SUTD Deep Learning (50.039) Final Report

Keith Low, Luv Singhal, Lee Le Xuan

Abstract

Real-time speech transcription presents the challenge of determining the optimal moment to begin decoding, balancing accuracy with latency. Existing approaches such as wait-k [1] rely on fixed-delay heuristics, while Efficient Monotonic Multihead Attention (EMMA) [2] leverages a learned policy to enable adaptive, low-latency decoding. In this work, we present WhisperEMMA, which integrates EMMA into OpenAI's Whisper [3] to enable real-time transcription of audio balancing latency and accuracy.

1. Introduction

Simultaneous audio transcription transcribes sentences before they are finished, and is useful in many scenarios, enabling real-time applications such as live translation, closed captioning, and conversational artificial intelligence (AI). Unlike offline transcription systems that process all audio before generation output, simultaneous models must balance accuracy with latency.

This introduces a fundamental challenge in determining when to start the transcription without compromising quality. The introduction of SeamlessStreaming by Meta [4] broke new ground with EMMA [2], a new state-of-the-art attention mechanism tailored for simultaneous sequence generation tasks.

Integrating the new mechanism into general-purpose transcription models remains an open area of research, especially for lightweight architectures suitable for edge devices or resource-constrained deployment. In this work, we present WhisperEMMA. An augmentation of OpenAI's Whisper for [3] for real-time audio translation by modifying Whisper's text decoder with a simultaneous policy network using a modified version of EMMA [2] to enable real-time transcriptions that balances latency and accuracy.

2. Background

2.1. Whisper Architecture

Whisper [3] is a family of speech recognition models developed by OpenAI and released in September 2022. Each member of the Whisper family is a transformer-based model that features an audio encoder and a text decoder.

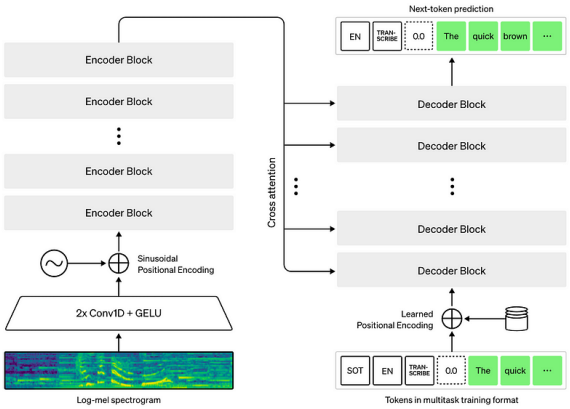

Figure 1: Whisper Model Architecture [5]

Whisper was trained on a large dataset of audio and text pairs, with audio files broken down into 30-second segments paired with their corresponding text transcription within the same time segment. For audio files longer than 30 seconds, a few methods were used.

The sequential method uses a "sliding window" approach to process the audio file, where the model is fed with a sequence of 30-second segments at a time. The chunked method splits long audio files into shorter ones, and stitch the transcriptions at their boundaries.

The current implementation of Whisper inference pipelines does not support real-time transcription of audio files. We propose augmenting Whisper's text decoder blocks inspired by the SeamlessStreaming [4] model to incorporate monotonic multihead attention using EMMA-style regularization. Specifically, we augment the decoder with policy network layers to encourage token alignment and interpolate the decoder's cross-attention values using modulated monotonic multihead attention weights.

2.2. Efficient Monotonic Multihead Attention

In November 2023, Meta Research released SeamlessStreaming, a model with a simultaneous text decoder that performs real-time transcription of audio files using a novel read-and-write policy [4].

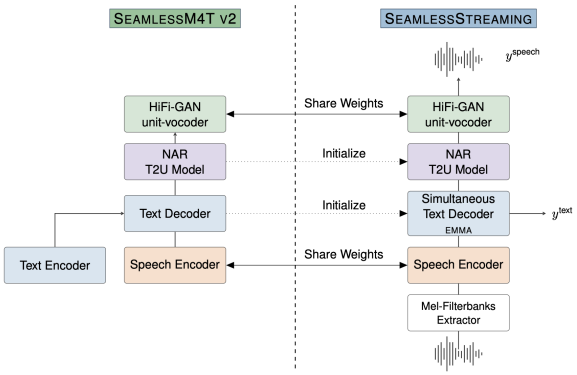

Figure 2: SeamlessStreaming Architecture [4]

The decoder is trained with a learned policy mechanism inspired by EMMA, which determines optimal output timings for generating transcriptions from a continuous audio stream.

Adapting monotonic attention to speech transcription presents two primary challenges. First, monotonic alignment estimation can suffer from numerical instability and introduce bias during training. Second, the encoder's continuous output space leads to high variance in alignment estimation [2].

EMMA addresses these issues by providing a numerically stable formulation, shaping the alignment path to encourage monotonicity, and enabling fine-tuning in streaming scenarios.

2.2.1. Numerically Stable Estimation

At a given time when $(i-1)$-th target translation has been predicted and j-th source input has been processed, a stepwise probability, denoted as $p_{i,j}$, describes the likelihood the model is going to write the i-th prediction rather than read the next input.

The monotonic alignment estimation, denoted as $a_{i,j}$, is to calculate the expected alignment from stepwise write probability $p_{i,j}$. The numerical instability arises from the denominator of previously proposed calculation of $a_{i,j}$, particularly when dealing with multiplication of several small probabilities.

The EMMA paper introduced this new calculation of the monotonic alignment between the i-th target item and j-th source item:

$$a_{i,j} = p_{i,j} \sum_{k=1}^{j} a_{i-1,k} \prod_{l=k}^{j-1} (1 - p_{i,l}) \tag{1}$$

This equation is numerically stable and unbiased, as it does not require a denominator as the product of probabilities within the equation [2].

2.2.2. Alignment Shaping

Without latency regularization, the naive and optimal policy produced by the policy network would be to `read` the entire sequence before `writing`. Thus, latency and variance regularization is added to control the trade-off between transcription accuracy and latency of the learned simultaneous transcription.

2.2.3. Simultaneous Finetuning

Training a model from scratch often requires substantial resources. The SeamlessStreaming framework [4] proposed using pre-trained SeamlessM4T v2 encoder and decoder weights to perform finetuning. The simultaneous text decoder weights were initialized from SeamlessStreaming text decoder weights, and the stepwise probability networks were initialized randomly. Furthermore, a negative bias was added to the stepwise probability networks to optimate the policy network from offline policy. Only supervised finetuning was performed from audio datasets.

3. Proposed Approach

3.1. Motivation

The primary objective of this work is to investigate the feasibility of incorporating real-time streaming capabilities, inspired by Seamless, into the Whisper Tiny model—an open-source, lightweight automatic speech recognition (ASR) system. By enabling efficient, low-latency streaming inference on Whisper-Tiny, we aim to explore its potential for deployment on resource-constrained edge devices. To the best of our knowledge, such a streaming adaptation has not yet been applied to Whisper Tiny, and this work represents an initial step toward bridging that gap.

3.2. Past Works

Several works have attempted to extend Whisper's capabilities to support real-time or streaming transcription, primarily through pipeline engineering rather than architectural changes. A notable example is the `Whisper_Streaming` project [6], which implements a sliding-window approach over the input audio. In this setup, audio is segmented into overlapping chunks and processed sequentially, with transcription corrections applied post-hoc using timestamp alignment heuristics. While effective for simple use cases, this method incurs significant I/O overhead and introduces latency due to repeated encoder passes and overlapping inference windows.

Other approaches have explored strategies such as buffering and staggered decoding, where the model generates partial transcriptions as audio arrives, occasionally revising previous outputs when more context becomes available. These methods, while practical, remain constrained by Whisper's original architecture, which was not designed for causal decoding or monotonic attention.

In contrast to these pipeline-level solutions, our work focuses on architectural augmentation, introducing streaming-aware components directly into the model, such as monotonic multi-head attention mechanisms and learned decoding policies. This enables the model to operate in a monotonic fashion and balances latency and accuracy through a learned policy.

3.3. Overview

Our work introduces a parallel stepwise probability network along each cross-attention layer in each text decoder block. At inference, an optimal policy is derived each time an audio chunk is streamed into the audio encoder and the text decoder is prompted to generate partial text transcriptions based on the learned policy. During training, we augment each cross-attention layer with monotonic multi-head attention weights inspired by EMMA. This modulation encourages the decoder to learn temporally localized alignments between the audio and text modalities, enabling more stable and efficient streaming transcription.

3.4. Working Model

For our working model, we selected Whisper Tiny [3] due to its lightweight nature and open-source availability, which makes it well suited for experimentation and potential deployment on edge devices with limited computational resources.

3.5. Efficient Monotonic Multihead Attention

We extend Whisper's text decoder with an EMMA-based framework that introduces three core components: numerically stable monotonic alignment estimation, alignment shaping, and streaming-specific finetuning [2]. Additionally, we propose a novel EMMA-style modulation mechanism that integrates monotonic attention directly into the cross-attention layers of Whisper, encouraging localized and efficient attention patterns during streaming inference.

3.5.1. Monotonic Alignment Estimation

In cross-attention, the text decoder is allowed to focus on all parts of the input sequence when making predictions. However, in a streaming setting, given an input sequence of length l, a text decoder at time step t, $t < l$ is only allowed to focus on the first t tokens of the sequence.

Monotonic alignment further constraints the attention to maintain a strict left-to-right alignment. This assumes that input sequences are monotonically increasing, following a sequential order. In our

case, the monotonic alignment in speech models aligns input audio frames to output text tokens in a sequential, left-to-right (according to time) order.

The objective of monotonic alignment estimation is to calculate the expected alignment $\alpha_{i,j}$, from the stepwise `write` action probability $p_{i,j}$ of the model, where i is the step of the target sequence and j is the step of the input sequence.

To train the stepwise probability network, we estimate the probability of alignment between partial output $y^{\text{text}}_{\leq i-1}$ and partial input $x^{\text{speech}}_{\leq j}$, i.e., the event that the source position is j when the output length is $i-1$. EMMA introduces a numerically stable estimation method to compute α in a parallel, closed-form expression

$$\alpha_{i,:} = \pi_{i,:} \odot \text{triu}_0 \left(\text{cumprod} \left(1 - \text{triu}_1 \left(\text{roll}_1(\pi_{i,:}) \right) \right) \right) \tag{2}$$

This formulation avoids denominators, enhancing numerical stability and reducing bias. Once α is estimated, we compute the soft monotonic attention weights $\beta_{i,j}$

$$\beta_{i,j} = \sum_{k=j}^{|x^{\text{speech}}|} \alpha_{i,k} \cdot \frac{e_{i,j}}{\sum_{l=1}^{k} e_{i,l}} \tag{3}$$

where $e_{i,j}$ is the attention energy between the i-th output and j-th input. Computed in parallel, we have

$$\beta_{i,:} = e_{i,:} \odot \text{flip} \left(\text{cumsum} \left(\text{flip} \left(\alpha_{i,:} \odot \frac{1}{\text{cumprod}(e_{i,:})} \right) \right) \right) \tag{4}$$

The SeamlessStreaming framework which proposes the addition of monotonic attention layers on top of SeamlessM4T v2 self-attention, where the attention of each head during training is expressed as

$$\text{Attention(Q,K,V)} = \beta V. \tag{5}$$

However, our work differs from the SeamlessStreaming framework because Whisper Tiny already incorporates cross-attention. Instead, we proposed an EMMA-style modulation mechanism that integrates monotonic attention directly into the cross-attention layers of Whisper, encouraging localized and efficient attention patterns during streaming inference. This modulation combines the original cross-attention output with the monotonic attention output through a weighted interpolation, formulated as

$$\text{ModulatedAttention} = (1 - \beta_{\text{weight}}) \cdot \mathbf{w}_v + \beta_{\text{weight}} \cdot \beta V \tag{6}$$

where $\mathbf{w}_v$ represents the standard cross-attention output computed using softmax attention weights over the value vectors $\mathbf{V}$, and βV denotes the output obtained through monotonic attention. The interpolation factor $\beta_{\text{weight}} \in [0, 1]$ is a hyperparameter which we use to balance between relying on full-context attention and enforcing more localized, monotonic patterns during training.

3.5.2. Alignment Shaping

EMMA focuses on the infinite lookback variant of monotonic attention. Hence, it is important to add a latency regularization in order to prevent the model from learning a trivial policy. This trivial policy, is simply for the model to wait and `read` the entire input sequence before making a prediction. By penalizing the model for waiting too long, we can control the tradeoff between accuracy and the latency of the learned simultaneous prediction policy.

To encourage low-latency decoding, we introduce a regularization term based on the expected delay of each target token. During training, the expected delay $\bar{d}_i$ for the i-th target token y_i is computed from the expected alignment distribution α as

$$\bar{d}_i = \mathbb{E}[j \mid i] = \sum_{k=1}^{|X|} k \cdot \alpha_{i,k} \tag{7}$$

Here, $\alpha_{i,k}$ denotes the probability of aligning y_i to the k-th input position, k represents the position within the input sequence, and $\bar{d}_i$ quantifies how far the model has waited before generating y_i.

Given a latency metric C, the loss term is computed as:

$$L_{latency} = C(\bar{D}) \tag{8}$$

The variance of the alignment characterizes the uncertainty of the alignment estimation. Denoting the expected variances of the monotonic alignment as $V = \bar{v}_1, \ldots, \bar{v_{|Y|}}$, the expected variance of the target token y_i, denoted as $\bar{v_i}$ can be expressed as

$$\bar{v_i} = E[j^2|i] - E[j|i]^2 = \sum_{k=1}^{|X|} k^2 \alpha_{i,k} - (\sum_{k=1}^{|X|} k\alpha_{i,k})^2 \tag{9}$$

To encourage more stable alignments and reduce temporal uncertainty during decoding, the alignment variance loss is introduced, defined as

$$L_{\text{variance}} = \sum_{i=1}^{|Y|} \bar{v}_i, \tag{10}$$

where $\bar{v}_i$ denotes the variance in alignment for the i-th target token. To further mitigate alignment variance, an enhanced stepwise probability network is proposed, parameterized as

$$p_{i,j} = \sigma\left(\frac{\text{FFN}_s(s_{i-1})^\top \text{FFN}_h(h_j) + b}{\tau}\right), \tag{11}$$

where s_{i-1} and h_j denote the decoder and encoder states at time steps $i-1$ and j respectively. The networks FFN_s and FFN_h are multi-layer feedforward projections used to compute energy scores, while b is a learnable bias initialized to a negative value to ease the transition from an offline policy, and τ is a temperature hyperparameter controlling the sharpness of the sigmoid output.

The final objective of the model incorporates a weighted combination of the negative log-likelihood and the two alignment regularizers, formulated as

$$L_\theta = -\log P(Y|X) + \lambda_{\text{latency}} L_{\text{latency}} + \lambda_{\text{variance}} L_{\text{variance}}, \tag{12}$$

where λ_{latency} and $\lambda_{\text{variance}}$ control the influence of latency and variance regularization respectively.

3.5.3. Simultaneous Finetuning

We define the simultaneous model as $M(\theta_e, \theta_d, \theta_p)$, where θ_e and θ_d represent the parameters of the encoder and decoder, respectively, and θ_p denotes the parameters of the policy network. During training, the encoder parameters θ_e are frozen, and optimization is performed only over the decoder θ_d and policy network θ_p. This design choice is guided by the assumption that the generative backbone—the encoder-decoder architecture—should retain characteristics consistent with the offline model. Instead, the model is adapted to operate under partial context by updating the decoder and policy components.

To assess the impact of different finetuning strategies, we consider two variants: WhisperEMMA-PChoose_Only, which updates only the policy network, and WhisperEMMA-Cross_Attn, which additionally finetunes the cross-attention value projections within the decoder.

3.5.4. Evaluation

The evaluation of the model is based on two factors: accuracy and latency.

To assess transcription accuracy, we obtain the BLEU (Bilingual Evaluation Understudy) score [7] and Word Error Rate (WER) score [8] by comparing the transcription with the ground truth. Scores are calculated for individual transcribed segments and then averaged over the evaluation set to reach an estimate of the transcription accuracy.

To assess speed, we measure the time taken to transcribe an audio file using Average Lagging (AL) [9].The goal of AL is to quantify the degree the user is out of sync with the speaker, in terms of the number of source words. Consistent with prior works, we define AL as

$$AL = \frac{1}{\tau(|y_i^{\text{text}}|)} \sum_{i=1}^{\tau(|y_t^{\text{text}}|)} d_i^{\text{text}} - d_i^* \tag{13}$$

where $\tau(|y^t ext|) = \min\{i | d_i^{\text{text}} = |x^{\text{speech}}|\}$ is the index of the first target translation when the policy first reaches the end of the source sentence. d_i^* is the ideal policy defined as

$$d_i^* = (i-1) \cdot \frac{|x^{\text{speech}}|}{|y^{\text{text}}|} \tag{14}$$

where y^{text} is the reference translation.

To estimate d_i^{text}, we approximate word-level alignments using the word-level timestamps produced by Whisper Tiny. These timestamps serve as our gold references for computing how many audio frames are read before each output word is generated.

3.6. Simulated Streaming Evaluation Pipeline

The streaming inference pipeline consists of four main components: an audio source handler, a log-scaled mel spectrogram feature extractor, the audio encoder, and the simultaneous text decoder.

To simulate real-time inference during evaluation, we emulate the online arrival of audio by streaming audio in a chunked, time-synchronized fashion. At each step, the decoder receives only partial encoder outputs corresponding to the current buffer state. This setup enables us to systematically assess the trade-off between latency and accuracy, as the model must make predictions under limited and incrementally expanding context.

3.7. Streaming Pipeline

The WhisperEMMA inference algorithm enables real-time speech transcription. Initializing text output state with a start-of-sequence token and given a real-time audio stream, audio chunks are encoded step-by-step using a speech encoder, producing partial encoder hidden states. At each step, the encoder states are passed onto the monotonic policy networks alongside previously generated tokens to produce a stepwise policy network action probability. If the score exceeds a predefined threshold t_{EMMA}, the model proceeds with next-token generation. Otherwise, it stalls to wait for more audio chunks, keeping track of stalled chunks. If the model stalls too many times as defined by a maximum stall count, it is forced to decode regardless of its confidence, preventing infinite waiting and cold starts.

3.7.1. WhisperEMMA *Inference Algorithm*

Algorithm 1 WhisperEMMA Inference Algorithm

Require: t_{EMMA}: Decision threshold for `read`/`write`
Input: Streaming Speech x^{speech}
Output: Streaming Text y^{text}

$i \leftarrow 1$, $j \leftarrow 0$, $k \leftarrow 0$, $y_0^{\text{text}} \leftarrow$ StartOfSequence, $c_{\text{stallCount}} \leftarrow 0$
$s_0 \leftarrow \text{TextDecoder}(y_0^{\text{text}})$
while not EndOfStream(x^{speech}) **do**
    $j \leftarrow j + 1$
    $h_{\leq j} \leftarrow \text{SpeechEncoder}(x_{\leq j}^{\text{speech}})$
    while $y_{i-1}^{\text{text}} \neq$ EndOfSequence **do**
        $p \leftarrow 1$
        for all StepwiseProbability in all attention heads **do**
            $p \leftarrow \min(p, \text{StepwiseProbability}(h_j, s_{i-1}))$
        end for
        if $p < t_{EMMA}$ **then**
            if $c_{\text{stallCount}} <$ MaxStallCount **then**
                $c_{\text{stallCount}} \leftarrow c_{\text{stallCount}} + 1$
            else
                $c_{\text{stallCount}} \leftarrow 0$
                $y_i^{\text{text}}, s_i \leftarrow \text{TextDecoder}(s_{<i}, h_{\leq j})$
                $k \leftarrow k + 1$
                $i \leftarrow i + 1$
            end if
            break
        else
            $y_i^{\text{text}}, s_i \leftarrow \text{TextDecoder}(s_{<i}, h_{\leq j})$
            $k \leftarrow k + 1$
            $i \leftarrow i + 1$
        end if
    end while
end while

4. Dataset

4.1. Dataset Used

For this project, we used the Common Voice 17.0 dataset [10]. It includes 31175 recorded hours of speech in 124 languages. For training, we are using the English dataset with around 114k rows of data. We chose this dataset as its audio lengths are more diverse, which allows us to simulate writeable utterances better.

For evaluation, we utilize the MLCommons People's Speech dataset [11], which comprises over 30,000 hours of transcribed English speech from a diverse set of speakers. This dataset was chosen to assess the generalization capability of our model beyond the training distribution. Furthermore, the relatively long audio durations, with each exceeding nine seconds help serve as a suitable benchmark for evaluating the effectiveness of our proposed streaming mechanism.

Due to compute limitations, we used smaller subsets from each for training, validation, and testing purposes during finetuning.

4.2. Training Preprocessing

The audio data is converted to a sampling rate of 16kHz. Human voice's most critical frequencies lie between 300Hz and 3400Hz, and a sampling rate of at least twice the highest frequency is required for accurate signal representation [12]. We chose to convert our audio data to 16kHz as was the sampling rate used in Whisper Tiny training [3].

5. Implementation

5.1. Workflow Breakdown

5.1.1. Code Base

We started with the Whisper repository from GitHub as the foundational code base. Building on the modified streaming module, we replaced the original decoder with a custom monotonic text decoder [3]. This decoder employs a monotonic residual attention block, consisting of self-attention layers followed by cross-attention layers. In addition, we integrated a policy network that uses q and k energy projections as feedforward networks, added as additional layers to the monotonic residual attention blocks. The policy network attends to each cross-attention head and generates an action probability, guiding the text decoder to either `read` or `write` at each step. During training, the policy network is used to modulate the process via the EMMA-style modulation mechanism as described in Section 3.5.1.

5.1.2. Data Preparation

Data preparation involved the use of data loaders to collate various features, such as converting audio arrays into tensors and extracting log-scaled mel spectrograms, which are then fed into the Whisper audio encoder. For evaluation, preprocessing was done on each example's transcription according to Whisper's set of English text normalization rules (refer to **Appendix B**) [3].

5.1.3. Training Loop

The training loop follows a typical setup for speech-to-text training, but with several customizations specific to our modifications. For each training batch, the following steps were executed:

- Model Training: For each iteration, provide encoder-ingested log-scaled mel spectrograms and teacher-forced sequence tokens to the monotonic text decoder.
- Monotonic Attention Modulation: Specific to the training of WhisperEMMA-Cross_Attn, the hyperparameter β_{weight} was eased into attention calculations as training steps progressed.
- Loss Computation: The losses were computed using the custom `MonotonicRegularizationLoss` module which accesses the computed stepwise policy network probabilities to produce monotonic alignment estimations used in monotonic regularization loss calculations.
- Gradient Update and Optimization: Training parameters were derived from the original Whisper paper [3], with some modifications to fit our compute limitations. The training parameters used can be seen in **Appendix C**.
- Validation: To monitor model's training, performance on a validation dataset was measured using the same loss functions and compared to the training loss.
- Checkpointing: Saved state weights periodically.

At each step, both the training and validation losses were logged for later analysis. The training process was monitored through loss curves which can be further broken down into `CrossEntropyLoss` and `MonotonicRegularizationLoss`. The training losses are reported in Section 5.2.2.

5.1.4. Hyperparameter Tuning

One key tunable hyperparameter in our system is the decision threshold for the probability *p_choose*, which determines whether the model should continue reading or start writing. To find an optimal threshold range (min_p, max_p) that balances latency and accuracy, we propose simulated audio chunking.

We simulate inference-time chunked audio input by feeding partial inputs to the model and observing stepwise policy network action probabilities at each timestep. We record values where the model makes the correct decision to either`read` or `write` and analyze the distribution of the probabilities across these two classes. This helps to estimate the threshold regions where decisions are most accurate. By controlling decision threshold values between min_p and max_p, a suitable decision threshold is determined by evaluating the model's performance on the given decision thresholds. Once a good threshold is found, it can be integrated into the inference loop as a lightweight gate to control output timing.

Training-related tunable hyperparameters we introduced include β_{weight}, $\lambda_{\text{latency}}, \lambda_{\text{variance}}$ in Section 3.5, controlling the WhisperEMMA-Cross_Attn's reliance on its pre-trained cross attention mechanism against monotonic attention weights and monotonic regularization losses respectively. However, the computation of monotonic attention weights during training time is non-trivial [2], and finding the optimal hyperparameters was a difficult task given our compute constraints.

5.2. Results

The performance of our proposed model is evaluated in comparison with two baselines: the original Whisper Tiny model and a `Whisper_Streaming` implementation with uses a local agreement policy with self-adaptive latency to enable streaming transcription [6]. These baselines were selected to assess the improvements in inference speed introduced by our approach, while ensuring that accuracy remains comparable.

5.2.1. Figures

Here are the results in table form:

Table 1: BLEU Score, WER, and Average Lagging

Model	BLEU Score	WER	Average Lagging
Whisper-Tiny (Original)	84.11	0.11	–
Whisper-Tiny (Whisper_Streaming)	61.29	0.32	**2.65**
Whisper-Tiny (WhisperEMMA-PChoose_Only)	79.95	0.15	3.48
Whisper-Tiny (WhisperEMMA-Cross_Attn-0.15)	78.87	0.15	2.96
Whisper-Tiny (WhisperEMMA-Cross_Attn-0.17)	82.15	0.13	4.32
Whisper-Tiny (WhisperEMMA-Cross_Attn-0.20)	83.28	**0.12**	5.91
Whisper-Tiny (WhisperEMMA-Cross_Attn-0.40)	**83.29**	**0.12**	5.92

The finetune of WhisperEMMA-PChoose_Only leads to great balanced performance with a much lower WER of 14% compared to the SOTA Whisper_Streaming without too much added lag (only 1s).

Looking at the finetune of WhisperEMMA-Cross_Attn, the model at different decision thresholds WhisperEMMA-Cross_Attn-0.20 and WhisperEMMA-Cross_Attn-0.40 performed the best in terms of accuracy, with the lowest streaming WER of 0.12 in both cases. However, these models incurred higher latency (AL $\approx$ 5.9), indicating a delay in output generation due to more conservative `read` decisions. Reducing the cross-attention threshold to 0.17 further lowered latency to 4.32, while maintaining a strong BLEU score of 82.15 and a reduced WER of 0.13, suggesting it may offer a better quality-latency trade-off. Further reducing the threshold, WhisperEMMA-Cross_Attn-0.15 variant achieved the lowest latency among WhisperEmma-Cross_Attn variants (2.96 AL), but at the cost of lower BLEU score (78.87) and higher WER (0.15).

In contrast, the Whisper_Streaming baseline saw a sharp drop in translation quality (BLEU score 61.29, WER 0.32), despite having the lowest average lagging (2.65), with the WhisperEMMA-Cross_Attn-0.15 performing almost as well in terms of latency but with significantly higher accuracy.

Overall, the results demonstrate that WhisperEmma-Cross_Attn with cross-attention thresholds between 0.17 and 0.20 offers a compelling balance, achieving near-offline quality without adding significant latency compared Whisper_Streaming.

5.2.2. Losses

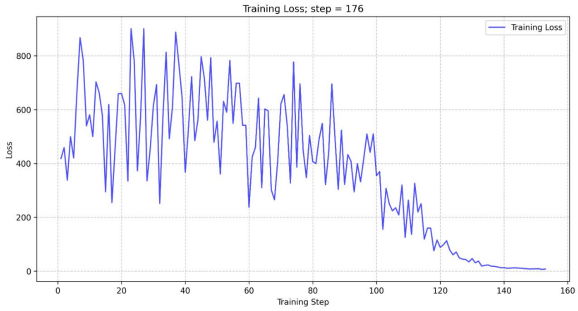

Figure 3: Training Loss for WhisperEMMA-PChoose_Only

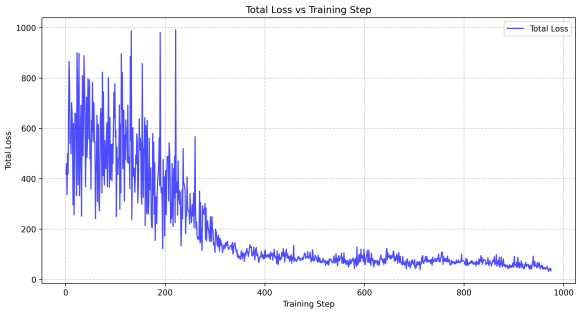

Figure 4: Training Loss for WhisperEMMA-Cross_Attn

6. Limitations and Challenges

6.1. Buggy KV Caching

Key-Value (KV) caching is a technique used to optimize the inference performance of transformer models by storing and reusing intermediate key and value states from previous decoding steps. As outlined by Hugging Face [13], KV caching enables significant speedups, particularly in auto regressive generation tasks, by avoiding redundant computations. However, integrating KV caching into existing architectures can introduce complexity and fragility.

In our work with the Whisper code base, incorporating KV caching led to numerous bugs and unexpected behaviors. This is primarily due to the way KV caching is implemented in Whisper—it is highly decoupled from the core model logic and instead managed externally. As a result, integrating KV caching required meticulous coordination, particularly across multiple read iterations in the EMMA architecture. The lack of native support for tightly integrated caching made the system prone to state mismatches, stale memory, and subtle bugs, all of which required substantial engineering effort to mitigate.

7. Work Distribution

Keith Low - Implementation of EMMA decoder, data sourcing, training loop, training and evaluation of models.

Luv Singhal - Implementation of streaming pipeline and EMMA decoder, training and evaluation of models.

Lee Le Xuan - Structure, contents of report, documentation, evaluation of models and research on implementation.

8. Conclusion

In this work, we explored the feasibility of enabling real-time speech translation through the integration of EMMA into Whisper Tiny. We trained two finetuned variants of WhisperEMMA, WhisperEMMA-PChoose_Only, which updates only the policy network, and WhisperEMMA-Cross_Attn, which additionally finetunes the cross-attention value projections within the decoder. Our results demonstrate that while the baseline Whisper_Streaming provides minimal latency, it suffers from a significant drop in translation quality. In contrast, EMMA-enhanced models, particularly WhisperEMMA-Cross_Attn with decision thresholds between 0.17 and 0.20, achieve near-offline translation performance with competitive latency, offering a more favorable quality-latency trade-off. This suggests that EMMA can serve as a practical approach for scaling up to other Whisper models and streaming translation, making real-time deployment on resource-constrained environments more viable. Future work could involve further latency optimization, multilingual extensions, and on-device deployment testing.

References

[1] Mingbo Ma et al. "STACL: Simultaneous Translation with Implicit Anticipation and Controllable Latency using Prefix-to-Prefix Framework". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Ed. by Anna Korhonen, David Traum, and Lluís Màrquez. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 3025–3036. DOI: 10.18653/v1/P19-1289. URL: https://aclanthology.org/P19-1289/.

[2] Xutai Ma et al. *Efficient Monotonic Multihead Attention*. 2023. arXiv: 2312.04515 [cs.CL]. URL: https://arxiv.org/abs/2312.04515.

[3] Alec Radford et al. *Robust Speech Recognition via Large-Scale Weak Supervision*. 2022. arXiv: 2212.04356 [eess.AS]. URL: https://arxiv.org/abs/2212.04356.

[4] Seamless Communication et al. *Seamless: Multilingual Expressive and Streaming Speech Translation*. 2023. arXiv: 2312.05187 [cs.CL]. URL: https://arxiv.org/abs/2312.05187.

[5] Louis Bouchard. *openAI's Most Recent Model: Whisper (explained)*. 2022. URL: https://www.louisbouchard.ai/whisper/.

[6] Dominik Macháček, Raj Dabre, and Ondřej Bojar. *Turning Whisper into Real-Time Transcription System*. 2023. arXiv: 2307.14743 [cs.CL]. URL: https://arxiv.org/abs/2307.14743.

[7] Kishore Papineni et al. "Bleu: a Method for Automatic Evaluation of Machine Translation". In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Ed. by Pierre Isabelle, Eugene Charniak, and Dekang Lin. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, July 2002, pp. 311–318. DOI: 10.3115/1073083.1073135. URL: https://aclanthology.org/P02-1040/.

[8] Shammur Absar Chowdhury and Ahmed Ali. "Multilingual Word Error Rate Estimation: E-Wer3". In: *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2023, pp. 1–5. DOI: 10.1109/ICASSP49357.2023.10095888.

[9] Mingbo Ma et al. “STACL: Simultaneous Translation with Implicit Anticipation and Controllable Latency using Prefix-to-Prefix Framework”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Ed. by Anna Korhonen, David Traum, and Lluís Màrquez. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 3025–3036. DOI: 10.18653/v1/P19-1289. URL: https://aclanthology.org/P19-1289/.

[10] R. Ardila et al. “Common Voice: A Massively-Multilingual Speech Corpus”. In: *Proceedings of the 12th Conference on Language Resources and Evaluation (LREC 2020)*. 2020, pp. 4211–4215.

[11] Daniel Galvez et al. *The People’s Speech: A Large-Scale Diverse English Speech Recognition Dataset for Commercial Usage*. 2021. arXiv: 2111.09344 [cs.LG]. URL: https://arxiv.org/abs/2111.09344.

[12] picovoice. *Audio Sampling and Sample Rate*. 2024. URL: https://picovoice.ai/blog/audio-sampling-and-sample-rate/#:~:text=The%20Nyquist%2DShannon%20sampling%20theorem,using%20human%20speech%20and%20voice..

[13] Hafedh Hichri. *KV Caching Explained: Optimizing Transformer Inference Efficiency*. 2025. URL: https://huggingface.co/blog/not-lain/kv-caching.

Appendix A. Running Code

Appendix A.1. Package Dependencies

```
numba
numpy
torch
tqdm
more-itertools
tiktoken
triton>=2.0.0;platform_machine=="x86_64" and sys_platform=="linux" or sys_platform=="linux2"
datasets
librosa
soundfile
transformers
```

Appendix A.2. Dataset

Training: Common Voice 17.0 (Subset:en Split:Train)

https://huggingface.co/datasets/mozilla-foundation/common_voice_17_0/viewer/en?views%5B%5D=en_train

Evaluation: MLCommons (Subset:Test Split:Test)

https://huggingface.co/datasets/MLCommons/peoples_speech/viewer/test?views%5B%5D=test

Appendix A.3. User Instructions

We used Python 3.10, PyTorch 2.1.0, CUDA 11.8.0 to train and test our models, but the codebase is expected to be compatible with Python 3.8-3.11 and recent PyTorch versions. The codebase also depends on a few Python packages, most notably OpenAI's `tiktoken` for their fast tokenizer implementation. To clone the code repository and update package requirements, please run:

```
git clone https://github.com/keeve101/whisper-emma
pip install -r requirements.txt
```

It also requires the command-line tool ffmpeg to be installed on your system, which is available from most package managers:

```
# on Ubuntu or Debian
sudo apt update && sudo apt install ffmpeg

# on Arch Linux
sudo pacman -S ffmpeg

# on MacOS using Homebrew (https://brew.sh/)
brew install ffmpeg

# on Windows using Chocolatey (https://chocolatey.org/)
choco install ffmpeg
```

```
# on Windows using Scoop (https://scoop.sh/)
scoop install ffmpeg
```

The following sections are with reference to our code base.

Appendix A.3.1. Code Implementations
The computations for α are implemented in `PChooseLayer._monotonic_alignment`.

The computations for p_{choose} are implemented in `MonotonicTextDecoder.decode_with_pchoose`.

The computations for β and the modulation mechanism are implemented in `MultiHeadAttention.forward`.

Appendix A.3.2. Finetuning
Our finetuning script is available in `finetune-whisper-policy-network.py`.

All detailed explanations can be found in our report `report.pdf`, including training and evaluation procedures.

To run finetuning, execute:

```
python finetune-whisper-policy-network.py
```

You may specify model checkpoints and training parameters within the script. Further details of the training parameters are included in the report.

Our finetuned model weights are available on HuggingFace

https://huggingface.co/keeve101/whisper-emma

Appendix A.3.3. Evaluation
To evaluate the model, first checkout to the evaluation branch:

```
git checkout evaluation
```

To evaluate the finetuned model on a test example:

```
python test.py
```

To run full evaluation on the test set and obtain results:

```
python get_results.py
```

For comparison with the baseline `whisper_streaming` implementation run:

```
python test_whisper_streaming.py
```

Appendix B. Text Standardization [3]

Since Whisper may output any UTF-8 string rather than a restricted set of graphemes, the rules for text standardization need to be more intricate and comprehensive than those defined on, e.g., ASCII characters. We perform the following steps to normalize English texts in different styles into a standardized form, which is a best-effort attempt to penalize only when a word error is caused by actually mistranscribing a word, and not by formatting or punctuation differences.

Appendix B.1. English Text Normalization

1. Remove any phrases between matching brackets ([,]).
2. Remove any phrases between matching parentheses ((,)).
3. Remove any of the following words: *hmm*, *mm*, *mhm*, *mmm*, *uh*, *um*.
4. Remove whitespace characters that come before an apostrophe (').
5. Convert standard or informal contracted forms of English into the original form.
6. Remove commas (,) between digits.
7. Remove periods (.) not followed by numbers.
8. Remove symbols as well as diacritics from the text, where symbols are characters with the Unicode category starting with M, S, or P, except for period, percent, and currency symbols that may be detected in the next step.
9. Detect any numeric expressions of numbers and currencies and replace with a form using Arabic numbers, e.g., *"Ten thousand dollars"* → *"$10000"*.
10. Convert British spellings into American spellings.
11. Remove remaining symbols that are not part of any numeric expressions.
12. Replace any successive whitespace characters with a single space.

Appendix C. WhisperEMMA Parameters

Table C.2: WhisperEMMA-PChoose_Only Training Hyperparameters

Hyperparameter	**Value**
Updates	175
Batch Size	1
Warmup Updates	50
Max grad norm	1.0
Optimizer	AdamW
β_1	0.9
β_2	0.98
ϵ	10^{-6}
Weight Decay	0.1
Learning Rate	1.5e-3
Learning Rate Schedule	Linear Decay
λ_{latency}	0.5
$\lambda_{\text{variance}}$	0.5

Table C.3: WhisperEMMA-Cross_Attn Training Hyperparameters

Hyperparameter	**Value**
Updates	975
Batch Size	1
Warmup Updates	50
Max grad norm	1.0
Optimizer	AdamW
β_1	0.9
β_2	0.98
ϵ	10^{-6}
Weight Decay	0.1
Learning Rate	2e-5
Learning Rate Schedule	Linear Decay
λ_{latency}	0.5
$\lambda_{\text{variance}}$	0.5
Max β_{weight}	0.5