# Overlap Savings Analysis for Real-Time Voice Translation

> **Document Purpose**: Explain why pipeline overlap optimization provides zero benefit for real-time streaming, and identify potential architectural improvements.
>
> **Target Audience**: AI systems analyzing optimization opportunities
>
> **Last Updated**: 2026-02-19 21:32 HKT (Rev 3 - Week 0 Added)

---

## 1. Executive Summary

The current parallel pipeline (`orchestrator_parallel.py`) implements worker threads for VAD, ASR, Translation, and Output. However, **overlap savings = 0ms** in real-time streaming mode because the pipeline is fundamentally **I/O bound by human speech speed**, not **compute bound**.

**Key Insight**: Human speech (3-8 seconds per segment) is 5-10x slower than processing time (~0.7 seconds). When ASR finishes segment N, segment N+1 hasn't been fully spoken yet.

---

## 2. Timing Analysis

### 2.1 Real-Time Streaming Timeline

```
gantt
    title Real-Time Streaming: Sequential by Nature
    dateFormat X
    axisFormat %s

    section Human Speech
    Segment 1 (5s speech)       :active, speech1, 0, 5
    Silence (400ms)             :crit, silence1, 5, 5.4
    Segment 2 (4s speech)       :active, speech2, 5.4, 9.4

    section VAD Detection
    VAD: Seg 1 detected         :done, vad1, 5, 5.1
    VAD: Seg 2 detected         :done, vad2, 9.4, 9.5

    section ASR Processing
    ASR: Process Seg 1 (~450ms) :active, asr1, 5.1, 5.55
    ASR: WAITING (idle)         :crit, asr_wait, 5.55, 9.5
    ASR: Process Seg 2 (~450ms) :active, asr2, 9.5, 9.95

    section Translation
    MT: Translate Seg 1 (~250ms):active, mt1, 5.55, 5.8
    MT: WAITING (idle)          :crit, mt_wait, 5.8, 9.95
    MT: Translate Seg 2 (~250ms):active, mt2, 9.95, 10.2
```

**Observation**: Notice the large **idle gaps** (red segments). When ASR finishes at T=5.55s, the next segment hasn't even started being spoken (starts at T=5.4s, but VAD needs silence to detect it at T=9.4s).

### 2.2 Batch Processing Timeline (For Comparison)

```
gantt
    title Batch Processing: True Parallelism Possible
    dateFormat X
    axisFormat %s

    section VAD (File Scan)
    Detect ALL segments         :done, vad_batch, 0, 2

    section ASR Worker Pool
    ASR: Segment 1              :active, asr_b1, 2, 2.45
    ASR: Segment 2              :active, asr_b2, 2.45, 2.9
    ASR: Segment 3              :active, asr_b3, 2.9, 3.35

    section Translation Worker Pool
    MT: Segment 1               :active, mt_b1, 2.45, 2.7
    MT: Segment 2               :active, mt_b2, 2.7, 2.95
    MT: Segment 3               :active, mt_b3, 2.95, 3.2

    section Output
    Output Seg 1                :done, out_b1, 2.7, 2.8
    Output Seg 2                :done, out_b2, 2.95, 3.05
    Output Seg 3                :done, out_b3, 3.2, 3.3
```

**Observation**: In batch mode, all segments are available at T=0 (after VAD scan), enabling true pipeline parallelism.

---

## 3. Mathematical Constraint

### 3.1 Timing Parameters (from STATUS.md)

| Parameter | Symbol | Value | Description |
|---|---|---|---|
| Speech Duration | T_speech | 3-8s | Human speech segment length |
| Silence Duration | T_silence | 400ms | VAD silence threshold |
| ASR Processing | T_asr | ~450ms | faster-whisper base model |
| Translation | T_mt | ~250ms | MarianMT inference |
| Total Processing | T_process | ~700ms | ASR + MT |

### 3.2 Overlap Condition

For pipeline overlap to provide savings:

```
Next segment must be available before current processing finishes

T_available < T_process
```

**Real-time case:**

```
T_available = T_speech + T_silence = 3–8s + 0.4s = 3.4–8.4s
T_process = 0.7s

3.4–8.4s < 0.7s = FALSE ❌
```

**Batch case:**

```
T_available = 0 (all segments ready immediately)
T_process = 0.7s

0 < 0.7s = TRUE ✅
```

### 3.3 Worker Utilization

```
pie title Real-Time Worker Utilization
    "ASR Worker Idle (Waiting for Audio)" : 85
    "ASR Worker Active (Processing)" : 15
```

```
pie title Batch Worker Utilization
    "ASR Worker Idle" : 5
    "ASR Worker Active" : 95
```

---

## 4. Current Architecture Deep Dive

### 4.1 Parallel Pipeline Structure

```
flowchart TB
    subgraph AudioCapture["Audio Capture Thread"]
        A[Microphone Input] --> B[Ring Buffer]
    end

    subgraph VADWorker["VAD Worker (1 thread)"]
        B --> C{Speech Detected?}
        C -->|Yes| D[Buffer Speech Segment]
```

```
        C -->|No| C
        D --> E[Queue to ASR]
    end

    subgraph ASRWorker["ASR Worker (1 thread)"]
        E --> F[ASR Inference]
        F -->|2-thread pool| G[CTranslate2]
        G --> H[Queue to MT]
    end

    subgraph MTWorker["Translation Worker (1 thread)"]
        H --> I[MT Inference]
        I -->|2-thread pool| J[MarianMT/NLLB]
        J --> K[Queue to Output]
    end

    subgraph OutputWorker["Output Worker (1 thread)"]
        K --> L[Display/Log]
    end

    style VADWorker fill:#ffcccc
    style ASRWorker fill:#ffcccc
    style MTWorker fill:#ffcccc
    style OutputWorker fill:#ffcccc
```

### 4.2 The Blocking Problem

```
sequenceDiagram
    participant H as Human
    participant V as VAD Worker
    participant A as ASR Worker
    participant T as Translation Worker
    participant O as Output Worker

    Note over H,O: Segment 1 Begins

    loop Speech Duration (5s)
        H->>V: Audio chunks
        V->>V: Accumulating...
    end

    H->>V: Silence (400ms)
    V->>V: Detect end of speech
    V->>A: Segment 1 ready

    rect rgb(200, 255, 200)
        Note over A: ASR Active (450ms)
        A->>A: Process audio
    end
    A->>T: ASR result
```

```
    rect rgb(200, 255, 200)
        Note over T: MT Active (250ms)
        T->>T: Translate text
    end
    T->>O: Translation result
    O->>O: Output

    Note over H,O: ASR/MT Workers IDLE waiting for Segment 2

    rect rgb(255, 200, 200)
        Note over A,T: IDLE (~3-7s)
Next segment not yet spoken
    end

    loop Speech Duration (4s)
        H->>V: Audio chunks
        V->>V: Accumulating...
    end

    H->>V: Silence
    V->>A: Segment 2 ready
    Note over A: ASR starts again...
```

**Critical Observation**: The 4 workers run sequentially in practice because each waits for the previous stage to produce data, and data production is bound by human speech speed.

---

## 5. Fundamental Limitations

### 5.1 Why Current Parallel Architecture Can't Help

| Limitation | Explanation |
|---|---|
| **Sequential Data Dependency** | VAD must wait for speech+silence before producing a segment. ASR must wait for VAD. MT must wait for ASR. |
| **No Lookahead** | In real-time, you cannot see future audio. Batch can scan the entire file first. |
| **Speech-Processing Speed Mismatch** | Humans speak at ~150 words/min. ASR processes at ~1000 words/min (after receipt). |
| **VAD as Bottleneck** | VAD cannot detect segments faster than they are spoken. |

### 5.2 The "Idle Gap" Problem

```
Time:    0s    1s    2s    3s    4s    5s    6s    7s    8s    9s    10s
         |     |     |     |     |     |     |     |     |     |     |
Speech:  [▓▓▓▓▓▓▓▓▓▓▓▓ Segment 1 ▓▓▓▓▓▓▓▓▓▓▓▓▓▓]   [▓▓▓▓▓▓ Segment 2
▓▓▓▓▓▓]
         |     |     |     |     |     |     |     |     |     |     |
```

```
VAD:     └─────┴─────┴─────┴─────┴─────┴─────┘detected at 5.4s
                                             │     │     │     │     │
ASR:                                         └─────┘process
                                                   │     │     │     │
ASR Output:                                           done at 5.85s
                                                      │     │     │
Translation:                                          └─────┘process
                                                            │     │
Translation Output:                                            done at
6.1s
                                                               │
Output:                                                        └display

        ↑                                          ↑
    Speech starts                              Result ready
    at 0s                                      at 6.1s

    Total latency: 6.1s (mostly waiting for human to finish speaking!)
```

---

# 6. Potential Improvement Directions

## 6.1 Streaming/Incremental ASR

**Concept**: Process audio incrementally as it arrives, not waiting for segment end.

```
gantt
    title Streaming ASR: Potential for Overlap
    dateFormat X
    axisFormat %s

    section Human Speech
    Word 1 "Hello"              :active, w1, 0, 0.5
    Word 2 "world"              :active, w2, 0.6, 1.1
    Word 3 "today"              :active, w3, 1.2, 1.7

    section Streaming ASR
    Process Word 1              :done, sw1, 0.5, 0.7
    Process Word 2              :done, sw2, 1.1, 1.3
    Process Word 3              :done, sw3, 1.7, 1.9

    section Speculative Translation
    Start MT (partial)          :crit, mt_p1, 0.7, 0.9
    Update MT (partial)         :crit, mt_p2, 1.3, 1.5
    Finalize MT                 :active, mt_f, 1.9, 2.1
```

**Challenges**:

- ASR accuracy on partial audio (context loss)
- MT quality on incomplete sentences

- Stitching partial results

**Implementation**: faster-whisper supports `word_timestamps=True` and incremental decoding.

### 6.2 Sentence-Breaking VAD (Chunk-Based)

**Concept**: Break long sentences into smaller chunks (e.g., 2-second chunks) instead of waiting for full silence.

```
flowchart LR
    A[2s chunk 1] -->|ASR| B[Partial result]
    C[2s chunk 2] -->|ASR| D[Partial result]
    E[2s chunk 3] -->|ASR| F[Partial result]
    B --> G{Stitch}
    D --> G
    F --> G
    G --> H[Full sentence]
    H --> I[MT]
```

**Challenges**:

- Context window management for ASR
- Sentence boundary detection without full pause
- Overlapping chunk processing

### 6.3 Predictive Pre-processing

**Concept**: Use lightweight model to predict if speech is ending soon, start ASR early.

```
sequenceDiagram
    participant V as VAD + Predictor
    participant A as ASR
    participant T as Translation

    Note over V: Speech detected for 3s
    V->>V: Lightweight predictor:
"90% probability speech ends in 500ms"
    V->>A: Start ASR early (speculative)

    alt Speech actually ends
        A->>A: Result ready immediately!
        A->>T: Translation starts
    else Speech continues
        A->>A: Discard partial result
(wasted compute)
        Note over A: Trade-off:
Latency vs Compute
    end
```

**Challenges**:

- Prediction accuracy
- Wasted compute on false predictions
- Model size (must be very lightweight)

### 6.4 Pipeline Stage Interleaving

**Concept**: Process multiple speakers or audio sources (if available) to keep workers busy.

```
gantt
    title Multi-Source Interleaving
    dateFormat X
    axisFormat %s

    section Source A
    Speech A1                   :active, sa1, 0, 5

    section Source B
    Speech B1                   :active, sb1, 2, 7

    section ASR Worker
    Process A1                  :active, asra1, 5, 5.45
    Process B1                  :active, asrb1, 7, 7.45

    section Translation Worker
    Translate A1                :active, mta1, 5.45, 5.7
    Translate B1                :active, mtb1, 7.45, 7.7
```

**Note**: This only works for multi-source scenarios (e.g., conference calls), not single-speaker translation.

---

## 7. Key Questions for Analysis

When considering improvements, answer these:

1. **Is latency actually a problem?**
    - Current end-to-end: ~700-850ms after speech ends
    - Human perception threshold: ~200-300ms feels "instant"
    - But most latency is speech duration itself, not processing
2. **What's the real bottleneck?**
    - Compute: No (processing is 5-10x faster than speech)
    - I/O: Yes (waiting for audio)
    - Memory: No (models already loaded)
3. **Can we reduce speech duration dependency?**
    - Only by processing partial segments

- Trade-off: Quality vs Latency

4. **Is batch mode the actual use case?**
   - If translating files/videos, overlap optimization is valuable
   - If translating live speech, current architecture is near-optimal

---

## 8. Conclusion

**The overlap savings = 0ms is architecturally correct for real-time streaming.**

The parallel pipeline (`orchestrator_parallel.py`) is designed for throughput optimization (batch processing), but real-time streaming is fundamentally limited by the speed of human speech production.

**To achieve actual latency improvements in real-time mode**, you would need to:

1. Process audio **before** the speaker finishes (streaming ASR)
2. Accept **quality trade-offs** (partial context)
3. Implement **speculative execution** (risk wasted compute)

For the current single-speaker real-time use case, the architecture is already near-optimal. The parallel pipeline provides value for batch video/file processing, where overlap savings are real and significant.

---

## 9. References

- Current Pipeline: `src/core/pipeline/orchestrator_parallel.py`
- Performance Metrics: `STATUS.md` (Section "Performance Metrics")
- ASR Implementation: `src/core/asr/faster_whisper.py`
- VAD Implementation: `src/audio/vad/silero_vad_adaptive.py`

---

## 10. Update: Streaming Solution Design (REVISED)

After evaluation of architectural suggestions (see `docs/evaluation_streaming_suggestions.md`), a **Hybrid Streaming Mode with Partial Translation** has been designed.

### 10.1 The Solution: Hybrid Draft/Final Mode

```
Current (Sequential - No Meaning Until End):
Speech:  [████████████████████] (5s) → Silence → ASR → MT → "Hola mundo"
at 5.8s

Hybrid (Overlapped - Meaning Early):
Speech:       [████████████████████] (5s)
Draft ASR 1:        [ASR] → "Hello world"
Draft MT 1:             [MT] → "Hola mundo" at 2.75s ✓ MEANING!
Draft ASR 2:                 [ASR] → "Hello world today"
Draft MT 2:                      [MT] → "Hola mundo hoy" at 4.5s
```

```
Final:                                                    [ASR+MT] → "Hola mundo hoy"
at 5.3s
```

### 10.2 REVISED Key Design Decisions

| Aspect | Original | REVISED | Rationale |
|---|---|---|---|
| Draft interval | Every 2s | **Adaptive (every 2s, skip if paused)** | Reduce compute overhead |
| Draft confidence | Low (0.6) | Low (0.6) ✅ | Show grey/italic |
| **Draft translation** | No (ASR only) | **Yes (conditional on verb/punctuation)** | Users need **meaning**, not just words |
| **Compute strategy** | Standard | **INT8 for drafts, standard for final** | Manage 3x ASR overhead |
| **Context window** | Chunk-based | **Cumulative (0-N, not just N-2→N)** | Ensures grammatical consistency |
| **UI transition** | Replace | **Diff-based with highlight** | Smooth, show what changed |

### 10.3 Critical Improvement: Draft Translation with Semantic Gating

**Why Conditional Translation Matters**:

| Approach | T=2s | T=5.3s | User Experience |
|---|---|---|---|
| **Without Translation** | "Hello world..." (EN) | "Hola mundo" (ES) | Wait 5.3s for meaning ❌ |
| **With Translation** | "Hola mundo..." (ES) | "Hola mundo" (ES) | Get meaning at 2s ✅ |

**Semantic Gating Rules** (only translate if complete thought):

```python
def should_translate_draft(text, source_lang, target_lang):
    # Must have minimum content
    if len(text.split()) < 2:
        return False

    # Must have verb or punctuation (complete thought)
    has_verb = any(v in text.lower() for v in VERBS[source_lang])
    has_punct = any(text.endswith(p) for p in ['.', '!', '?', '。'])

    # Word order safety for SOV languages
    # SOV languages (Japanese, Korean) put verb at END
    # Translating before verb arrives = grammatical chaos
    SOV_LANGUAGES = ['ja', 'ko', 'de', 'tr']
    if target_lang in SOV_LANGUAGES:
        if not has_punct:
            return False  # Must wait for sentence end
```

```
    return has_verb or has_punct

# Examples:
# "Hello" → No translation (incomplete)
# "Hello world" → No translation (no verb)
# "I went" → Translate! (has verb)
# "Hello world." → Translate! (has punctuation)
# EN→JA: "I went to" → No (wait for Japanese verb at end)
# EN→JA: "I went to store." → Translate! (has punctuation)
```

**Language-Specific Rules**:

| Language Pair | Draft Translation | Notes |
|---|---|---|
| EN→ES, EN→FR | ✅ Yes | SVO→SVO (similar order) |
| EN→JA, EN→KO | ⚠ Conditional | SVO→SOV (wait for punctuation) |
| JA→EN | ✅ Yes | SOV→SVO (can translate early) |

### 10.4 Managing 3x Compute Overhead

**Problem**: 5s segment → Draft at 2s + Draft at 4s + Final at 5s = **3x ASR calls**

**Mitigation Strategy**:

| Technique | Implementation | Impact | Priority |
|---|---|---|---|
| **INT8 Quantization** | Draft: `compute_type="int8"` | 2x faster drafts | Week 2 |
| **Adaptive Skipping** | Skip draft if VAD detects pause >500ms | Reduce unnecessary calls | Week 2 |
| **Queue Monitoring** | Skip draft if queue depth > 2; Alert on overflow | Prevent backlog, **fix data loss** | 🔴 **Week 0** |
| **Cumulative Context** | Draft 2 reuses Draft 1 computation conceptually | Better quality per compute | Week 2 |
| **Error Logging** | No silent failures, comprehensive tracing | Debug sentence loss | 🔴 **Week 0** |
| **Platform Testing** | Intel i7 (OpenVINO), Mac M1 (CoreML) | Optimize INT8 per platform | 🔴 **Week 0** |

**Compute Estimate**:

```
Unoptimized: 3 × 450ms = 1350ms (3x overhead)
With INT8:   2 × 225ms + 450ms = 900ms (2x overhead)
With Adaptive: ~1.5x average (depends on speech pattern)
```

### 10.5 Expected Improvements

| Metric | Current | Target | Priority | Notes |
|---|---|---|---|---|
| **Sentence Loss Rate** | Bug exists (~?) | **0%** | 🔴 **Week 0 Critical** | Data integrity first |
| **TTFT** (any output) | ~5000ms | < 2000ms | 🟡 Week 2 | Draft visible |
| **Meaning Latency** (translated) | ~5000ms | **< 2000ms** | 🟡 Week 2 | ⭐ **Critical improvement** |
| **Ear-to-Voice Lag** | ~700ms | < 500ms | 🟢 Week 3 | Final optimization |
| **Compute Overhead** | 1x | ~1.5x | 🟢 Week 2 | Managed with INT8 + adaptive |

### 10.6 Implementation Plan (REVISED with Week 0)

See detailed plan: `docs/design/streaming_latency_optimization_plan.md`

**Week 0: CRITICAL - Fix Sentence Loss Bug 🔴**

> **MUST complete before any optimization!**

| Task | Deliverable |
|---|---|
| Add segment sequence tracking | UUID per segment, trace capture→VAD→ASR→MT→Output |
| Add queue depth monitoring | Alert if queue > 3 segments |
| Add comprehensive error logging | No silent failures anywhere |
| Stress test: 10-min continuous speech | Verify **0% sentence loss** |
| Fix root cause | Queue overflow? VAD threshold? Race condition? |
| Platform-specific testing | Intel i7: OpenVINO INT8, Mac M1: CoreML optimization |

**Week 1**: Metrics + Adaptive Config (TTFT/Lag, segment=4s, adaptive controller)
**Week 2**: StreamingASR (cumulative context, INT8 drafts) + StreamingTranslator (semantic gating)
**Week 3**: Diff-based UI (smooth transitions, stability indicators) + Integration

### 10.7 What Was Rejected

| Approach | Reason |
|---|---|
| **Wait-k Translation** | MarianMT doesn't support streaming; fails for different word order languages |
| **AsyncIO Rewrite** | Over-engineering; ThreadPool performs equally for ML tasks |

| Approach | Reason |
|---|---|
| **True Streaming ASR** | Requires model change (RNN-T); too much effort |
| **Draft Without Translation** | Provides "listening feedback" but not "meaning" - insufficient UX |

*Document written for AI analysis and improvement planning.*
*Updated 2026-02-19 with REVISED streaming solution (partial translation, semantic gating, INT8 optimization).*