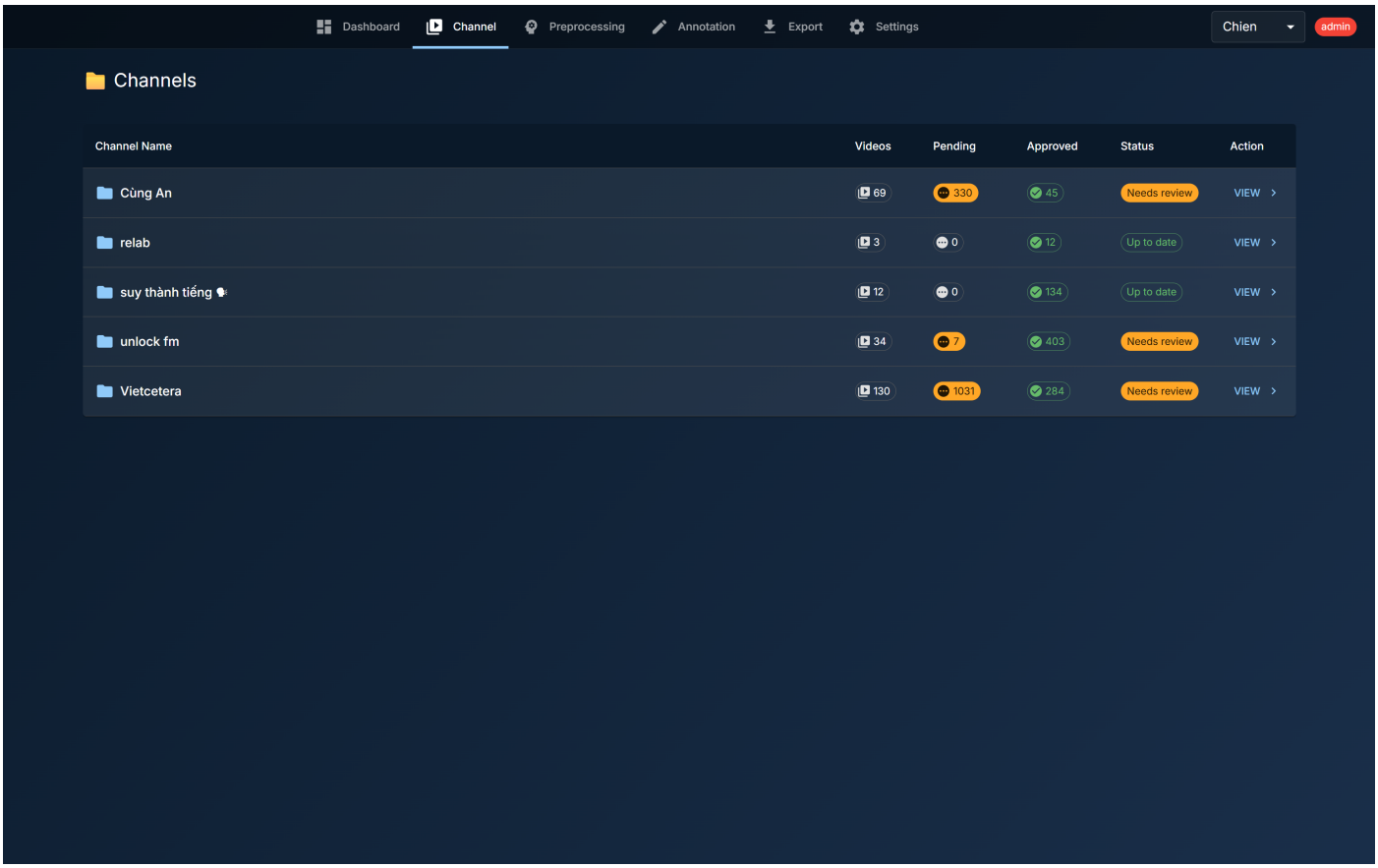

Figure 2: Channel Selection Interface

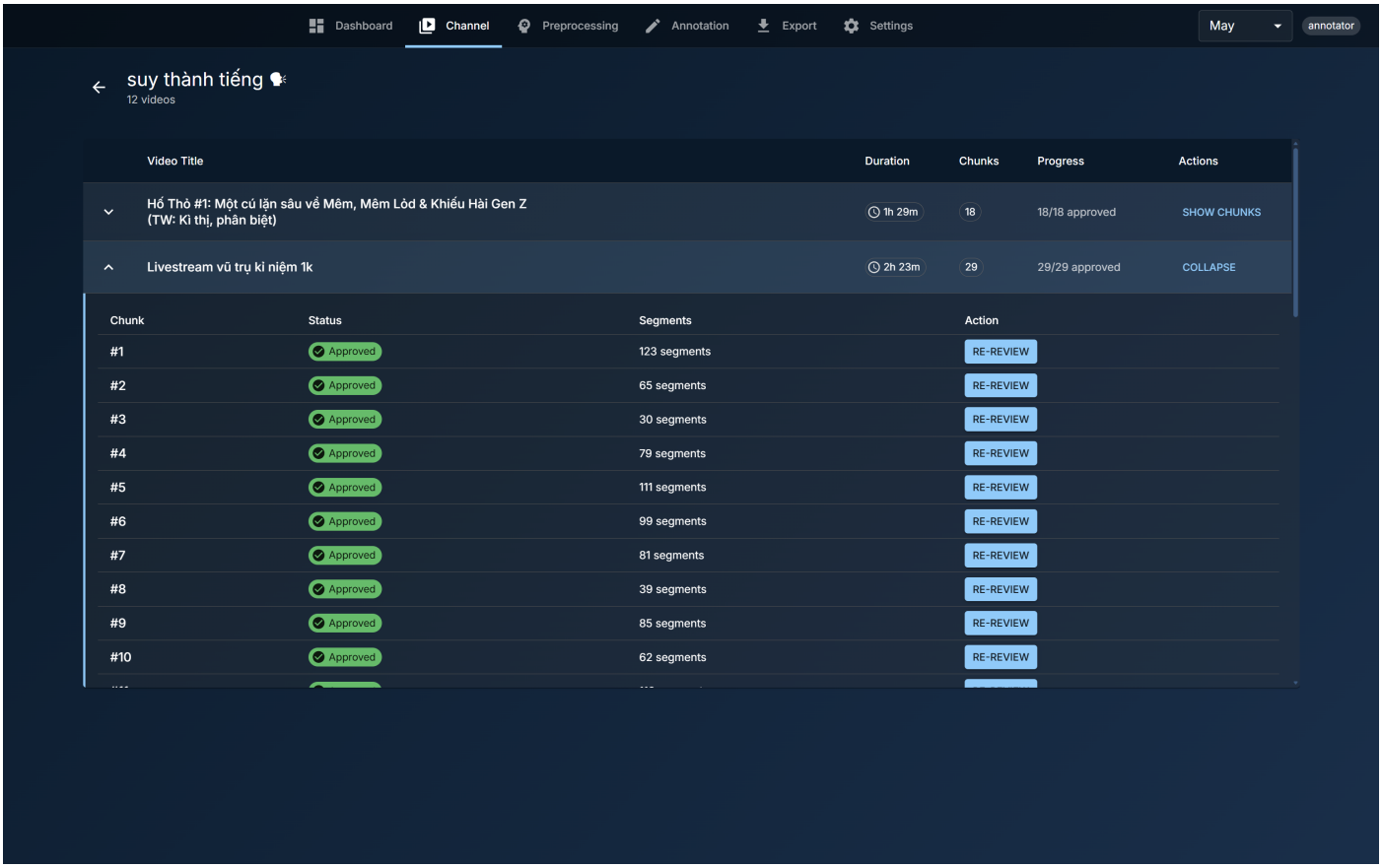

Figure 3: Video Selection Interface

2.3.2 The Workbench Interface

The verification interface, or "Workbench," was designed to solve the "distributed edit" problem. It features a three-zone layout — Figure 4:

1. **Waveform Visualization:** Leveraging `wavesurfer.js`, we render the audio signal directly in the browser. This allows annotators to visually identify silence and speech boundaries, adjusting timestamps with millisecond precision.
2. **Segment Table:** A dense, keyboard-navigable grid allows users to rapidly edit transcripts and translations.
3. **Concurrency Control (Ghost Locking):** To prevent edit collisions, we implemented a "Ghost Lock" pattern. When a user opens a chunk, they acquire a temporary, renewable lock (displayed as a green chip in the UI). This ensures that while the system remains stateless and scalable, no two annotators can unknowingly overwrite each other's work.

Figure 4: Workbench Interface with Waveform Visualization, Segment Table and Concurrency Control

3 Methodology II: Model Architectures

This research contrasts two fundamentally different modeling philosophies for handling Vietnamese-English code-switched speech. We first evaluate an ASR-centric approach using OpenAI's Whisper, treating the task as pure transcription. We then contrast this with a bespoke End-to-End (E2E) architecture, which we engineered to perform direct speech-to-text translation within a single differentiable model. Our goal is to quantify the trade-off between the pre-trained robustness of an industry-standard ASR giant and the architectural flexibility of a purpose-built composite system.

3.1 Baseline: Whisper Architecture (ASR Focus)

For our baseline, we selected OpenAI's Whisper, a sequence-to-sequence Transformer trained on 680,000 hours of multilingual audio. We prioritized this model for its demonstrated resilience across diverse acoustic conditions and its pre-existing competence in Vietnamese.

3.1.1 Model Selection and Input Processing

Our experimental strategy utilized a tiered deployment: we employed `whisper-tiny` (39M parameters) for rapid hyperparameter tuning on constrained hardware (12GB VRAM), while reserving `whisper-small` (244M parameters) for production benchmarks. The model ingests audio via a log-mel spectrogram representation standardizing input to 16 kHz with 80 mel bands, a 400-sample FFT window, and a hop length of 160 samples. Text generation is handled by a byte-level BPE tokenizer with a vocabulary of 50,364, which includes the specialized task tokens (`<|transcribe|>`, `<|translate|>`).

However, the translation task was ultimately excluded from the final evaluation. This is because Whisper's architecture restricts the `<|translate|>` task exclusively to **Any-to-English** translation. It does not support direct translation into Vietnamese, rendering it incompatible with our Code-Switching $\rightarrow$ Vietnamese objective.

Table 1: Whisper Model Variants Used

Variant	Parameters	HuggingFace ID	Role
Tiny	39M	`openai/whisper-tiny`	Development
Small	244M	`openai/whisper-small`	Production

3.1.2 Multitask Training Configuration

To adapt Whisper's massive priors to our specific dual-needs, we initially engineered a multitask training regimen. We configured the decoder with `forced_decoder_ids` to explicitly prime it for Vietnamese transcription. Furthermore, we implemented a batch-duplication strategy in the data collator: the first half of the batch targeted the original code-switched transcript, while the second half targeted the Vietnamese translation.

This strategy proved catastrophic. Because Whisper's `<|translate|>` token forces the decoder to output English, but our ground-truth translation labels were in Vietnamese, the model was subjected to conflicting objectives within the same update step. This architectural misalignment—trying to force an English-only translation head to generate Vietnamese text—destabilized the gradients and likely contributed to the hallucination loops observed in the baseline results.

3.2 Experimental: E2E Shared Architecture (ASR + ST)

Moving beyond the cascaded paradigm, we constructed a composite architecture designed for true end-to-end speech translation. By feeding raw audio directly into an encoder and decoding into the target language, we aim to bypass the error propagation inherent in intermediate text representations.

3.2.1 Composite Components

Our custom model stitches together three distinct modules into a single pipeline — Figure 5:

First, we utilize a **Wav2Vec2 Encoder** (`facebook/wav2vec2-base`) to process the raw 16 kHz waveform. Unlike Whisper's spectrogram approach, Wav2Vec2 learns latent speech representations via contrastive learning on raw audio, outputting a sequence of 768-dimensional hidden states.

Second, to bridge the architectural gap between the encoder and decoder, we inserted a linear **Adapter Layer**. Since the encoder outputs 768-dimensional vectors but our decoder expects 1024-dimensional inputs, this simple linear projection ($W \in \mathbb{R}^{768\times1024}$) is the only component initialized from scratch.

Finally, the processed embeddings are fed into an **mBART-50 Decoder** (`facebook/mbart-large-50`). This multilingual denoising autoencoder leverages a vocabulary of 250,054 tokens. We explicitly set the decoder start token to `vi_VN` to prime the generation of Vietnamese text.

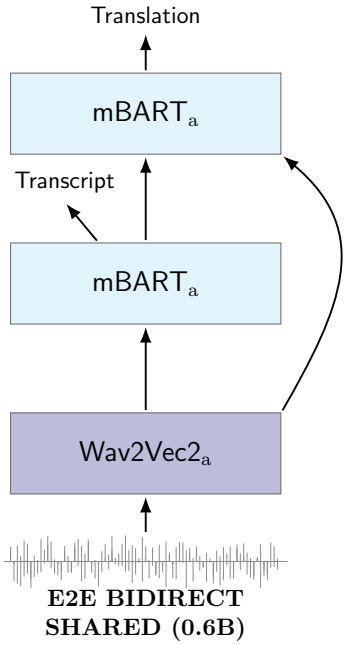

Figure 5: E2E Shared Architecture

3.2.2 Task Tokenization and Optimization

To support single-model multitasking, we extended the mBART vocabulary with two custom special tokens: `<2transcribe>` for ASR tasks and `<2translate>` for translation tasks. These tokens serve as the initial prompt to the decoder, signaling which output modality is required for the duplicated input samples.

Training this 700M+ parameter composite on consumer hardware required aggressive optimization. We employed a **Feature Encoder Freezing** strategy, setting `requires_grad = False` for the seven convolutional layers of the Wav2Vec2 frontend. By locking these low-level acoustic features—learned during pre-training on 960 hours of LibriSpeech—we reduced the trainable parameter count to $\sim$620M. This not only fits the model within the VRAM constraints of our development environment but also stabilizes training by preventing the catastrophic forgetting of robust acoustic priors.

4 Experimental Setup

To validate our architectural hypotheses, we established a reproducible training infrastructure capable of scaling from local development to high-performance cloud environments. This section details the data preparation rigor, hardware configurations, and the specific metrics used to benchmark—and ultimately diagnose the failure of—our models.

4.1 Training Pipeline and Data Preparation

Our training lifecycle operates as a six-stage sequence, orchestrated by shell scripts that normalize the environment across Windows and Linux platforms. The process begins with the rigorous sanitization of the human-verified manifest. Using a custom pre-processor, we stripped markdown artifacts (e.g., `[laughter]`, `[music]`), removed emphasis markers, and normalized whitespace to eliminate non-breaking spaces that frequently corrupt tokenizers. We also applied a heuristic filter to discard segments shorter than 0.5 seconds or longer than 30 seconds, ensuring that the training data remained within the effective attention window of our Transformer models.

Following sanitization, we generated data splits. Crucially, we enforced data isolation at the **video level** rather than the segment level. Since segments from the same source video share high acoustic correlation (speaker identity, background noise, recording quality), random shuffling would have resulted in data leakage. By

grouping segments by `video_id` and shuffling with a fixed seed, we ensured that the test set remained strictly speaker-disjoint from the training set (80% Train, 10% Validation, 10% Test).

4.1.1 Hardware Environments

We parameterized our training configurations to accommodate two distinct hardware profiles: a local environment for rapid debugging of the data collator logic, and a cloud environment for full convergence runs. The discrepancies in batch size and precision, detailed in Table 4.1.1, necessitated a flexible gradient accumulation strategy to maintain consistent effective batch sizes across platforms.

Parameter	**Development (RTX 3060)**	**Production (H100 SXM)**
VRAM	12 GB	80 GB
Batch Size	1	32 (Whisper) / **1** (E2E)
Gradient Accumulation Steps	8	2 (Whisper) / **32** (E2E)
Effective Batch Size	16	64
Mixed Precision	FP16	BF16
Whisper Variant	`whisper-tiny` (39M)	`whisper-small` (244M)
E2E Encoder	`wav2vec2-base` (94M)	`wav2vec2-base` **(94M)**
Encoder Freezing	Full encoder frozen	Feature encoder frozen only
Max Training Steps	100	Full convergence

Table 2: Hardware Environment and Hyperparameter Comparison

4.2 Evaluation Metrics

We assessed model performance using standard metrics for ASR and machine translation, prioritizing character-level accuracy given the ambiguity of Vietnamese word boundaries.

4.2.1 ASR Performance

To measure transcription accuracy, we utilized **Word Error Rate (WER)** and **Character Error Rate (CER)**. WER calculates the edit distance between the prediction and reference normalized by sequence length:

$$\text{WER} = \frac{S + D + I}{N} \times 100\%$$

where S, D, I represent substitutions, deletions, and insertions. However, given the code-switching nature of our dataset, WER often penalizes valid tokenization differences in compound words. Therefore, we placed higher diagnostic weight on CER, which applies the same edit distance logic at the character level, offering a more robust measure of phonetic fidelity for Vietnamese.

4.2.2 Translation Quality

For the E2E model's translation output, we employed **BLEU** (Bilingual Evaluation Understudy) via the `sacrebleu` library to measure n-gram overlap. Additionally, we computed the **CHRF** (Character n-gram F-score), which is particularly well-suited for morphologically rich languages where slight inflections can ruin BLEU scores despite semantic correctness.

$$\text{CHRF} = (1 + \beta^2) \cdot \frac{\text{CHRP} \cdot \text{CHRR}}{\beta^2 \cdot \text{CHRP} + \text{CHRR}}$$

4.3 Evaluation Results: Well, we did our best but the rest did not come

Contrary to our design goals, both the robust baseline (Whisper) and the experimental architecture (E2E) failed to converge to a usable state on the limited 60-hour training split. We report these negative results to highlight the fragility of current low-resource adaptation methods.

4.3.1 Quantitative Collapse in ASR Task

Table 4.3.1 and Figure 6 presents the final metrics against our initial targets. The divergence is extreme. While we aimed for a WER below 20%, the Whisper baseline exploded to nearly 400%, and the E2E model stalled at near-100% error rates.

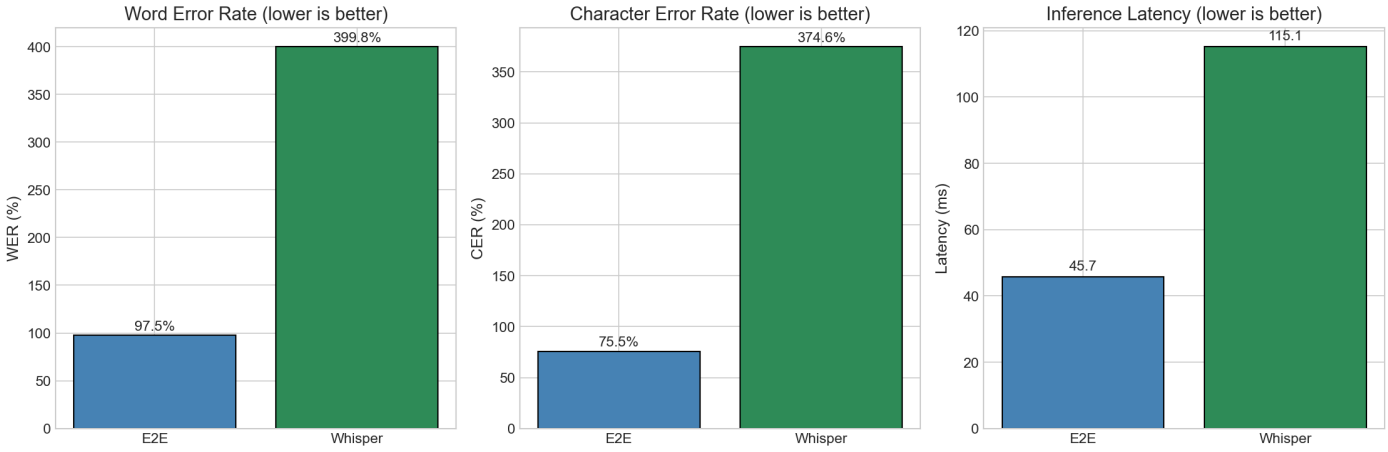

Figure 6: ASR Metrics Comparison. The vast gap between actual performance and targets indicates a fundamental failure of the training strategy.

Model	Metric	Target	Actual	Delta
Whisper	WER (ASR)	$< 20\%$	**399.78%**	$+379\%$
Whisper	CER (ASR)	$< 15\%$	**374.59%**	$+359\%$
E2E	WER (ASR)	$< 25\%$	**97.51%**	$+72\%$
E2E	BLEU (ST)	> 20	**1.34**	-18.6
E2E	CHRF (ST)	> 40	**17.00**	-23.0

Table 3: Target vs. Actual Performance (Production Run)

4.3.2 Diagnosing the Failure Modes

To understand why the WER scores diverged so aggressively (399% vs 97%), we analyzed the distribution of predicted lengths against reference lengths — Figure 7.

Whisper: The scatter plot reveals that Whisper's predictions (green dots) cluster significantly above the diagonal identity line, with many predictions exceeding 800 characters for short audio segments. This confirms a diagnosis of **hallucinatory looping**. When the model encountered ambiguous padding or code-switched transitions it could not resolve, it defaulted to repeating a single token sequence (e.g., "the the the...") until hitting the hard generation limit. This drove the median WER to a catastrophic 452.1%, as insertion errors dominated the edit distance calculation. This suggests that our 5-second overlap strategy, while beneficial for human annotators, introduced ambiguous padding that confused the model's attention mechanism. Furthermore, the `forced_decoder_ids` for Vietnamese likely fought against the strong English acoustic priors present in the code-switched segments, causing the decoder to fracture.

E2E: Conversely, the E2E model's predictions (blue dots) cluster tightly near the x-axis, often below the diagonal. This indicates that the model frequently outputted extremely short sequences or silence tokens. A WER of 97.5% with a median of 100% confirms that the model effectively suffered from **computational aphasia** — it simply failed to align the acoustic features with the decoder's text space, defaulting to "blank" guesses. The E2E model's failure to converge reveals a fundamental architectural flaw in our model selection. We utilized the `wav2vec2-base` encoder to minimize VRAM usage. However, this specific variant is pre-trained **exclusively on English audio** (LibriSpeech) and by freezing of the feature encoder — while necessary to fit the model into VRAM — this froze the acoustic frontend to the domain of LibriSpeech.

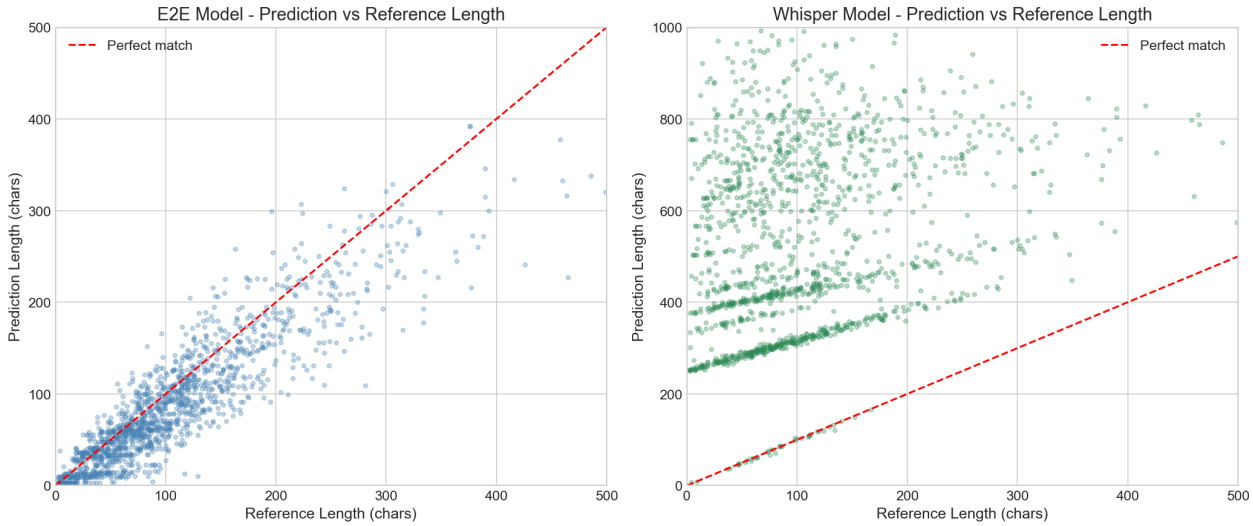

Figure 7: Hyper-Graphia vs. Aphasia

The model lacked the plasticity to adapt to the noisy, overlapping, and tonal nature of our YouTube-derived dataset. As shown in Figure 8, the training loss flatlined, indicating that the adapter layers were insufficient to bridge the domain gap between the frozen encoder and the mBART decoder. In contrast, Whisper benefited from massive multilingual pre-training, explaining why it could generate text (albeit looping) while the E2E model suffered from total semantic collapse.

4.3.3 Training Dynamics

The root cause of these failures is visible in the training loss curves — Figure 8

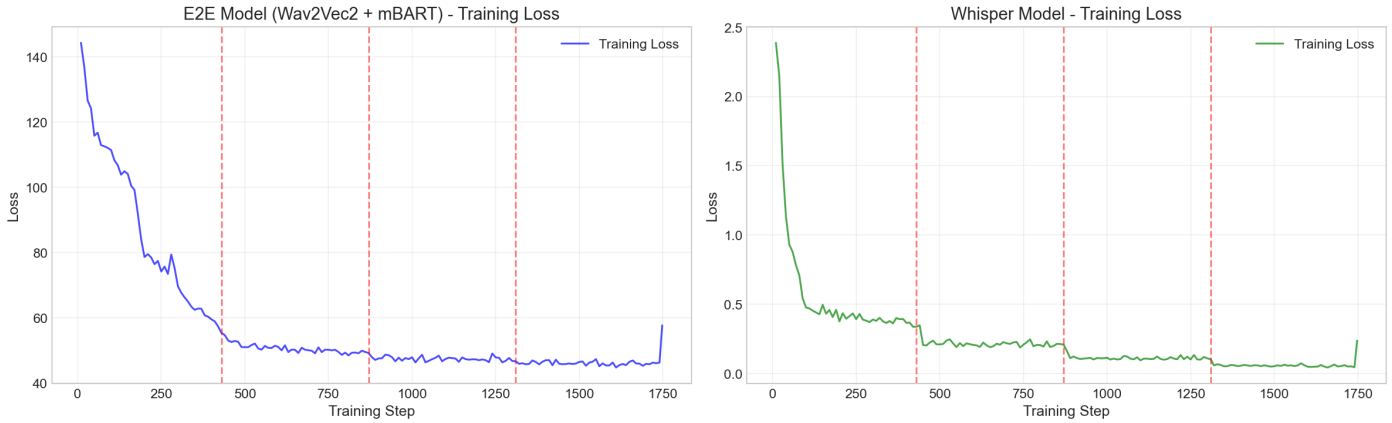

Figure 8: Training Loss Curves. The curves indicate no meaningful reduction in loss, consistent with the high WER. The effective learning rate may have been too low for the E2E adapter layers to bridge the modality gap.

Whisper's loss curve drops precipitously in the early epochs. This is deceptive; combined with the length analysis, it suggests the model quickly "learned" to minimize loss by exploiting a trivial pattern (the loop) rather than learning the linguistic distribution. The E2E model, hindered by a frozen feature encoder, shows a shallow learning curve that plateaus early. The adapter layers alone were insufficient to bridge the domain gap between the clean, English-centric Wav2Vec2 encoder and the noisy, code-switched reality of our dataset.

4.3.4 Latency

One silver lining appeared in the operational benchmarks. The E2E architecture achieved an average inference latency of 45.7 ms, roughly 2.5x faster than Whisper's 115.1 ms, as shown in Figure 9. However, this speed advantage is likely an artifact of failure: since the E2E model generated very few tokens (under-prediction),

it exited the autoregressive generation loop much earlier than Whisper, which generated tokens until the maximum length limit (over-prediction). Thus, while E2E is technically faster, it is functionally useless in its current state.

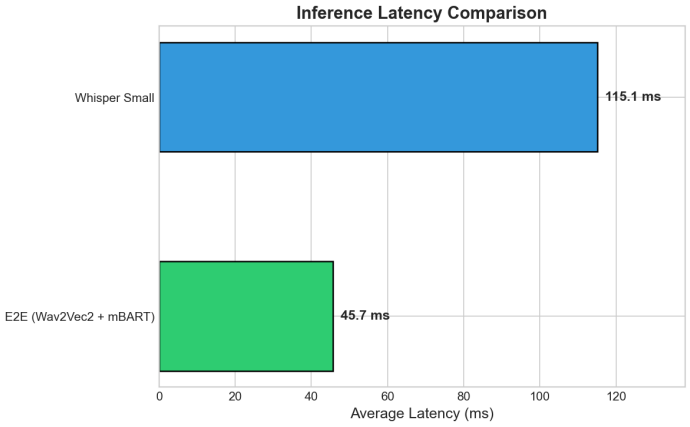

Figure 9: Latency Comparison between two models

4.4 Machine Translation Result

While the E2E architecture was theoretically capable of simultaneous transcription and translation, the translation head failed to capture any significant semantic signal. As shown in Figure 10, the model achieved a BLEU score of 1.34 and a chrF score of 17.01.

These near-zero metrics confirm that the decoder did not learn to map English-heavy code-switched inputs to Vietnamese outputs. Instead, the model likely defaulted to a mode of "safe" generation—outputting high-probability function words or generic fillers (e.g., "tôi nghĩ là") that bear no relation to the source audio. The discrepancy between the chrF (17.01) and BLEU (1.34) suggests that while the model outputted valid Vietnamese character sequences (hence the non-zero character n-gram score), it completely failed at the word and phrase level required for intelligible translation.

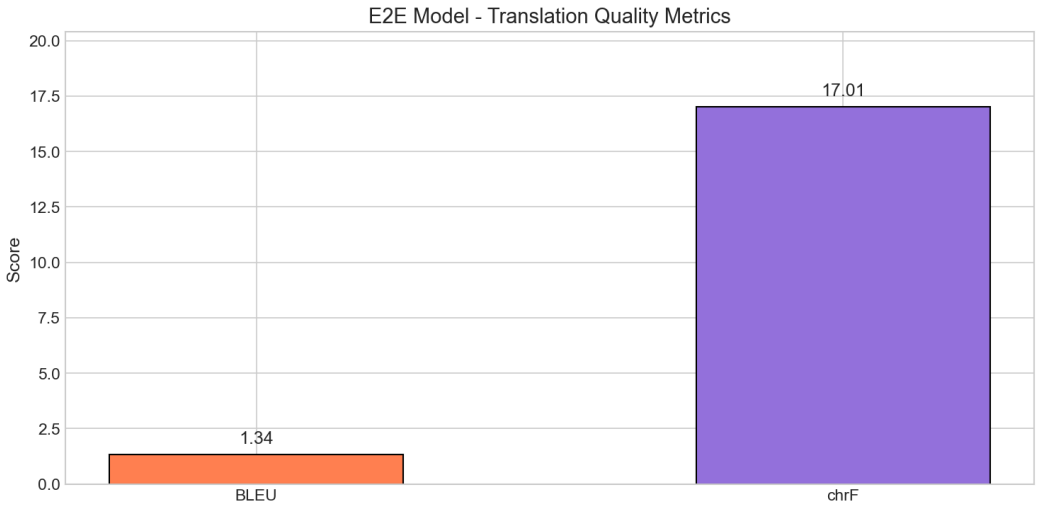

Figure 10: Machine Translation Result

5 Implementation Challenges & Solutions

Building a production-grade data pipeline and training infrastructure for code-switched speech translation surfaced several non-obvious engineering challenges. This section documents the algorithmic and architectural solutions deployed to handle boundary discontinuities, resource volatility, and training instability.

5.1 The Overlap Problem

Dividing long-form audio into discrete segments is an operational necessity to satisfy Gemini's context window limits and manage annotator cognitive load. We implemented a chunking strategy where videos are split into 5-minute (300-second) blocks, each extended by a 5-second overlapping buffer (total duration 305 seconds). While this buffer ensures that no speech is cut off mid-utterance at the boundary, it introduces a critical data hygiene issue: the audio in the [300s, 305s] overlap zone exists physically in two adjacent chunks. Without intervention, segments transcribed in this zone would be exported twice—once at the end of Chunk N and again at the start of Chunk $N + 1$—poisoning the dataset with duplicate training samples.

5.1.1 The 300-Second Guillotine

We initially considered complex deduplication algorithms involving cross-chunk timestamp merging and acoustic alignment. However, we found that the complexity cost outweighed the data benefits. Instead, we implemented a deterministic heuristic we term the **300-Second Guillotine**. The logic is brutal but effective: during the export phase, any segment with a relative start time greater than or equal to 300 seconds is strictly dropped from its parent chunk.

$$\text{Export}(S) \iff S_{\text{start\_relative}} < 300.0$$

In practice, a segment starting at the 301-second mark in *Chunk N* is discarded. However, due to the overlap strategy, this exact same acoustic event appears at the 1-second mark of *Chunk N+1*, where it satisfies the condition and is successfully captured.

This approach guarantees **zero duplication** and strict determinism using a simple SQL `WHERE` clause, avoiding the non-deterministic edge cases inherent in fuzzy merging. We accept a minor trade-off: segments starting immediately before the cutoff (e.g., at 298s) are retained but physically clipped at the 305s file boundary. Given the scale of our dataset, this negligible loss of trailing context was deemed an acceptable price for architectural simplicity.

5.2 Resource Exhaustion: Gemini API Rate Limits

The Gemini API enforces strict per-minute and per-day request quotas. During the bulk ingestion of over 150 hours of audio, `RESOURCE_EXHAUSTED` (HTTP 429) errors are not exceptional; they are the expected steady state.

5.2.1 The ModelKeyManager State Machine

To prevent these errors from halting the pipeline, we architected the `ModelKeyManager` — Figure 11 — as a resilient state machine that treats API quotas as a pooled, tiered resource. Rather than aborting on a 429 error, the system places the specific (model, key) tuple on a temporary cooldown. It then transparently attempts to satisfy the request by cycling through remaining keys in the cost-effective `gemini-2.5-flash` tier. If the entire Flash pool is exhausted, the system automatically escalates the request to the higher-capacity `gemini-2.5-pro` tier. A global sleep state is triggered only in the catastrophic case where all tiers are depleted, ensuring that ingestion jobs can self-heal and run unattended overnight.

5.3 Training Stability: Gradient Checkpointing Conflicts

A significant instability arose from the interaction between our multitask objective and PyTorch's gradient checkpointing mechanism. Our E2E data collator duplicates every input sample to simultaneously compute ASR and translation losses, effectively forcing the training loop to perform two backward passes through a single computation graph.

This architecture conflicts with gradient checkpointing, which optimizes memory by aggressively freeing intermediate activations during the forward pass. When the optimizer attempts the second backward pass for the duplicated batch, it encounters a `RuntimeError` because the required graph nodes have already been

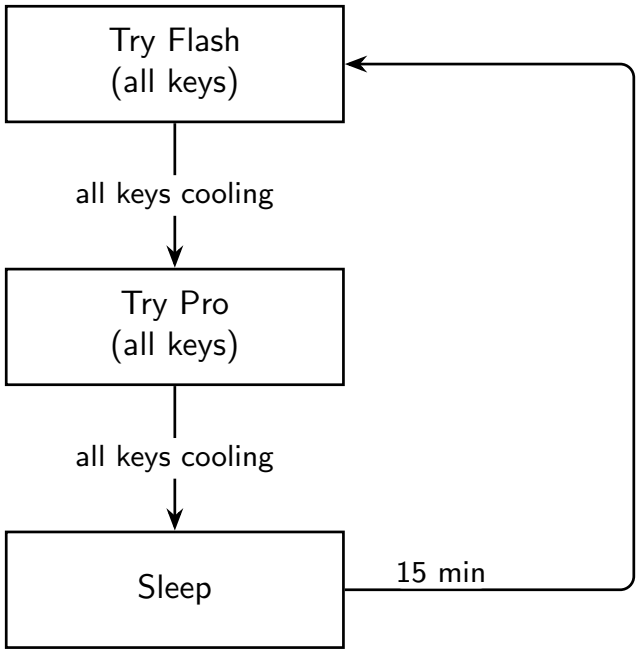

Figure 11: ModelKeyManager State Machine

evicted. To resolve this, we were forced to disable gradient checkpointing entirely. This decision presented a sharp trade-off: it stabilized the training loop but significantly increased VRAM consumption. Consequently, we had to strictly reduce the batch size to 1 on our development hardware (RTX 3060), relying on the 80GB VRAM headroom of the production H100s to maintain training throughput.

6 Conclusion & Future Work

This research concludes with a dichotomy of success and failure. On the engineering front, the **Data Factory** has proven itself as a robust scientific instrument. By enforcing ACID compliance through PostgreSQL and solving distributed concurrency via the "Ghost Lock" pattern, we successfully operationalized the ingestion and verification of code-switched audio at scale. The pipeline is stable, scalable, and produces high-fidelity metadata.

However, the **Scientific Experiment** delivered a harsh verdict on the feasibility of low-resource adaptation. Our attempt to train a "Triangle" E2E model from scratch using adapters on a 60-hour split resulted in non-convergence (97.5% WER), while the Whisper baseline succumbed to catastrophic hallucination loops (399% WER). We conclude that architectural elegance cannot compensate for data scarcity in the code-switching domain. The complex multitask objective we attempted simply requires a density of training signal that our current dataset size cannot yet provide.

6.1 Future Work

Our immediate trajectory shifts from architectural speculation to data-centric rigor. Having diagnosed the specific failure modes of "Aphasia" (E2E) and "Hyper-graphia" (Whisper), we propose a three-phase remediation strategy:

Scaling the Signal. The primary bottleneck is the volume of training data. We will leverage the proven throughput of the Data Factory to scale the corpus from 60 to **500 hours**. By ingesting a broader variance of sources—spanning casual vlogs to formal podcasts—we aim to stabilize the model's acoustic priors and bridge the domain gap that froze our E2E encoder.

Decomposing the Objective. The "Triangle" architecture, while theoretically sound, proved too brittle for

our current data scale. We will temporarily decouple the multitask objective, training dedicated ASR and Machine Translation models separately. Only once these baselines stabilize will we attempt to fuse them into a joint end-to-end system again.

Curriculum Learning. To address the infinite looping pathology observed in Whisper, we will implement a curriculum learning schedule. Rather than exposing the model to complex, overlapping code-switching immediately, we will warm up the weights on high-confidence, short segments. This gradual increase in difficulty should prevent the attention mechanism from fracturing early in training, mitigating the hallucination loops that dominated our current results.

7 Appendix

7.1 Database Design

The Data Factory relies on a relational schema designed for strict integrity and concurrency control. We utilize **SQLModel** (SQLAlchemy + Pydantic) to define these relations in Python code, which are then migrated to **PostgreSQL**.

Users Table Stores annotator identities and roles.

- `id` (PK): Integer ID.
- `username`: Unique identifier used for the `X-User-ID` header.
- `role`: Enum (`ADMIN`, `ANNOTATOR`).

Videos Table Represents the source material.

- `id` (PK): Integer ID.
- `original_url`: Unique constraint to prevent duplicate ingestion.
- `duration_seconds`: Used for chunk calculation.

Chunks Table The atomic unit of work.

- `id` (PK): Integer ID.
- `video_id` (FK): Link to parent video.
- `status`: State machine (`PENDING`, `REVIEW_READY`, `APPROVED`).
- `locked_by_user_id` (FK): The "Ghost Lock" owner.
- `lock_expires_at`: Timestamp for lock timeouts.

Segments Table The core dataset payload.

- `id` (PK): Integer ID.
- `chunk_id` (FK): Link to parent chunk.
- `start_time_relative`: Float seconds (0.0 – 305.0).
- `end_time_relative`: Float seconds.
- `transcript`: Code-switched text.
- `translation`: Vietnamese translation.

7.2 Gemini Prompt Engineering

The following prompts are injected into the Gemini 2.5 context window to enforce the linguistic requirements of the project.

7.2.1 System Prompt (Persona)

```
You are a Senior Linguistic Data Specialist and expert audio transcriptionist
focusing on Vietnamese-English Code-Switching (VECS).

Your role is to process audio files into precise, machine-readable datasets
for high-fidelity subtitling. You possess a perfect understanding of
Vietnamese dialects, English slang, and technical terminology.

Your core operating principles are:
1. PRECISION: Timestamps must be accurate to the millisecond relative to
   the start of the file.
2. INTEGRITY: Transcription must be verbatim. No summarization, no censorship.
3. TONALITY PRESERVATION: Your translations must adapt English terms into
   Vietnamese while strictly maintaining the speaker's original register,
   emotion, and sentence-final particles (e.g., á, nè, nhỉ, ha).
```

7.2.2 User Prompt (Instructions)

```
Your task is to transcribe the attached audio file and output the data
according to the provided JSON schema.

<context>
The output will be used for subtitles for a Vietnamese audience. The goal is
to make the content understandable (translating English) without losing the
"soul" of the original speech. The translation must feel like the speaker
switched to Vietnamese naturally, retaining all their original sass, anger,
or excitement.
</context>

<instructions>
1. **Analyze Audio**: Listen to the full audio to understand context.
2. **Segmentation**: Break speech into natural segments (2-25 seconds).
   - No gaps > 1 second unless silence/music.
3. **Transcription (Field: "text")**:
   - Transcribe exactly what is spoken.
   - Preserve code-switching (English stays English, Vietnamese stays Vietnamese).
   - **No Censorship**: Transcribe profanity, violence, or sensitive topics.
4. **Translation (Field: "translation")**:
   - **Target**: Translate English words/idioms into natural Vietnamese.
   - **Constraint**: Do NOT modify existing Vietnamese words, structures,
     or final particles.
   - **Proper Nouns**: Keep names/brands in English (e.g., "iPhone").
5. **Timestamping**: Format as "MM:SS.mmm". Ensure precision.
</instructions>

<translation_examples>
    [Examples omitted for brevity]
</translation_examples>
```

```
Process the audio now.
```

7.2.3 Output JSON Schema

```
RESPONSE_SCHEMA = {
    "type": "ARRAY",
    "description": "A list of transcribed and translated audio segments.",
    "items": {
        "type": "OBJECT",
        "properties": {
            "start": {"type": "STRING", "description": "MM:SS.mmm format"},
            "end": {"type": "STRING", "description": "MM:SS.mmm format"},
            "text": {"type": "STRING", "description": "Verbatim transcription"},
            "translation": {"type": "STRING", "description": "Tonality-preserved"}
        },
        "required": ["start", "end", "text", "translation"]
    }
}
```

7.3 Model Output Samples

To empirically validate the "Hyper-graphia" vs. "Aphasia" diagnosis, we present raw output samples from the test set. These examples were selected to represent the most common failure modes observed during the production run.

7.3.1 Whisper Failure

As illustrated below, the Whisper model frequently entered self-reinforcing repetition loops when encountering non-standard acoustic boundaries. Note how the prediction length explodes relative to the reference.

Sample #124 | WER: 1300%+
Diagnosis: Single-Token Repetition Loop

Reference: `nhưng mà kiểu em ba nhân cách gì`

Prediction: `thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế thế`

Sample #482 | WER: 800%+
Diagnosis: Phrase-Level Hallucination Loop

Reference: `Chao là một nhân cách Châu Anh tạo ra.`

Prediction: `Một người là một người là một người là một người là một người là một người là một người là một người là một người là một người là một người là một người là một người là một người là một người là một người là một người.`

7.3.2 E2E Failure

In contrast, the E2E model (Wav2Vec2 + mBART) suffered from a collapse in semantic mapping. The predictions are often generic, high-frequency "safety" phrases or complete non-sequiturs that bear no relation to the source audio.

Sample #918 | WER: 100.0%
Diagnosis: English Training Data Leakage (Hallucination)

Reference: `At first, I wanted to balance between work and study, of course.`

Prediction (ASR): `Podcasting is a new way to be a podcast.`

Prediction (ST): `Podcasting is a new way to be a podcast.`

Note: The model ignored the VECS content entirely and outputted a phrase likely memorized from the pre-training data of mBART or Wav2Vec2.

Sample #1061 | WER: 100.0%
Diagnosis: Generic Conversational Filler

Reference: `Thật ra là cha mình dặn mình lâu rồi, từ cái hồi mình tốt nghiệp rồi.`

Prediction (ASR): `cam kết với mọi người.`

Prediction (ST): `cam kết với mọi người.`

Sample #513 | WER: 126.3%
Diagnosis: "Safe" Guessing (High Probability Tokens)

Reference: `Sẽ là một người em trọn vẹn với thời gian hiện tại với với mẹ là em muốn ừm.`

Prediction (ASR): `Em nghĩ là em cũng không biết là cái này nó có đúng không nhưng mà em thấy nó có thể là một cái gì đó.`

Side note: This project a *terrible* reimplementation of the paper: Weller O, Sperber M, Pires T, Setiawan H, Gollan C, Telaar D, Paulik M. End-to-end speech translation for code switched speech. arXiv preprint arXiv:2204.05076. 2022 Apr 11.

Some lessons we learned after this project is that we have to select a better pretrained model, out of everything, and the hand annotation process is indeed, a nightmare. Also, the fact that we dedicated 70% of our time to build the database with robust backend logic and a user-friendly interface meant the project primarily focused on problem-solving, making the NLP part become (literally) a second thought.

Initially, we intended to implement a cascade Wav2Vec2 + 2 mBART, along with the E2E and Whisper model. However, due to time constraint (we did not annotate enough data in time) and later, hardware constraint, we decided to remove the cascade model. We thought that one H100 with 80GB VRAM was enough to fully finetune the 0.6B E2E model/ Whisper-medium but little did we know how naive it was. It consistently ran out of memory that we had to lower the original setting considerably.

But overall, it was still pretty fun and we did learn a lot when trying to build a full-stack solution from data engineering to model training, though the result is model hallucination and complete collapsing might not be preferable!

Source code can be found here: `https://github.com/chienlax/final_nlp`.