

Date: 2025년 8월 28일   
Prepared by: Tiong-Sik Ng   
(Format per provided template.)

English 

## 1. Summary of Accomplishments

- Faster first audio: server now returns **first\_url** immediately while backgrounding remaining sentences; playback starts sooner.
- Barge-in preemption: added **/synthesize\_stream\_cancel** and cooperative checks so earlier speech stops as soon as a new interruption arrives.
- Sentence-by-sentence kept: one sentence → one `partN.wav` for natural pacing and fine-grained barge-in.
- Model speedups: enabled **torch.inference\_mode + AMP(float16)** and **compile once** at engine init; added warmup at startup.
- I/O optimizations: raised S3 **max\_pool\_connections**; trimmed per-sentence overhead; reduced unnecessary waits/polls where safe.
- Developer QoL: `direnv` workflows for Uvicorn/ngrok (auto setup, optional tmux/foreground), making restarts and resets simpler.

## 2. Issues & Risks

- Amazon Connect **Wait** block has a hard **2s minimum**; can't lower further, so perceived gaps rely on server/Lambda quickness.
- Occasional missed playback appears tied to **Lambda timing** (tight time limits vs transient slowness).
- Using **ngrok** introduces extra RTT; staying on NIPA infra limits some network-path optimizations.
- Aggressive Lambda timeouts can cause mid-call failures; needs careful balance.

## 3. Next Steps

- Add lightweight metrics: timestamps for **/start** → **part0**, inter-sentence gaps, Lambda durations; log to CloudWatch.
- Fine-tune Lambda **quick-poll** (e.g., 300–500 ms budget, 60–120 ms steps) to dodge the 2s Wait path more often.
- Consider **STREAM\_BATCH=2** (return 2 clips when available) and minor flow tweak to play both before looping—fewer round trips.
- Stabilize Lambda timeouts: keep **function** timeouts generous (10–15 s start / 6–10 s next) while **flow** invoke limits stay tighter.
- Optional: coalesce only ultra-short sentences (<~1.0 s) upstream when barge-in tolerance allows, to reduce micro-parts.

한국어 

## 1. 주요 성과

- 첫 오디오 가속: 서버가 **first\_url** 을 즉시 반환하고 나머지는 백그라운드로 생성하여 재생 시작이 빨라졌습니다.
- 바지-인 선점 종료: **/synthesize\_stream\_cancel** 및 협조적 취소 체크 추가로, 새 발화가 오면 이전 합성을 즉시 중단합니다.

- 문장 단위 유지: 문장 1 개 → 파일 1 개(`partN.wav`)로 자연스러운 흐름과 세밀한 바지-인 지점을 보장합니다.
- 모델 최적화: **inference\_mode + AMP(float16)** 적용, 초기 1 회 컴파일, 시작 시 워밍업으로 생성 지연 감소.
- I/O 최적화: S3 **max\_pool\_connections** 확대, 문장별 오버헤드 축소, 불필요한 대기/풀링 최소화.
- 개발 편의: Uvicorn/ngrok `direnv` 자동화(설치/인증/실행, tmux/포그라운드 옵션)로 재시작·리셋 작업 간소화.

## 2. 문제 / 리스크 ↗

- Amazon Connect **Wait** 블록은 최소 2 초 제약—더 낮출 수 없어, 체감 속도는 서버/람다 응답에 좌우됩니다.
- 간헐적 재생 누락은 람다 타이밍(과도한 제한 시간 vs 일시적 지연)과 연관 가능성.
- **ngrok** 사용으로 RTT 가 추가 발생; NIPA 인프라 유지 방침상 네트워크 경로 최적화에 한계.
- 람다 타임아웃을 과도하게 줄이면 호출 중단 위험—세심한 조정 필요.

## 3. 다음 단계 ❤️

- 타임스탬프 지표 추가: `/start-part0` 시간, 문장 간 간격, 람다 소요시간을 CloudWatch에 기록.
- 람다 쿼 풀 미세 조정(예: 총 300–500 ms, 간격 60–120 ms)으로 2 초 대기 경로 진입 최소화.
- **STREAM\_BATCH=2** 적용 검토(가능 시 두 클립 연속 재생 후 루프)로 왕복 호출 수 절감.
- 람다 함수 타임아웃은 여유(시작 10–15 s / 다음 6–10 s), 플로우 호출 제한은 타이트하게 유지.
- 선택: 바지-인 허용 범위에서 \*\*초단문(<~1.0 s)\*\*만 사전 병합하여 과도한 파일 분할 방지.