WhisperKit: On-device Real-time ASR with Billion-Scale Transformers

Berkin Durmus [1 2] **Arda Okan** [2] **Eduardo Pacheco** [2] **Zach Nagengast** [2] **Atila Orhon** [2]

Abstract

Real-time Automatic Speech Recognition (ASR) is a fundamental building block for many commercial applications of ML, including live captioning, dictation, meeting transcriptions, and medical scribes. Accuracy and latency are the most important factors when companies select a system to deploy. We present WhisperKit, an optimized on-device inference system for real-time ASR that significantly outperforms leading cloud-based systems. We benchmark against server-side systems that deploy a diverse set of models, including a frontier model (OpenAI `gpt-4o-transcribe`), a proprietary model (Deepgram `nova-3`), and an open-source model (Fireworks `large-v3-turbo`).Our results show that WhisperKit matches the lowest latency at 0.46s while achieving the highest accuracy 2.2% WER. The optimizations behind the WhisperKit system are described in detail in this paper.

1. Introduction

Frontier models have been scaled to trillions of parameters to serve virtually all applications across all modalities with a single model, narrowing down deployment options to the cloud due to unprecedented memory requirements. At the same time, task-focused models, whether distilled or trained from scratch, match or exceed the accuracy of frontier models at a fraction of the inference cost for most applications, making on-device deployment of best-in-class models scalable and economical, even for real-time streaming inference. Given this dichotomy, frontier models have become the choice for fast prototyping and time-to-market with a cloud-based inference API while task-focused models with an on-device inference API become the steady-state solution as the industry matures.

For Automatic Speech Recognition (ASR), Whisper Large v3 Turbo (Radford et al., 2022), a 1-billion parameter Encoder-Decoder Transformer model, matches or exceeds many frontier models, such as `gpt-4o-transcribe` (OpenAI, 2025), in ASR accuracy while being sufficiently compact for on-device deployment. WhisperKit is an optimized on-device inference system designed to deploy Whisper models for real-time streaming transcription on Apple devices. [1].

In specific, WhisperKit makes the following contributions:

- We modified Whisper's system architecture such that the *Audio Encoder* model natively supports streaming inference and the *Text Decoder* model yields accurate output text streams even when running on partial audio.
- We reimplemented Whisper for native acceleration on the *Apple Neural Engine* (ANE) to achieve near-peak hardware utilization while retaining the required energy efficiency for on-device deployment.
- We compress Whisper with a new technique that retains the *Word Error Rate* (WER) within 1% of the original uncompressed model while reducing the model file size from 1.6 GB to 0.6 GB.

2. Method

2.1. Architecture

Real-time streaming transcription is a challenging ASR task with major commercial applications such as live captioning, dictation, meeting transcriptions, medical scribes, etc. The challenges are twofold: Achieving high accuracy with partial audio context due to the streaming nature of the input while maintaining low latency in real-time processing, where delays compound over time. The latency challenge is exacerbated for model architectures such as Whisper that do not natively support streaming inference. Whisper consists

[1]University of California, Los Angeles, USA [2]Argmax Inc., Palo Alto, USA. Correspondence to: Atila Orhon <a@argmaxinc.com>.

Proceedings of the 42nd International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025. Copyright 2025 by the author(s).

[1]`Code available at:https://github.com/argmaxinc/WhisperKit`

of an *Audio Encoder* and a *Text Decoder*. Both components introduce distinct challenges.

2.1.1. Audio Encoder

Whisper Audio Encoder can only process 30-second audio chunks. The naive streaming implementation with a 1-second target latency involves zero-padding the audio input to 30 seconds and running an audio encoder forward pass at most every 1 second as the audio buffer gets updated. For 1 minute of streaming inference, this implementation leads to at least 60 audio encoder forward passes, the equivalent of up to 30 minutes of processed audio, compared to offline inference which requires only a single forward pass.

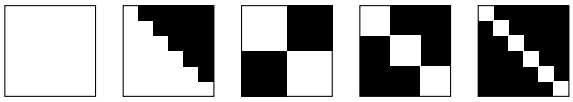
Moonshine (Jeffries et al., 2024) is a rearchitected version of Whisper that removes the need to pad the input to 30-second buffers. Although Moonshine significantly improves the streaming efficiency, it suffers accuracy loss for long-form transcription. Furthermore, the published models were not scaled beyond the `tiny` and `base` variants leading to an accuracy gap with larger state-of-the-art models.

(Liu et al., 2020) fine-tunes their audio encoder Transformer model with a lower-triangular attention mask such that its self-attention layers become causal. This technique removes the "look-ahead" conditioning caused by the original bidirectional self-attention and enables key-value (KV) caching for incremental encoding of input audio streams.

We apply self-distillation to Whisper Large V3 Turbo using the Common Voice 17 (Ardila et al., 2020) training split following the core ideas from (Liu et al., 2020). In our experiments, we found that the lower-triangular causal attention mask is too strict and led to an accuracy drop. Instead, we relax the attention mask to be block-diagonal where the blocks represent 15-second audio chunks, depicted as `d750` in Figure 1(c).

After self-distillation of the audio encoder, the text decoder still needs to attend over 30-second audio buffers. However, streaming inference severely underutilizes this buffer length and most of the input buffer needs to be zero-padded, leading to unnecessary compute and latency. Block-diagonal masking enables *silence caching* where the audio encoder's output for an entire 15-second zero-padded block of audio can be computed at compile time and reused instead of running inference on zero-padded inputs.

English Accuracy. Table 1 shows the impact of self-distillation using these self-attention masks on the audio encoder latency and WER for short-form and long-form and multilingual transcription. `d750` retains WER within 1% of the original model while reducing latency by 65% (602 ms →218 ms). Shorter blocks such as `d250` and `d500` yield an undesirable trade-off between latency and accuracy.



(a) original (b) c250 (c) d750 (d) d500 (e) d250

Figure 1. **Block-diagonal (d*) and block-causal (c*) masks for Whisper Audio Encoder 1500x1500 (30 seconds) Self-attention Matrix.**

Mask	TFLOPs	Audio Encoder Latency (ms)	librispeech-test.clean (WER)	earnings22[2] (WER)
original	2.27	612	1.93	11.55
c250	2.34	$117 * 6 = 702$	2.32 (+0.39)	12.89 (+1.34)
d750	1.04	218	**2.25 (+0.32)**	**12.85 (+1.30)**
d500	0.68	125	3.39 (+1.46)	13.86 (+2.31)
d250	**0.33**	**50**	25.58 (+23.65)	39.75 (+28.20)

Table 1. **Latency and WER benchmarks for Whisper Audio Encoders self-distilled with various attention masks.** Latency is measured on a MacBook Pro with M3 Max chip on the Neural Engine. `c250` latency is multipled by 6 because block-causal masking does not allow for *silence caching* and requires 6 forward passes of 5-second audio segments to fill the 30-second audio buffer.

On the other hand, block-causal masking, depicted as `c250` in Figure 1(b), retains the same accuracy as `d750` for 3x shorter audio blocks. However, unlike block-diagonal, block-causal conditioning requires key-value caching for previous blocks in the same 30-second window and does not allow *silence caching* as future blocks within the same 30-second window depend on earlier blocks that change as input audio streams.

Multilingual Accuracy. Figure 2 shows the WER/CER of the `d750` variant on a subset of the Common Voice 17 (Ardila et al., 2020) test split. Whisper Large v3 Turbo (original) was initially trained to recognize and transcribe in 100 languages. However, the model does not demonstrate useful quality on many of these languages. In order to focus on a useful subset, we focus our benchmarks on the 20 languages where the original model achieves the lowest WER/CER, plus Japanese and Chinese.

The `d750` variant was obtained by fine-tuning the original model while applying the self-attention mask from Figure 1(c) on a subset of Common Voice 17 training split covering the following 5 languages: English, German, Japanese, Chinese, and French. We hypothesized that the fine-tuned model would achieve even lower WER than the original model for the languages where it was fine-tuned because this strategy allows the model to forget the other languages and recycle its limited parameter count towards fewer languages.

Figure 2 shows that the model either retains WER within 1% (English and Japanese) or improves WER (French and Ger-

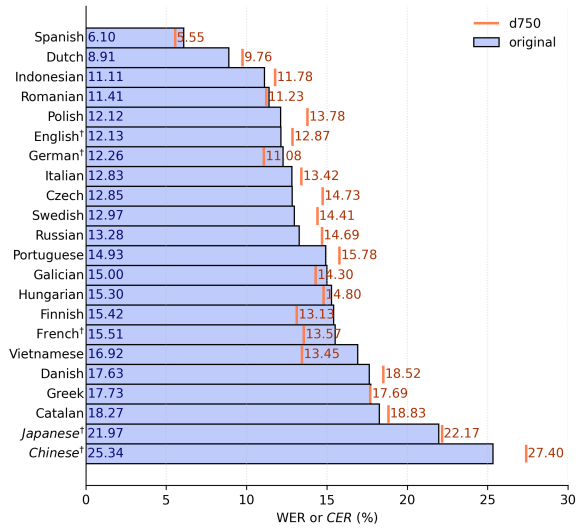


Figure 2. **Multilingual WER (Word Error Rate) and CER (Character Error Rate) before (original) and after (`d750`) self-distillation of Whisper Large v3 Turbo.** Metrics are reported for the 20 languages with the lowest WER/CER in the original model, plus Japanese and Chinese, on a subset of the Common Voice 17 test set. [4]. Languages marked with † were used during self-distillation.

man) when compared to the original model for languages where it was fine-tuned. The exception was Chinese which regressed by 2% WER, presumably due to small training dataset size.

These results demonstrate that fine-tuning Whisper Large v3 Turbo using this strategy is a scalable approach to improving multilingual accuracy while optimizing inference efficiency.

2.1.2. Text Decoder

Unlike the Audio Encoder, the Whisper Text Decoder is capable of streaming individual output tokens while attending over a fixed input audio buffer. However, the naive streaming implementation for the text decoder is also problematic. The output transcript buffer is filled with temporary text tokens, and the buffer is repeatedly flushed and refilled after each audio encoder forward pass until the text decoder can confirm a full transcript buffer by predicting the `<endoftranscript>` special token. Until this event occurs, all text token predictions are subject to change and the audio cursor can not be moved forward, leading to compounding latency. We call these temporary results *hypothesis text*.

(Liu et al., 2020) proposed the `LocalAgreement` streaming policy that frequently confirms the hypothesis text by searching for the longest common prefix across two consecutive *hypothesis text* buffers and moves the audio cursor to the end of the last token in this common prefix. WhisperKit and concurrent work (Macháček et al., 2023) applied the `LocalAgreement` policy to Whisper, leading to dual output text streams:

- *Confirmed text* stream can be leveraged in the user experience to build trust in stable and accurate results.
- *Hypothesis text* preserves low latency and responsiveness with occasional corrections as more audio context becomes available.

Hypothesis text helps real-time transcription systems achieve sub-second latency while retaining the flexibility for retroactive corrections to match offline transcription accuracy. However, if the number of corrections is high, it may also hurt the user experience and reduce trust in the accuracy of these systems.

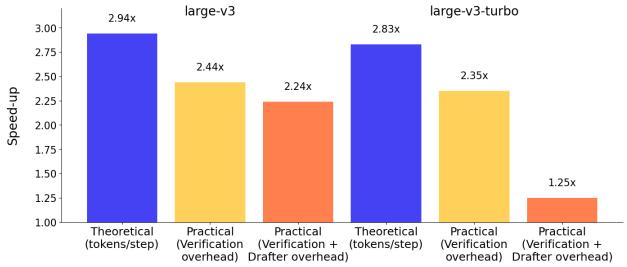


Figure 3. **Speedup factors for Whisper Text Decoder with Speculative Decoding**. Theoretical speedup (tokens/step) is the token acceptance rate of the verification step and signifies the speedup for an idealized hardware with infinite compute and limited memory bandwidth. The practical slowdowns are due to the *verification overhead* (processing multiple tokens in parallel) and *drafter overhead* (forward pass latency of the drafter model).

Speculative Decoding. (Cheng et al., 2024) introduced *Recurrent Drafter*, a lossless speculative decoding algorithm that uses a lightweight RNN drafter to propose multiple future tokens conditioned on the original model's hidden states, which are then verified in parallel by the original model.

We trained a draft model that achieves a 2.94x and 2.83x theoretical speedup (accepted tokens per step) for Whisper Large v3 and Large v3 Turbo respectively. This theoretical speedup is mostly retained even after accounting for the *verification overhead* (2.94x →2.41x and 2.83x →2.35x). The verification overhead is the slowdown of running the text decoder on 16 tokens (beam width multiplied by beam length, $4 \times 4 = 16$) in parallel instead of 1 token per forward pass.

(Cheng et al., 2024) observed that Apple Silicon GPUs achieved significantly lower practical speedups than Nvidia GPUs with identical drafter models due to increased parallel verification overhead which pushes the workload to

be compute-bound, especially for large models (7-billion parameters or higher). We show that WhisperKit's Apple Neural Engine implementation unlocks a much higher compute throughput than was available in (Cheng et al., 2024)'s Apple Silicon GPU experiments, minimizing the *verification overhead* to levels comparable to Nvidia GPUs.

However, we choose not to employ speculative decoding in our final WhisperKit implementation for Whisper Large v3 Turbo due to the *drafter overhead*, the slowdown that is attributed to running the drafter model 4 times (beam length) in addition to running the original text decoder once. For Whisper Large v3, this overhead is minimal due to the original model's high latency and the practical speedup ends up being 2.24x. For Whisper Large v3 Turbo, this overhead is significant due to the original model's low latency and the practical speedup ends up being 1.25x.

2.2. On-device Constraints

There are two challenges to overcome before billion-scale Transformers can be ubiquitously deployed on edge devices: Energy and peak memory consumption.

2.2.1. Energy

Energy consumption directly impacts battery life and thermal sustainability of using an edge device and on-device inference must be extremely energy efficient in order to avoid disruptive patterns such as degraded battery life and too-hot-to-hold devices that throttle themselves to a halt in order to cool down.

In 2018, Apple introduced the Neural Engine (ANE) with the A12 chip for iPhone XS, an inference accelerator that optimized *performance per watt* under 10 watts. In 2020, the Neural Engine was introduced to Mac and iPad with the M1 chip (Orhon et al., 2022). In 2023, all devices supported by iOS 17 had a Neural Engine. In 2025, iOS 17 is the oldest supported operating system for most commercial applications, making the Neural Engine ubiquitous for all Apple users.

In the Android market, Qualcomm, Mediatek, and others have driven a similar hardware transformation by introducing the Neural Processing Unit (NPU) in mobile chipsets such as Snapdragon 8 and Dimensity. In 2024, the Qualcomm Snapdragon X Elite chip was adopted by Microsoft in Copilot PCs, kickstarting the adoption of NPUs by Windows.

However, ANE and NPUs in general remain underutilized due to lack of public documentation and tooling. They have been primarily utilized by the device makers at the firmware level to power inference for operating system features such as computational photography (Orhon et al., 2021) and speech recognition (Xu et al., 2024). In 2022, Apple published "Deploying Transformers on the Apple Neural Engine" and open-sourced `ane-transformers` (Orhon et al., 2022) to demonstrate how a 0.1 billion parameter Transformer Encoder model could be deployed on the iPhone while being faster than server-side inference (Orhon et al., 2022). In 2024, Apple unveiled Apple Intelligence (Apple, 2024a) which is a 3-billion parameter Transformer language model running on the Neural Engine. The performance achieved by Apple Intelligence is not easily accessible to any third-party model due to lack of documentation. However, the necessary tooling is available with Core ML (Apple, 2017).

WhisperKit builds on the `ane-transformers` reference implementation of the Transformer architecture and applies additional optimizations to reach Apple Intelligence-level performance for a third-party Transformer model using the Core ML framework. In specific, *Stateful Models* (Apple, 2024b) feature is leveraged to ensure that the key-value cache of the Whisper Text Decoder is read and updated in-place, persisting across forward passes. This leads to a 45% latency reduction (8.4 ms →4.6 ms on M3 ANE) for the Whisper Large v3 Turbo Text Decoder forward pass compared to our previous implementation that passed the key-value cache as input tensors to the model.

Most importantly, the energy consumption for one forward pass of the same model is reduced by 75% (1.5W →0.3W), mitigating the battery degradation and heating problem.

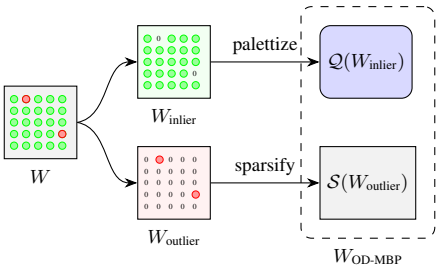


Figure 4. **Compile time with OD-MBP.** The original weight tensor $\mathbf{W}$ is decomposed into a dense inlier block $\mathbf{W}_{\text{inlier}}$ (green) and a sparse outlier block $\mathbf{W}_{\text{outlier}}$ (red). Outliers are defined as weight values that are more than 3 standard deviations away from the mean. Inliers are *palettized* and stored as a low-bit lookup table (LUT) as $Q(\mathbf{W}_{\text{inlier}})$, while outliers are kept in float16 precision and stored in a bit-packed sparse representation as $S(\mathbf{W}_{\text{outlier}})$.

2.2.2. Memory

The model size introduces different bottlenecks throughout the lifecycle of on-device deployment. First, the model file must be distributed to the end-user device over the air (OTA) *after* the app is installed to keep the app size small and decouple model updates from software updates. The only increase in the app size must be tied to the on-device inference SDK which is generally less than 5 MB. Once

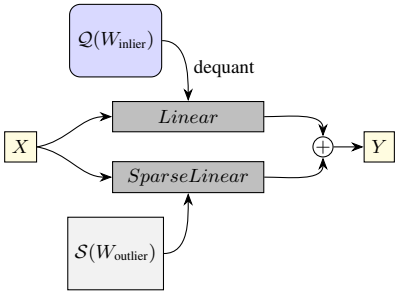


Figure 5. **Inference with OD-MBP.** The float16 input activation X is dispatched to two parallel operators: *(i) Dense path.* A `Linear` layer retrieves the low-bit palettized inlier weights $Q(W_{\text{inlier}})$, dequantizes them on the fly, and performs a dense matrix–vector product in float16 precision. *(ii) Sparse path.* A `SparseLinear` layer retrieves the float16 precision sparse outlier weights $S(W_{\text{outlier}})$ and performs a sparse matrix–vector product in float16 precision.

distributed, the model files take up storage space even when not being used, so models larger than 2 GB are not end-user friendly.

Peak memory consumption during active use is dominated by the model weights and models optimized for on-device deployment must require less than 2 GB of RAM in order to retain virtually universal device support. This size roughly corresponds to a 1-billion parameter model in float16 precision or 4-billion parameter model in 4-bit precision after weight compression. Whisper Large v3 Turbo requires 1.6 GB in float16 precision. Naive compression of Whisper models lead to increased Word Error Rate (WER) along with hallucination and repetitive patterns in the predicted transcription.

WhisperKit leverages a new compression technique called *Outlier-Decomposed Mixed-Bit Palletization* (OD-MBP) to retain WER within 1% of the original uncompressed model while staying below 1 GB in size. *Mixed-Bit Palettization* (MBP) was first proposed in (Cuenca & Orhon, 2023) to compress Stable Diffusion XL with the natively accelerated compression format called *palettization* (Apple, 2023). Subsequently, MBP was adopted and improved upon by others such as (Sui et al., 2024) to achieve low-bit high-accuracy weight compression for edge deployment.

Outlier Decomposition (OD) was first proposed in (Dettmers et al., 2022) where the forward pass of a linear layer is decomposed into a low-bit precision *inlier* branch and a float16 precision *outlier* branch to minimize the errors caused by low-bit compression. Outliers are determined based on the channel-wise magnitude statistics. By definition, they represent less than 1% of the data. However, this technique does not accelerate inference because the latency gains from low-bit compressed weights requiring less memory bandwidth are offset by the additional compute from the float16 branch. Furthermore, this technique is restricted to feature-wise structured decomposition and does not admit unstructured outlier decomposition.

OD-MBP builds on top of OD and MBP by decomposing the weight tensors as shown in Figure 4 such that the low-bit precision inlier branch is implemented using the natively-accelerated palettization format and the float16 precision outlier branch is implemented using the natively-accelerated sparse weight format (Apple, 2023) that is capable of accelerating almost fully unstructured sparsity (Apple, 2023). Figure 5 shows changes to the forward pass of a `Linear` layer when its weights are compressed with OD-MBP.

When applied to Whisper Large v3 Turbo, OD-MBP preserves the WER of the original model to within 1% across various short- and long-form datasets as shown in Table Table 2.

Compression State	Mask	Size [GB]	librispeech-test.clean (WER)	earnings22-12hours (WER)	CommonVoice17-en (WER)
FP16	original	1.6	1.93	11.55	12.13
OD-MBP	original	0.6	1.96 (+0.03)	12.35 (+0.80)	13.03 (+0.90)
FP16	d750	1.6	2.25 (+0.32)	12.85 (+1.30)	12.87 (+0.74)
OD-MBP	d750	0.6	2.30 (+0.37)	12.72 (+1.17)	14.21 (+2.08)

Table 2. **Impact of OD-MBP on WER**. OD-MBP, even when combined with the `d750` optimization, retains WER within 1% of the original uncompressed model while compressing Whisper Large v3 Turbo from 1.6 GB to 0.6 GB

3. Results

In this section, WhisperKit is benchmarked to diverse and competitive cloud-based inference APIs for real-time transcription.OpenAI `gpt-4o-transcribe` (OpenAI, 2025) sets the frontier model baseline. This particular version of `gpt-4o` is fine-tuned for transcription and sets an even stronger baseline than the base `gpt-4o`. Deepgram `nova-3` (Deepgram, 2025b) (Deepgram, 2025b) sets the baseline for proprietary ASR-focused models. Fireworks `large-v3-turbo` (Fireworks, 2025b) sets the baseline for ASR-focused models using the same open-source model as WhisperKit (Whisper Large v3 Turbo), serving it on the cloud instead of on device.

3.1. Latency

Real-time transcription latency [5] is measured as the difference between the *audio cursor* and the *transcript cursor* following Deepgram's methodology (Deepgram, 2025a) with one improvement: In order to remove the impact of predicted word-level timestamps when setting the *transcript cursor*, we run our benchmarks on the TIMIT dataset (Garofolo, John S. et al., 1993) and adopt its ground-truth word-level timestamps for this purpose.

This improvement was also necessary because Fireworks (Fireworks, 2025b) and OpenAI (OpenAI, 2025) APIs do

[5] SDBench is used for measurements. Code available at: `https://github.com/argmaxinc/SDBench`

not return predicted word-level timestamps. We verified that our latency results in Figure 6 for Deepgram and Fireworks approximately match their offically reported API latency in (Fireworks, 2025a) and (Deepgram, 2025a) despite our metric improvement.

Figure 6 shows the per-word latency for the *hypothesis text* stream. WhisperKit and Fireworks are the fastest, achieving a roughly equal mean latency of 0.45 seconds. Deepgram achieves 0.83 seconds and becomes the third fastest system. Note that OpenAI does not support *hypothesis text* streams so the results represent the latency of the *confirmed text* stream, which is expectedly much slower than all other systems. Figure 7 shows the per-word latency for the *confirmed text* stream. All systems achieve a similar latency of 1.7 seconds. Fireworks is not shown on this figure because their API does not confirm the transcription in contiguous segments.

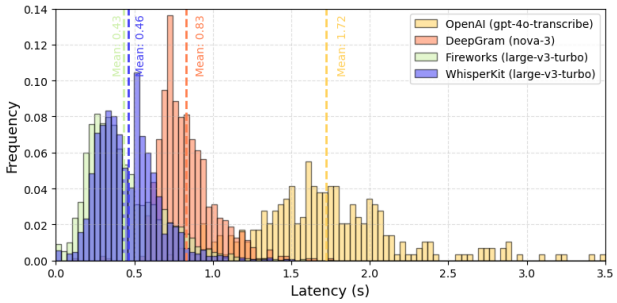


Figure 6. **Per-word Latency Histogram for Hypothesis Text.** Hypothesis text is also referred to as *Interim Result* in Deepgram's documentation (Deepgram, 2025a). Vertical dashed lines indicate the mean latency for each system.

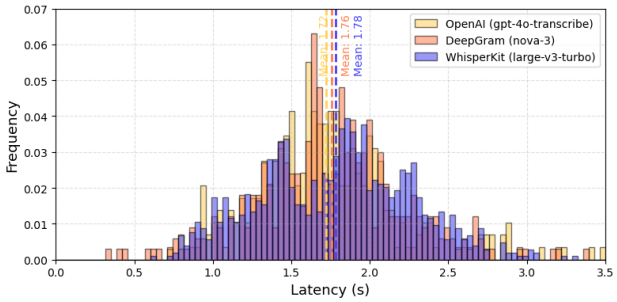


Figure 7. **Per-word Latency Histogram for Confirmed Text.** When *hypothesis text* results are not considered, all benchmarked systems achieve a similarly high latency of 1.7 seconds. Fireworks API does not support partial resul confirmations for any result and is excluded from this plot.

3.2. Accuracy

Evaluating the accuracy of real-time transcription systems is challenging due to the presence of retrospective corrections throughout real-time streaming inference. For comprehensive evaluation with and without retrospective corrections, we designed the following evaluation framework:

- *Confirmed text* accuracy is evaluated once the streaming session (per file) is complete. We leverage the standard Word Error Rate (WER) metric for evaluating this text stream (lower is better). Systems are allowed to correct past mistakes while a result is still marked as *hypothesis text*. Systems are not allowed to retrospectively change confirmed text.

- *Hypothesis text* accuracy is measured by the number of corrections made by the system before confirmation. If number of corrections is zero, then *hypothesis text* accuracy achieves its upper bound and matches the accuracy of *confirmed text*. If the number of corrections is too high, then the reduced latency afforded by *hypothesis text* is not useful and the user experience is degraded due to unstable results.

Figure 8 shows that OpenAI's *confirmed text* accuracy is lowest (highest WER). WhisperKit and Deepgram achieve the highest accuracy (lowest WER) of 2% WER while Fireworks trails behind at 4.72% WER. Notably, OpenAI demonstrates zero corrections because the API does not support *hypothesis text*. On the other hand, Fireworks issues an order of magnitude more corrections than WhisperKit and Deepgram. Despite tying with WhisperKit as the lowest latency system in Figure 6, Fireworks' extremely high number of corrections makes it undesirable for real-time applications. High number of corrections may be associated with small and less accurate models predicting ahead of large and more accurate models. WhisperKit latency is measured on a MacBook Pro with M3 Max chip on the Neural Engine.

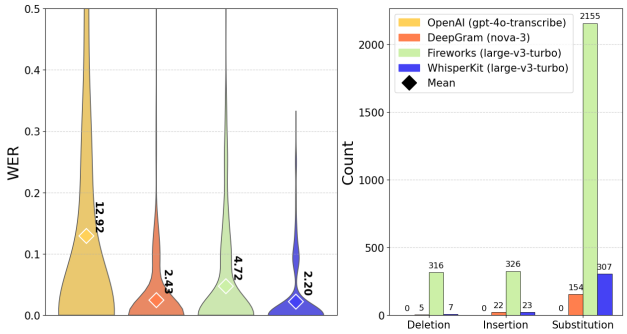


Figure 8. **Streaming Real-time Transcription Accuracy.** On the left, *confirmed text* accuracy is shown as WER plots (lower is better). On the right, *hypothesis text* accuracy is shown as the total number of corrections (lower is better) stratified across deletions, substitutions, and insertions.

References

Apple, I. Core ml, 2017. URL https://developer.apple.com/documentation/coreml.

Apple, I. Use core ml tools for machine learning model compression, 2023. URL https://developer.apple.com/videos/play/wwdc2023/10047/.

Apple, I. Apple intelligence foundation language models, 2024a. URL https://arxiv.org/abs/2407.21075.

Apple, I. Stateful models, 2024b. URL https://apple.github.io/coremltools/docs-guides/source/stateful-models.html.

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.

Cheng, Y., Zhang, A., Zhang, X., Wang, C., and Wang, Y. Recurrent drafter for fast speculative decoding in large language models, 2024. URL https://arxiv.org/abs/2403.09919.

Cuenca, P. and Orhon, A. Stable diffusion xl on mac with advanced core ml quantization, 2023. URL https://huggingface.co/blog/stable-diffusion-xl-coreml#.

Deepgram. Deepgram latency measurement, 2025a. URL https://developers.deepgram.com/docs/measuring-streaming-latency.

Deepgram. Nova 3, 2025b. URL https://deepgram.com/learn/introducing-nova-3-speech-to-text-api.

Dettmers, T., Lewis, M., Belkada, Y., and Zettlemoyer, L. Llm.int8(): 8-bit matrix multiplication for transformers at scale, 2022. URL https://arxiv.org/abs/2208.07339.

Fireworks. Fireworks streaming latency, 2025a. URL https://fireworks.ai/blog/streaming-audio-launch.

Fireworks. Whisper-large-v3-turbo, 2025b. URL https://fireworks.ai/models/fireworks/whisper-v3-turbo.

Garofolo, John S., Lamel, Lori F., Fisher, William M., Fiscus, Jonathan G., Pallett, David S., and Dahlgren, Nancy L. Timit acoustic-phonetic continuous speech corpus. https://catalog.ldc.upenn.edu/LDC93S1, 1993. Linguistic Data Consortium, LDC93S1.

Jeffries, N., King, E., Kudlur, M., Nicholson, G., Wang, J., and Warden, P. Moonshine: Speech recognition for live transcription and voice commands, 2024. URL https://arxiv.org/abs/2410.15608.

Liu, D., Spanakis, G., and Niehues, J. Low-latency sequence-to-sequence speech recognition and translation by partial hypothesis selection, 2020. URL https://arxiv.org/abs/2005.11185.

Macháček, D., Dabre, R., and Bojar, O. Turning whisper into real-time transcription system, 2023. URL https://arxiv.org/abs/2307.14743.

OpenAI. Gpt-4o transcribe, 2025. URL https://platform.openai.com/docs/models/gpt-4o-transcribe.

Orhon, A., Joergensen, M., Lillethorup, M., Vestergaard, J., and Jagadeesh, V. Deploying transformers on the apple neural engine, 2021. URL https://machinelearning.apple.com/research/panoptic-segmentation.

Orhon, A., Wadhwa, A., Kim, Y., Rossi, F., and Jagadeesh, V. Apple neural engine transformers, 2022. URL https://machinelearning.apple.com/research/neural-engine-transformers.

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.

Rio, M. D., Ha, P., McNamara, Q., Miller, C., and Chandra, S. "earnings-22: A practical benchmark for accents in the wild", 2022.

Sui, Y., Li, Y., Kag, A., Idelbayev, Y., Cao, J., Hu, J., Sagar, D., Yuan, B., Tulyakov, S., and Ren, J. Bitsfusion: 1.99 bits weight quantization of diffusion model, 2024. URL https://arxiv.org/abs/2406.04333.

Xu, M., Jin, A., Wang, S., Su, M., Ng, T., Mason, H., Han, M., Lei, Z., Deng, Y., Huang, Z., and Krishnamoorthy, M. Conformer-based speech recognition on extreme edge-computing devices. In *NAACL*, 2024. URL https://arxiv.org/abs/2312.10359.