

Moonshine: Speech Recognition for Live Transcription and Voice Commands

Nat Jeffries Evan King Manjunath Kudlur Guy Nicholson James Wang Pete Warden

Useful Sensors

Abstract

This paper introduces Moonshine, a family of speech recognition models optimized for **live transcription and voice command processing**. Moonshine is based on an encoder-decoder transformer architecture and employs Rotary Position Embedding (RoPE) instead of traditional absolute position embeddings. The model is trained on speech segments of various lengths, but **without using zero-padding, leading to greater efficiency for the encoder during inference time**. When benchmarked against OpenAI’s Whisper `tiny.en`, Moonshine Tiny demonstrates a 5x reduction in compute requirements for transcribing a 10-second speech segment while incurring no increase in word error rates across standard evaluation datasets. These results highlight Moonshine’s potential for real-time and resource-constrained applications.

1. Introduction

Real-time automatic speech recognition (ASR) is essential for many applications, including live transcription during presentations, accessibility tools for individuals with hearing impairments, and voice command processing for conversational interfaces in smart devices and wearables. These applications often run directly on low-cost hardware, where strict resource constraints and a lack of internet connectivity introduce unique technical challenges that are not present in other ASR domains.

The introduction of OpenAI’s Whisper has led to a significant improvement in the accuracy of general-purpose ASR systems (Radford et al., 2022). However, one of the primary challenges in applications of *on-device* ASR is the need to minimize latency—the time delay between a spoken utterance and the corresponding text appearing, e.g., on a live transcription display—without losing accuracy. During

our development of one such application—a Caption Box built to provide rapid, accurate, and private offline transcription of English speech—we found that current models are ill-suited to the task. When deployed on a low-cost ARM-based processor, we noticed that even the smallest Whisper `tiny.en` model had a firm lower latency bound of 500 milliseconds, irrespective of the duration of audio. User feedback indicated that this level of latency failed to provide a smooth and responsive user experience, which motivated us to investigate deeper.

Whisper, which is based on an encoder-decoder transformer architecture, processes fixed-length audio sequences in 30-second chunks regardless of the actual speech content of the audio. This means that shorter audio sequences require padding with zeros to meet the length requirement, resulting in a constant computational overhead in the encoder. While the decoder’s processing time is proportional to the length of the utterance, the constant overhead of fixed-length encoding places a firm lower bound to latency—e.g., the 500 milliseconds we identified during our own testing with the Caption Box. Our initial attempts to remove this bottleneck involved fine-tuning and distilling¹ Whisper models to handle variable-length sequences in the encoder, utilizing open audio datasets. However, these open datasets proved insufficient to surpass Whisper’s Word Error Rate (WER). Recognizing the limits of available data and the opportunity to leverage recent advancements in model architectures, we opted to develop and train new models from scratch.

Our first task was to better quantify the bottleneck imposed by Whisper’s fixed-length encoder. In an empirical test, we compared the GFLOPS (i.e., billion floating-point operations) needed for processing 30 seconds of zero-padded audio to the GFLOPS that would be required to instead process only a < 30 second subset of the same audio—i.e., we measured the potential for speed-ups if Whisper instead had a *variable*-length encoder. Our results (Figure 1) showed potential for a 35x speed-up in the best case, and nearly a 5x speed-up overall. Clearly, improvements to Whisper’s architecture could yield lightweight, low-latency speech-to-

. Correspondence to: Manjunath Kudlur <keve-man@usefulsensors.com>.

¹Hence the name Moonshine. We progressed to not use distillation eventually, but the name stuck.

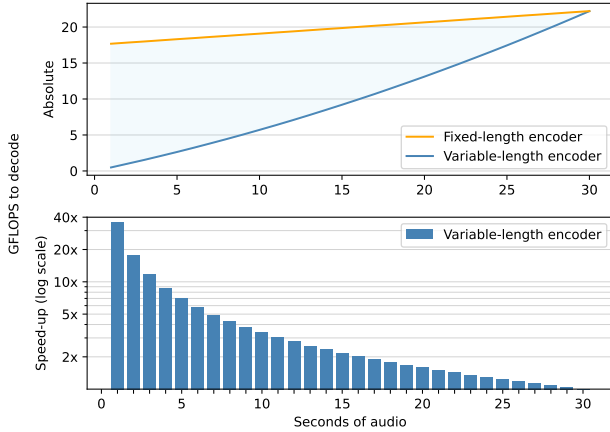


Figure 1. GFLOPS required by Whisper `tiny.en` for decoding as a function of input audio duration. OpenAI’s Whisper models use a fixed-length encoder that requires any audio less than 30 seconds to be zero-padded; testing with a hypothetical *variable-length* encoder shows that we can attain significant speed-ups by removing this requirement.

text models that are better-suited to resource-constrained applications.

Toward tackling the challenges faced by on-device, low-latency ASR applications, this paper introduces the Moonshine family of ASR models. Moonshine models are designed to match Whisper’s accuracy while optimizing computational efficiency by eliminating zero-padding requirements, instead scaling processing demands proportionally to audio input length. We describe Moonshine’s architecture and training process, and provide a detailed analysis of its performance benefits in comparison with Whisper models. Section 2 motivates our development of Moonshine by quantifying the WER when adapting Whisper for variable-length audio. Section 3 describes the Moonshine architecture, dataset preparation, and training process, while Section 4 provides an evaluation of the results on standard speech recognition datasets. Section 5 concludes.

2. Issues with fixed sequence length encoder

Whisper models use absolute position embeddings in both the encoder and decoder. The encoder in particular uses sinusoidal position embeddings of shape $[1500, \text{dim}]$, where dim is the transformer encoder’s dimension. The standard inference code provided by OpenAI zero-pads the input audio to an exact 30-second segment before passing it down to the model. Audio feature generation and 2 convolution layers convert this audio segment to a $[1500, \text{dim}]$ vector sequence that gets added to the sinusoidal position embedding. The left-most column of Figure 2 illustrates this mechanism.

Requiring an exact 30-second segment causes the encoder to incur a fixed computation cost, no matter the duration of actual speech in the audio segment. However, since Whisper’s model architecture is an off-the-shelf, variable-length sequence-to-sequence transformer, zero-padding is not strictly necessary. An audio segment that is shorter than 30 seconds can indeed be processed through the model’s frontend to yield a $[\text{seq_length}, \text{dim}]$ vector sequence where $\text{seq_length} \leq 1500$.

There are several options for adding the position embedding to a variable-length audio input. We compared the default implementation in OpenAI’s Whisper inference code (i.e., using zero-padding to form 30-second segments) with two other options, as illustrated in the center and right-most columns of Figure 2—namely, using the prefix and the suffix of the position embedding. We computed the WER for the `tiny.en` model using these two options on the `test.clean` split of the Librispeech dataset. This split has 2620 examples, 9 of which are more than 30 seconds in length. We ignore these 9 examples in this evaluation. We use a beam size of 5, so as to not suffer the pitfalls of greedy decoding such as repetitions. With the standard inference code provided by OpenAI, we get a WER of 5.21%. When the input is *not* zero-padded, however, we get worse results. When using only a prefix of the position embedding, the `tiny.en` model has a WER of 107.38%. We notice a lot of repetitions towards the end of the transcription, even with the beam search. When instead using the suffix of the position embedding, the accuracy degradation is not as dramatic, producing a WER of 18.45%—still more than three times worse than the baseline.

These results indicate that the OpenAI Whisper models are trained with examples that are all exactly 30 seconds long. The models indeed produce high quality transcriptions for long-form audio. However, the fixed computation budget needed for their encoder makes them inefficient for low-latency applications, such as live transcription. Since the audio segments in these applications tend to be short and/or vary in length, our effort to train models optimized for variable-length sequences is well-motivated.

3. Approach

This section describes Moonshine’s model architecture before detailing our data collection, preprocessing, and model training process.

3.1. Model architecture

Moonshine is an off-the-shelf encoder-decoder Transformer (Vaswani et al., 2017) model. The input is an audio signal sampled at 16,000 Hz. We do not use any hand-engineering (e.g., Mel spectrograms) to extract audio features. Instead,

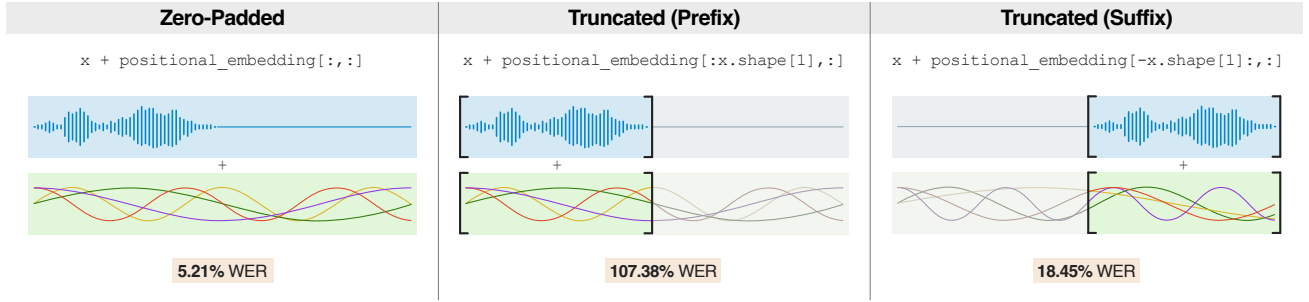


Figure 2. Variations of absolute position embeddings usage and the corresponding WER produced by `Whisper tiny.en` (test.clean split of the Librispeech dataset). Simply adapting Whisper’s inference code to avoid encoding of fixed-length audio (as in the center and right-most columns) introduces significant increases in WER, motivating our development of new models with variable-length encoding.

the input is processed by a short stem of 3 convolution layers with strides 64, 3, and 2 respectively. The filter widths, number of channels and activation functions are shown in Figure 3. These set of strides compress the input by a factor of 384x. Note that in the Whisper model, the input is compressed by 320x—firstly 160x in the Mel spectrogram computation, then by a further 2x in the convolution stem. We use Rotary Position Embeddings (RoPE) (Su et al., 2023) at each layer of the encoder and decoder. Table 1 compares Moonshine’s architecture with OpenAI’s Whisper models.

We use the same byte-level BPE text tokenizer used in Llama 1 and 2 (Touvron et al., 2023; Sennrich et al., 2015) for tokenizing English text. The original vocabulary size is 32000; we add 768 special tokens for future extensions.

3.2. Training data collection & preprocessing

We train Moonshine on a combination of 90K hours from open ASR datasets and over 100K hours from own internally-prepared dataset, totalling around 200K hours. From open datasets, we use Common Voice 16.1 (Ardila et al., 2020), the AMI corpus (Carletta et al., 2005), GigaSpeech (Chen et al., 2021), LibriSpeech (Panayotov et al., 2015), the English subset of multilingual LibriSpeech (Pratap et al., 2020), and People’s Speech (Galvez et al., 2021). We then augment this training corpus with data that we collect from openly-available sources on the web. We discuss preparation methods for our self-collected data in the following.

Preprocessing noisily-labeled speech. Many speech sources available on the web have subtitles or captions available, which can serve as labels. However, captions tend to be noisy—they may be manually-generated and thus contain text that is orthogonal to the audio content, or they may contain the names of speakers or verbal descriptions of non-speech content. In cases where a manually-generated but possibly-unreliable caption is available, we use a heuristic

process to filter out low-quality instances. First, we lower-case and normalize the caption text, removing or replacing, e.g., ambiguous unicode characters, emoji, and punctuation. We then use Whisper large v3 to generate a pseudo-label of the audio content, applying the same text normalization to this pseudo-label as we do the caption. Finally, we compute a normalized Levenshtein distance (between [0.0, 1.0], where 0.0 is identical and 1.0 is orthogonal) between the normalized caption and the pseudo-label, filtering out labels with a distance above a threshold. This allows us to treat the human-generated labels in captions as ground truth without introducing excessive noise. After filtering out noisy labels, we prepare the remaining text by applying standardized punctuation and capitalization.

Preprocessing unlabeled speech. The majority of speech available on the web is unlabeled. In these cases, we leverage the Whisper large v3 model to generate training labels for our lighter-weight Moonshine model. The risk inherent in training one model on another model’s outputs is that the new model learns the old model’s errors. From inspection, we noted that the majority of hallucinated outputs from Whisper large v3 occurred below a predictable value of the average log probability of the output. We thus mitigate the risk of introducing hallucination and other noise in the training set by filtering out instances with an average log probability below this threshold. During this process, we benefited from speed-ups provided by batched inference in the WhisperX implementation (Bain et al., 2023).

Controlling instance duration. We assemble successive speech segments into longer training instances, such that the resulting instance duration is $\in [4, 30]$ seconds with no more than 2 seconds between successive segments. For our manually-captioned data sources, we use timestamped segments provided by a subtitle file (e.g., an .srt file); for our pseudo-labeled audio, we use the timestamps output by Whisper. Upon combining this data with open datasets, the

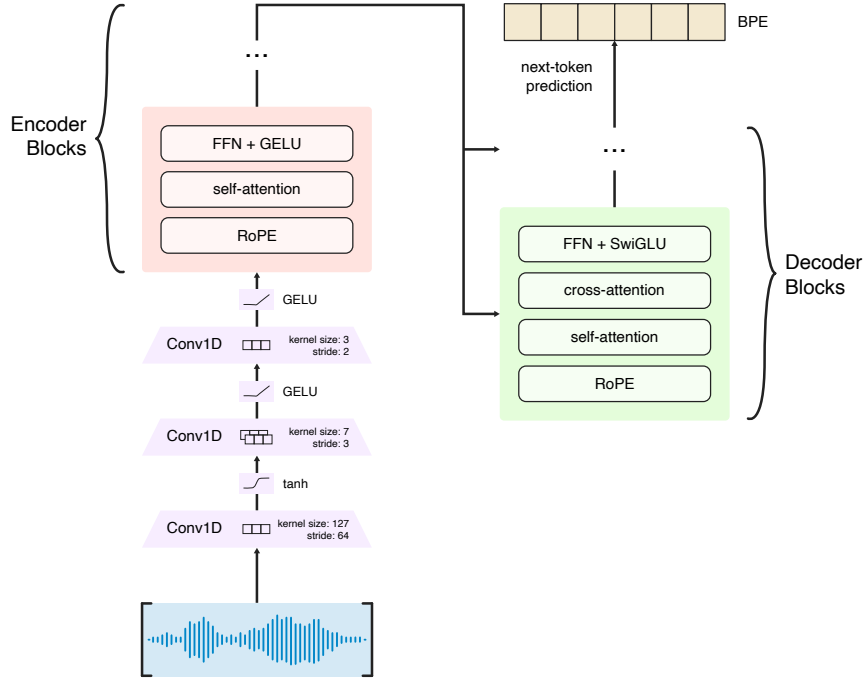


Figure 3. Moonshine’s model architecture.

| Parameter | Moonshine Tiny | Whisper <code>tiny.en</code> | Moonshine Base | Whisper <code>base.en</code> |
|--|----------------|------------------------------|----------------|------------------------------|
| Dimension | 288 | 384 | 416 | 512 |
| Encoder layers | 6 | 4 | 8 | 6 |
| Decoder layers | 6 | 4 | 8 | 6 |
| Attention heads | 8 | 6 | 8 | |
| Encoder FFN activation | GELU | | | |
| Decoder FFN activation | SwiGLU | GELU | SwiGLU | GELU |
| Parameters in Millions | 27.1 | 37.8 | 61.5 | 72.6 |
| FLOPs normalized to Whisper <code>tiny.en</code> | 0.7x | 1.0x | 1.6x | 2.3x |

Table 1. Model shapes and GFLOPs

duration of instances in our aggregate training set obeys a slightly bimodal distribution as depicted in Figure 4.

3.3. Training

We performed training on a 32x H100 GPU cluster, using GPU data parallelism provided by Huggingface’s Accelerate library (Gugger et al., 2022). We used Accelerate’s BF16 mixed precision optimizations. We trained the models for 250K steps with a batch size of 32 per GPU (1024 global batch size). We leveraged the AdamW variation of the schedule-free optimizer (Defazio et al., 2024); gradient norm clipping and the initial learning rate warmed up to $1.4e-3$ after 8192 steps.

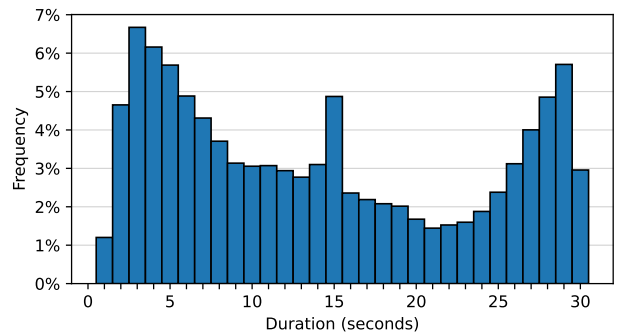


Figure 4. Distribution of training instance durations after combining open and internally-prepared datasets. A slightly bimodal distribution results from our preprocessing procedure, which assembles successive audio segments into instances between 4 and 30 seconds in length.

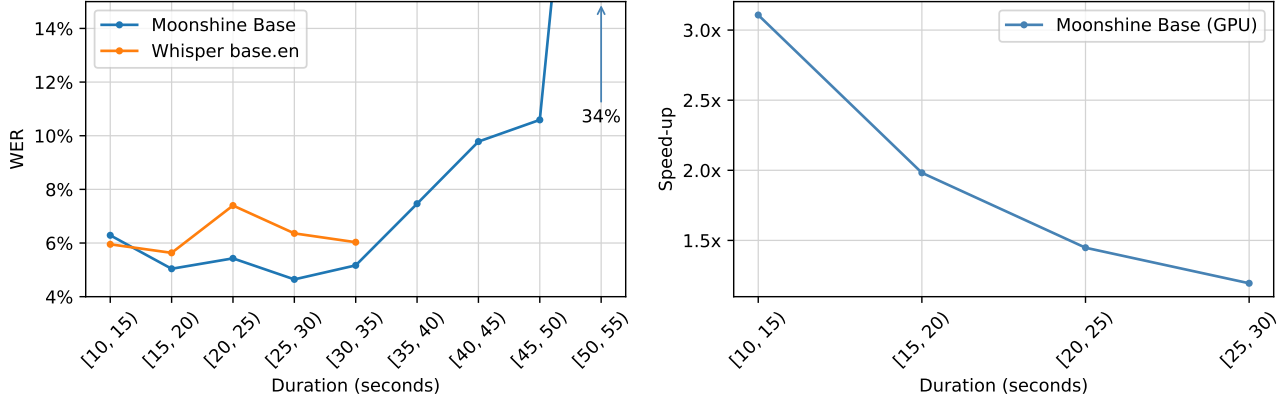


Figure 5. Left: Word Error Rates (WER) across various ranges of input audio duration. Right: Speed-up in decoding time of Moonshine Base over Whisper `base.en` across various ranges of input audio duration.

| | Whisper <code>base.en</code> | Moonshine Base |
|---------------------|---------------------------------|-------------------|
| AMI | 21.13 | 17.79 |
| Earnings22 | 15.09 | 17.65 |
| Gigaspeech | 12.83 | 12.19 |
| LibreSpeech (Clean) | 4.25 | 3.23 |
| LibreSpeech (Other) | 10.35 | 8.18 |
| SPGISpeech | 4.26 | 5.46 |
| TEDLium | 4.87 | 5.22 |
| Voxpopuli | 9.76 | 10.81 |
| Average | 10.32 | 10.07 |

Table 2. WER comparison of base Moonshine vs. Whisper models

| | Whisper <code>tiny.en</code> | Moonshine Tiny |
|---------------------|---------------------------------|-------------------|
| AMI | 24.24 | 22.77 |
| Earnings22 | 19.12 | 21.25 |
| Gigaspeech | 14.08 | 14.41 |
| LibreSpeech (Clean) | 5.66 | 4.52 |
| LibreSpeech (Other) | 15.45 | 11.71 |
| SPGISpeech | 5.93 | 7.7 |
| TEDLium | 5.97 | 5.64 |
| Voxpopuli | 12.00 | 13.27 |
| Average | 12.81 | 12.66 |

Table 3. WER comparison of tiny Moonshine vs. Whisper models

4. Evaluations

Our experiments measure Moonshine against OpenAI’s Whisper models, specifically Whisper `tiny.en` and `base.en`. These two are the most suitable for running on low cost edge devices given their lower memory and compute requirements. We measure the WER on datasets in the OpenASR Leaderboard (Srivastav et al., 2023). We use greedy decoding, with a heuristic limit of 6 output tokens per second of audio to avoid repeated output sequences.

Moonshine Tiny and Base models achieve better average WER compared to their Whisper counterparts (`tiny.en` and `base.en`, respectively), as depicted in Table 2 and Table 3. Earnings22 is the dataset for which Moonshine performs the worst compared to Whisper. 8% of the examples in Earnings22 are less than 1 second long, and the corresponding transcriptions are short utterances such as “So.”, “Yes.”, “Okay.”, etc. Moonshine models tend to produce repeated tokens in the output for such examples, leading to

a WER greater than 100%. Fewer than 0.5% of examples in our training set are shorter than a second. We hypothesize that this weakens generalization to the Earnings22 dataset.

4.1. Accuracy vs. Input Length

Moonshine models demonstrate the ability to transcribe audio clips of varying lengths. However, we hypothesize that shorter inputs may result in higher WER due to the lack of sufficient contextual information. Conversely, for longer clips—particularly those exceeding the sequence lengths observed during training—there is a notable increase in transcription errors due to hallucinations. To evaluate this behavior, we measured the WER across a range of audio segments from the TEDLium dataset, with clip durations varying between 10 and 55 seconds.

To evaluate inference speed across different clip lengths, we compare Moonshine and Whisper on GPU (H100). The results are depicted in Figure 5. The speed-up comparison

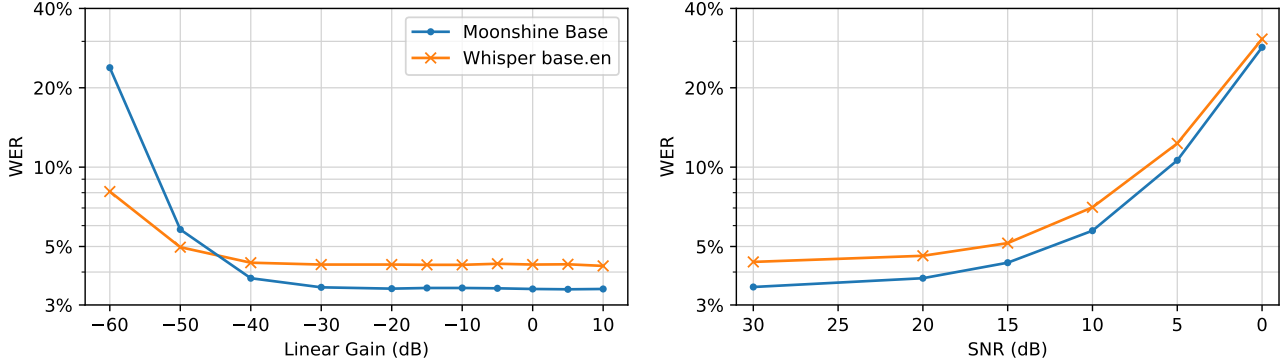


Figure 6. Robustness comparisons on LibriSpeech test-clean. Left: WER as linear gain of test audio increases. Moonshine degrades more at the lowest gains, but maintains superior WER values at -40 dB and beyond. Right: WER as signal-to-noise ratio increases under additive computer fan noise. Moonshine degrades similarly to OpenAI’s Whisper counterpart while maintaining superior WER values.

is limited to clips up to Whisper’s 30-second maximum, as longer clips require multiple Whisper runs, introducing overhead and skewing the results unfairly against Whisper. Moonshine demonstrates superior performance, particularly on shorter sequences, due to its support for variable sequence lengths in the encoder.

4.2. Robustness to Input Speech Signal Level

During this work, we observed that the robustness of Whisper models varies relative to the level of the input speech signal. To measure the impact of input levels (i.e., linear gain) on Moonshine, we tested model robustness as increasing gain is applied to the audio samples in the dataset. As shown in the left portion of Figure 6, Moonshine Base maintains superior WER over most of this range until the input speech signal is very attenuated. At this point (where gain is below -40 dB, i.e., the input audio is very quiet) the WER rises above the Whisper model. For practical applications where high speech attenuation can be avoided in system design, Moonshine Base is robust to varying input levels.

4.3. Robustness to Additive Noise

We tested the noise robustness of the Moonshine Base model by measuring the WER for the fan noise observed in a tablet computer application under load. In user studies, we quantified the signal-to-noise ratio (SNR) for this application in the range [9, 17] dB depending on the speaker. We calculate the level of additive noise corresponding to a given SNR based on the average signal power of individual dataset examples with quiet sections removed. The right portion of Figure 6 shows how the performance degrades as fan noise increases. Whisper’s robustness to noise is known (Radford et al., 2022); Moonshine Base maintains this robustness while providing a superior WER.

5. Discussion & Conclusion

We briefly discuss limitations before concluding the paper in this section.

Architectures and optimizers. As in any research of this kind, many variations of model architecture could be explored. Likewise, recent advancements in optimizers—particularly Shampoo (Gupta et al., 2018) and SOAP (Vyas et al., 2024)—show promise for improving WERs in our architecture. Conducting an ablation study on these model architectures and training methods would significantly enhance the community’s understanding of the models’ limitations. However, due to resource constraints, particularly limited affordability of GPUs, such studies are beyond the scope of this paper. Consequently, we chose our architecture and optimizer based on the authors’ experience and a thorough review of relevant literature.

Generalization to shorter audio segments. The relatively low performance that we observed on the Earnings22 dataset motivates collection of more data representative of this case, and investigation of training techniques to introduce more contextual information in the model. This will likely improve model performance with no changes to architecture.

In this paper we introduced Moonshine, a family of lightweight ASR models optimized for low latency, on-device speech-to-text applications. We outlined our model architecture, data collection and preprocessing procedures, and training. We benchmarked Moonshine against two OpenAI Whisper models—tiny.en and base.en—showing that their Moonshine counterparts provide up to 3x reductions in latency scaled to the duration of input audio. Our work opens the door for new applications of real-time ASR in live transcription, accessibility technologies, and smart devices.

6. Acknowledgements

We would like to acknowledge the developers of the excellent `x-transformers`² library, which formed the basis of the Moonshine training code. We would also like to thank the folks at Lambda Labs for providing reliable server up-time and alacritous support. Finally, we are thankful for the giants, whose shoulders we all stand on.

References

- Ardila, R., Branson, M., Davis, K., Henretty, M., Kohler, M., Meyer, J., Morais, R., Saunders, L., Tyers, F. M., and Weber, G. Common voice: A massively-multilingual speech corpus. In *Proceedings of the 12th Conference on Language Resources and Evaluation (LREC 2020)*, pp. 4211–4215, 2020.
- Bain, M., Huh, J., Han, T., and Zisserman, A. Whisperx: Time-accurate speech transcription of long-form audio. *INTERSPEECH 2023*, 2023.
- Carletta, J., Ashby, S., Bourban, S., Flynn, M., Guillemot, M., Hain, T., Kadlec, J., Karaiskos, V., Kraaij, W., Kronenthal, M., et al. The ami meeting corpus: A pre-announcement. In *International workshop on machine learning for multimodal interaction*, pp. 28–39. Springer, 2005.
- Chen, G., Chai, S., Wang, G., Du, J., Zhang, W.-Q., Weng, C., Su, D., Povey, D., Trmal, J., Zhang, J., et al. Gigaspeech: An evolving, multi-domain asr corpus with 10,000 hours of transcribed audio. *arXiv preprint arXiv:2106.06909*, 2021.
- Defazio, A., Yang, X. A., Mehta, H., Mishchenko, K., Khaled, A., and Cutkosky, A. The road less scheduled, 2024. URL <https://arxiv.org/abs/2405.15682>.
- Galvez, D., Diamos, G., Ciro, J., Cerón, J. F., Achorn, K., Gopi, A., Kanter, D., Lam, M., Mazumder, M., and Reddi, V. J. The people’s speech: A large-scale diverse english speech recognition dataset for commercial usage. *arXiv preprint arXiv:2111.09344*, 2021.
- Gugger, S., Debut, L., Wolf, T., Schmid, P., Mueller, Z., Mangrulkar, S., Sun, M., and Bossan, B. Accelerate: Training and inference at scale made simple, efficient and adaptable. <https://github.com/huggingface/accelerate>, 2022.
- Gupta, V., Koren, T., and Singer, Y. Shampoo: Pre-conditioned stochastic tensor optimization. In *International Conference on Machine Learning*, pp. 1842–1850. PMLR, 2018.
- Panayotov, V., Chen, G., Povey, D., and Khudanpur, S. Librispeech: an asr corpus based on public domain audio books. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pp. 5206–5210. IEEE, 2015.
- Pratap, V., Xu, Q., Sriram, A., Synnaeve, G., and Collobert, R. Mls: A large-scale multilingual dataset for speech research. *ArXiv*, abs/2012.03411, 2020.
- Radford, A., Kim, J. W., Xu, T., Brockman, G., McLeavey, C., and Sutskever, I. Robust speech recognition via large-scale weak supervision, 2022. URL <https://arxiv.org/abs/2212.04356>.
- Sennrich, R., Haddow, B., and Birch, A. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.
- Srivastav, V., Majumdar, S., Koluguri, N., Moumen, A., Gandhi, S., et al. Open automatic speech recognition leaderboard. https://huggingface.co/spaces/hf-audio/open_asr_leaderboard, 2023.
- Su, J., Lu, Y., Pan, S., Murtadha, A., Wen, B., and Liu, Y. Roformer: Enhanced transformer with rotary position embedding, 2023. URL <https://arxiv.org/abs/2104.09864>.
- Touvron, H., Martin, L., Stone, K., Albert, P., and et al., A. A. Llama 2: Open foundation and fine-tuned chat models, 2023. URL <https://arxiv.org/abs/2307.09288>.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.
- Vyas, N., Morwani, D., Zhao, R., Shapira, I., Brandfonbrener, D., Janson, L., and Kakade, S. Soap: Improving and stabilizing shampoo using adam. *arXiv preprint arXiv:2409.11321*, 2024.

²<https://github.com/lucidrains/x-transformers>