Take-home Exercise: Low-Latency Streaming ASR

Jonas Rochdi

October 3, 2025

1 Introduction

The goal of this exercise is to design a low-latency automatic speech recognition (ASR) system. The system should support real-time speech-to-text transcription in English, with optional support for German. The design must operate on a single NVIDIA T4 GPU. Mixed precision for inference is allowed to balance computational efficiency with recognition accuracy. The task states latency and throughput requirements for the system:
First partial token ≤ 300 ms. This metric measures the responsiveness of the asr system, i.e., how quickly it produces an initial hypothesis after receiving speech input.
Finalization after endpoint ≤ 150 ms. This metric captures the delay between the detection of a speech endpoint and the availability of a finalized transcription.
Real-time factor (RTF) ≤ 0.2. The RTF is defined as the ratio between the processing time and the input audio duration. It guarantees that the system can operate comfortably under real-time conditions and we have additional computational headroom.

2 System Design

2.1 Model Family Choice

We adopt the **Cache-Aware FastConformer Hybrid** model [2] from NVIDIA NeMo as the acoustic backbone. This is a streaming-optimized Conformer variant that combines convolutional subsampling with efficient attention, achieving SOTA accuracy while being faster than Conformer. The cache-aware mechanism restricts attention to a fixed context, so memory use is constant in input length and bounded by the chunk size.

As mentioned, the model employs a *cache-aware mechanism* to enable streaming inference, maintaining a limited right context during both training and inference. By caching and reusing intermediate activations across chunks, it eliminates redundant computations. The models supports lookaheads from fully causal (zero lookahead) to 1040 ms.

2.2 Architecture

We propose an architecture and explain each component in detail:
Preprocessing is the front end that turns raw waveform into acoustic features. Raw audio is collected in short frames, resampled (if necessary) to 16 kHz, converted to mono, and normalized. We use a feature extractor provided by NeMo to extract Mel spectrogram features. This module serves as the acoustic front end for the encoder.
The **FastConformer** serves as our acoustic model. It retains the Conformer block structure with feed-forward layers, multi-head self-attention, and convolutional blocks with residual connections, but introduces subsampling and reduced kernel sizes, yielding up to 2.4× faster inference while maintaining accuracy. Streaming is enabled via a fixed left cache and limited right lookahead, ensuring constant memory use and low latency, making the model suitable for real-time deployment on a single T4 GPU.

The **Hybrid RNNT–CTC Decoder** is used to map encoder representations to text. It consists of a standard recurrent neural network transducer (RNNT) decoder, a recurrent prediction network with an embedding and joint network, which produces subword tokens incrementally for streaming inference. A connectionist temporal classification (CTC) projection head is also attached directly to the encoder outputs, so the model works as both a CTC and RNNT model. We use greedy RNNT decoding as the default since RNNT offers more accurate, flexible frame-to-token alignment than CTC, and greedy search minimizes latency by avoiding beam search, making it suitable for real-time streaming.

Endpointing. The FastConformer Hybrid model does not natively include endpoint detection. For deployment, we adopt a lightweight VAD in which the stream is considered ended if no speech activity is observed for approximately 100 ms. This trailing margin balances low finalization latency with robustness against premature finalization.

Barge-in. Although our prototype focuses purely on ASR, we note that in deployment the system must handle user interruptions during ongoing playback, for example when used together with a TTS system. We assume interruption handling at the endpointing layer, where new speech activity immediately preempts ongoing output and is passed through the streaming pipeline without delay.

2.3 Latency

Key factors. Streaming latency in FastConformer arises from four sources: (i) the *lookahead size*, i.e. future context used, (ii) *chunking*, i.e. the duration of audio fed into the model per step, which is 160 ms in our system (the duration of a single output timestep from the decoder fixed to 80 ms for FastConformer models + the lookahead size), (iii) the *compute latency* of preprocessing, encoder, and decoder, and (iv) additional system-level overheads such as I/O.

First partial and Finalization latency. The time to the first partial token is approximately the chunk size plus compute latency. The time to finalize after endpointing is approximately the endpointing delay plus the lookahead and the compute latency for the final chunk. To stay within the overall targets (≤ 300 ms for first partial, ≤ 150 ms for finalization), we budget ≤ 50 ms per step for preprocessing and model inference, and ≤ 80 ms for endpointing (Table 1). This leaves comfortable headroom for overheads.

Stage	Budget (ms)
Preprocessing (per step)	≤ 10
Encoder + Decoder (per step)	≤ 40
Endpointing delay (per utt.)	≤ 100

Table 1: Latency budget per stage. Budgets apply to each 80 ms step (160 ms chunk), with an additional one-off endpointing delay.

3 Micro-Prototype

We build a micro-prototype and simulate streaming to evaluate our system. Simulation and experiments were conducted on a single NVIDIA RTX 4090 GPU.. The limitations of this setup and their implications for latency targets and real-time performance on a T4 are addressed in the discussion section.

3.1 Simulation

To simulate streaming over a dataset of audio recordings instead of a live microphone input, we feed the pre-recorded waveform files from the English test set of Common Voice v17.0 [1], which contains approximately 20 minutes of pre-recorded speech from diverse speakers across varying acoustic conditions, into the streaming interface. We start from a microphone-based NeMo example and adapt the input callback to incrementally push audio frames from a waveform file. This is functionally equivalent to online inference, except that the audio source is a file rather than a microphone. The system could therefore be used with a real live audio stream with little change to the code.

In a real microphone-based setup, audio is captured continuously and sliced into buffers (e.g., 80 ms frames) by the audio interface, which are then passed directly into the callback. In our simulation, the full waveform is first loaded into memory and then sliced into the same buffer size before being fed sequentially into the callback. We implement virtual pacing to simulate the real-time arrival and collection of audio frames to measure latency under realistic timing conditions without actually waiting in wall-clock time. For RTF, this pacing is disabled so that compute time is measured independently of simulated delays. For endpointing, we simplify by treating the end of file (EOF) as the end of speech. In a real streaming environment, endpointing would instead be handled by a VAD as described in Section 2.2.

3.2 Experiments

We evaluate streaming performance and accuracy. All timing is measured on the host to capture end-to-end latency, including both model compute and framework overheads, thus reflecting true application-level responsiveness.

For each utterance we record the raw and normalized Word Error Rate (WER) using the `jiwer` library, the first-partial latency , the preprocessing and model step latencies per chunk, the RTF (reported as median and 95th percentile for these three latency measures), and the peak GPU memory consumption measured with NVML.

Inference is performed with the PyTorch engine. To measure the onset of speech, we use WebRTC VAD [3]. We use this onset as the reference start time for first-partial latency measurements.

3.3 Ablation

We compare 0 ms and 80 ms lookahead. While the difference in latency on this system may be modest, demonstrating that 0 ms lookahead maintains comparable accuracy could indicate a substantial advantage on weaker GPUs such as the NVIDIA T4.

4 Results and Discussion

Lookahead	WER (%)	RTF p50 / p95	First-Partial p50 / p95	Peak GPU-Mem (GB)
80	19.9	0.11 / 0.12	220 / 380	1.55
0	22.2	0.21 / 0.21	199 / 380	1.54

Table 2: Streaming ASR evaluation results on a NVIDIA 4090. Lookahead and Latencies are reported in ms.

Lookahead	Stage	Budget (ms)	Measured p50 / p95 (ms)
80 ms	Preprocessing	≤ 10	0.5 / 0.5
	Encoder + Decoder	≤ 40	17.4 / 17.7
0 ms	Preprocessing	≤ 10	0.5 / 0.5
	Encoder + Decoder	≤ 40	15.9 / 17.6

Table 3: Latency budgets per 80 ms step (160 ms chunk) compared with measured latencies on NVIDIA 4090, for different lookahead settings.

- First-partial and RTF targets were generally met: median first-partial and RTF p50 stayed well under the 200 ms / 0.2× targets (Table 2). However, the p95 for first-partial latency reached 380 ms, which is higher than expected. This should be further investigated.
- Unexpected RTF difference: the 80 ms lookahead setting produced lower RTF compared to 0 ms lookahead (0.11 vs. 0.21, Table 2). This suggests that the extra lookahead stabilizes decoding and reduces recomputation, maybe there is a trade-off between latency and throughput here.
- Accuracy vs. latency trade-off: WER was slightly better for 80 ms lookahead than for 0 ms (19.9 % vs 22.2 %, Table 2).
- Latency budgets per chunk were comfortably within target (Table 3), leaving significant headroom versus the 50 ms budget.
- Hardware implications: since the RTX 4090 is substantially stronger than the target T4 (roughly 5–6× higher throughput and larger memory capacity), real deployment latencies will be higher. Still, the wide margin in Table 3 suggests feasibility, though a proper re-run on T4 is needed.
- Future directions: exploring TensorRT engines and mixed-precision inference could further reduce compute cost and lower both RTF and memory footprint.

5 Conclusion

We presented a micro-prototype of a streaming ASR system based on NeMo's FastConformer RNNT model. Measured preprocessing and inference times remained below budget on an RTX 4090, suggesting feasibility on the target T4 GPU, though a direct evaluation on T4 hardware is needed to confirm. The prototype demonstrates that cache-aware FastConformer with greedy RNNT decoding is a viable backbone for low-latency streaming ASR, and further gains are expected from deploying with TensorRT and mixed-precision inference.

References

[1] Rosana Ardila, Megan Branson, Kelly Davis, Michael Henretty, Michael Kohler, Josh Meyer, Reuben Morais, Lindsay Saunders, Francis M Tyers, and Gregor Weber. Common voice: A massively-multilingual speech corpus. *arXiv preprint arXiv:1912.06670*, 2019.

[2] Vahid Noroozi, Somshubra Majumdar, Ankur Kumar, Jagadeesh Balam, and Boris Ginsburg. Stateful conformer with cache-based inference for streaming automatic speech recognition. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 12041–12045. IEEE, 2024.

[3] John Wiseman. py-webrtcvad: Python interface to the webrtc voice activity detector. `https://github.com/wiseman/py-webrtcvad`. Accessed: 2025-10-02.