Sink or SWIM: Tackling Real-Time ASR at Scale

Federico Bruzzone, Walter Cazzola, Matteo Brancaleoni and Dario Pellegrino

***Abstract*—Real-time automatic speech recognition systems are increasingly integrated into interactive applications, from voice assistants to live transcription services. However, scaling these systems to support multiple concurrent clients while maintaining low latency and high accuracy remains a major challenge. In this work, we present `SWIM`, a novel real-time ASR system built on top of OpenAI's `Whisper` model that enables multi-stream inference for scalable, multilingual transcription. `SWIM` supports multiple concurrent audio streams without modifying the underlying model. It introduces a buffer merging strategy that maintains transcription fidelity while ensuring efficient resource usage. We evaluate `SWIM` in multi-client settings—scaling up to 20 concurrent users—and show that it delivers accurate real-time transcriptions in English, Italian, and Spanish, while maintaining low latency and high throughput. While `Whisper-Streaming` achieves a word error rate of approximately 8.2% with an average delay of approximately 3.4 s in a single-client, English-only setting, `SWIM` extends this capability to multilingual, multi-client environments. It maintains comparable accuracy with significantly lower delay—around 2.4 s with 5 clients—and continues to scale effectively up to 20 concurrent clients without degrading transcription quality and increasing overall throughput. Our approach advances scalable ASR by improving robustness and efficiency in dynamic, multi-user environments.**

***Index Terms*—Automatic Speech Recognition, Whisper, Multi-client ASR Systems, Real-time ASR Systems.**

I. Introduction

AUTOMATIC speech recognition (ASR) systems[1] are ubiquitous in everyday applications [1], ranging from voice assistants [2, 3] to transcription services [4, 5]. Over the past decade, ASR has advanced significantly through deep learning [6, 7]. The field has shifted from traditional *cascaded* architectures [8, 9], which separate the recognition pipeline into distinct stages, to *end-to-end* models [10], where a single neural network jointly learns the entire task [11, 12]. `Whisper` is a transformer-based end-to-end ASR model [13]. It is a well-balanced end-to-end solution in the ASR landscape, offering a trade-off between computational cost, speed, and transcription accuracy, as demonstrated by Radford et al. [13].

ASR Systems and Their Challenges. Contemporary ASR systems are generally characterized along two dimensions: *single-client* systems [14, 15], which serve a single user at a time, and *multi-client* systems, which concurrently serve multiple users [16, 17]. These architectures are further classified based on deployment, into *local* systems, which operate on the user's device, and *remote* systems, which are hosted on servers and accessed via network connections. Recent trends in ASR research have increasingly focused on real-time applications, where both low latency and high throughput are critical [18, 19, 20]. In these settings, the system must generate responses within a few seconds to meet users' expectations for immediate feedback. Although local ASR systems offer enhanced privacy [21, 22], their performance is inherently constrained by the limited computational power and memory of user devices [23], as well as their susceptibility to adverse environmental conditions. For example, in audio-only communication channels such as *voice over internet protocol* (VoIP) and telephone calls, audio signals are typically routed to centralized remote ASR systems that benefit from greater computational resources and cloud-based services [24, 25].

Single- vs. Multi-Client Systems. Single-client systems are not without limitations [26]. E.g., `Whisper-Streaming` [27] and `Whispy` [28] lack model-level parallelization, limiting their scalability in multi-audio streams scenarios. Conversely, remote ASR systems can leverage greater computational resources and memory, enabling efficient scaling to support multiple clients concurrently [29]. Such systems benefit from optimized resources, lower latency via parallelism, and consistent performance across diverse users [30, 31]. Still, scaling real-time ASR to handle many concurrent clients remains challenging due to the computational demands of low-latency, high-accuracy transcription—critical for applications like voice assistants [32], live captioning [33], and safety-critical domains as healthcare [34] and legal proceedings [35].

Contributions. We introduce **`SWIM`** (**S**erve **W**hisper **I**n **M**ulti-client), a novel real-time ASR system built on top of the `Whisper` model, designed to serve multiple clients simultaneously with low latency and high throughput. **`SWIM`** prioritizes scalability, efficiently handling multilingual audio streams under varying workloads. Unlike `Whisper-Streaming` [27] and `Whispy` [28], **`SWIM`** enables multi-stream inference, maintaining stable performance under high load. Our evaluation shows that a single `Whisper` instance running **`SWIM`** can transcribe multiple multilingual audio streams in parallel with real-time accuracy. We evaluate up to 20 concurrent users and measure: (i) real-time playback delay, (ii) transcription quality via *word error rate* and accuracy, and (iii) performance on multilingual inputs (English, Italian, and Spanish). **`SWIM`** tackles scalability challenges in real-time ASR, advancing the state of the art in multi-client speech recognition for dynamic environments.

Our main contributions are:

1) a scalable real-time ASR system supporting model-level parallelization for multilingual concurrent transcription;

F. Bruzzone is with the Computer Science Department, Università degli Studi di Milano, Italy. E-mail: federico.bruzzone@unimi.it.

W. Cazzola (corresponding author) is with the Computer Science Department, Università degli Studi di Milano, Italy. E-mail: cazzola@di.unimi.it.

M. Brancaleoni is with the Computer Science Division, VoiSmart, Milan, Italy. E-mail: matteo.brancaleoni@voismart.it.

D. Pellegrino is with the Computer Science Division, VoiSmart, Milan, Italy. E-mail: dario.pellegrino@voismart.it.

[1]We use the term *system* to refer to a software system, typically in a client-server setup, where the client is the user and the server hosts the ASR model.

2) a buffer-merging strategy that preserves accuracy without added latency; and
3) a demonstration that unmodified `Whisper` can handle multiple parallel audio streams from diverse sources.

The paper is organized as follows. Sect. II provides background. Sect. III reviews real-time ASR systems, focusing on `Whisper` and its limitations. Sects. IV and V describes and evaluates **`SWIM`** respectively. Sect. VII concludes the paper.

II. Background

We provide background on speech-to-text ASR models, the `Whisper` Model, and the `Whisper-Streaming` system.

Speech-to-Text ASR Models. ASR converts spoken language into written text using machine learning. Earlier ASR systems relied on hidden Markov models [36, 9] and Gaussian mixture models [8]. ASR models range from high-performance systems for offline processing [11, 37] to *lightweight* real-time versions for limited hardware [38, 39]. These differences drive key architectural and optimization choices. Traditional ASR models relied on *cascaded* pipelines with separately trained components [8, 9]. In contrast, *end-to-end* models integrate these stages into a single jointly trained network [10, 11, 12], simplifying the pipeline and often improving performance, especially with large datasets. *Transformers* [11, 40] replace recurrent and convolutional networks with self-attention, enabling high parallelization and state-of-the-art ASR [11, 41, 42]. ASR models are trained on large datasets such as LibriSpeech (including Long and Multilingual variants) [43, 44, 45] and Fleurs [46], which provide thousands of hours of transcribed speech. Performance is evaluated using *word*, *character*, and *sentence error rates* (WER, CER, SER) [36, 47, 48, 49], computed via normalized Levenshtein distance [50] between predicted and reference texts. For subtitles, captions, and live transcription [51, 52], ASR systems segment audio into temporal chunks and apply *timestamp-aligning*[2] [53, 54] to synchronize transcriptions with the audio.

Whisper Model. `Whisper` [13] is a transformer-based ASR model trained on large multilingual and multitask datasets for multilingual speech recognition. It processes raw audio using a sequence-to-sequence transformer with an *encoder* that produces a dense representation and a *decoder* that generates text tokens via cross-attention. Beyond speech-to-text transcription, `Whisper` also performs well on speech translation, language identification and voice activity detection. At inference time, start and end timestamps can be provided to define audio segments, bypassing voice activity detection by assuming speech within those intervals. `Whisper` is available in multiple sizes (e.g., `tiny`, `small`, `medium`, `large-v3`). Recently, a pruned and fine-tuned `turbo` variant of `large-v3` was introduced, providing a 5–8$\times$ speedup with minimal accuracy loss.

III. State-of-the-Art

Real-time Whisper. `Faster-whisper`[3] is a `Whisper` reimplementation built on `CTranslate2`.[4] and is widely used in real-time ASR systems such as `Whisper-Streaming` [27] and `Whispy` [28]. `Whisper-Streaming` supports real-time transcription and translation using incremental buffering and local agreement to reduce latency and increase throughput [27]. `Whispy` enables real-time `Whisper` by aligning overlapping transcriptions with Levenshtein distance to select the most accurate segments [28]. Despite different approaches, both projects rely on `faster-whisper` to reduce latency and contribute to the field. However, `Whisper`—including its variants—is GPU-bound and introduce major bottlenecks in real-time and parallel settings. `Whisper-T` [55] aims to reduce `Whisper`'s GPU load and power consumption through lightweight decoding, beam pruning, and CPU/GPU pipelining. However, it remains limited to a local, single-stream processing. Scalable, parallel `Whisper` for remote multi-client ASR systems remains an open challenge. No public implementation of `Whisper-T` is currently available.

Limitations of the State-of-the-art. The goal is to handle multiple requests on relatively modest hardware (e.g., one or a few GPUs). This raises challenges, including: 1) performance degradation in most `Whisper` implementations (except `faster-whisper`) under real-time workloads, and 2) the lack of model-level parallelization. Several `Whisper` implementations, including: `insanely-fast-whisper`[5] and `WhisperX` [56] (built on `faster-whisper`), are unsuitable for real-time use due to latency. The former performs poorly on short clips (e.g., 15–30 seconds) and does not support raw audio buffers, while the latter inherits `faster-whisper`'s limitations despite offering *diarization*[6] making it slower than its base implementation. Moreover, `Whisper` and its variants are not designed for concurrent multi-stream processing because GPU resources are quickly exhausted during inference, and multiple instances contend for GPU execution, A costly solution is horizontal scaling with one dedicated GPU for each `Whisper` instance. A cheaper alternative is asynchronous single-GPU processing, though support in `faster-whisper` and similar tools is limited.

IV. A Deep Dive into SWIM

`SWIM` is a multi-client, real-time ASR system built on `Whisper`. It is designed to: 1) process multiple multilingual audio streams concurrently, 2) maintain low latency and high throughput, and 3) deliver accurate transcriptions to all clients.

A. Overview

The actors in **`SWIM`**'s architecture are shown in Fig. 1. The system includes multiple clients ❶, each streaming audio to a dedicated service ❷. Each service: 1) builds an *audio buffer* from incoming chunks and submits it to the *shared ASR model* ❸ for transcription, and 2) maintains a *hypothesis buffer* ❽ to store intermediate transcriptions and apply a *local agreement* strategy to ensure reliability. The *shared ASR model* aggregates clips from all *audio buffers* ❺, runs inference via `faster-whisper` ❻, and forwards transcriptions to the *result dispatcher* ❼.

[2]Also known as *temporal-*, *forced-*, or *word-level alignment*.
[3]https://github.com/guillaumekln/faster-whisper
[4]https://github.com/OpenNMT/CTranslate2
[5]https://github.com/Vaibhavs10/insanely-fast-whisper
[6]*Diarization* segments audio by speaker, identifying who spoke when.

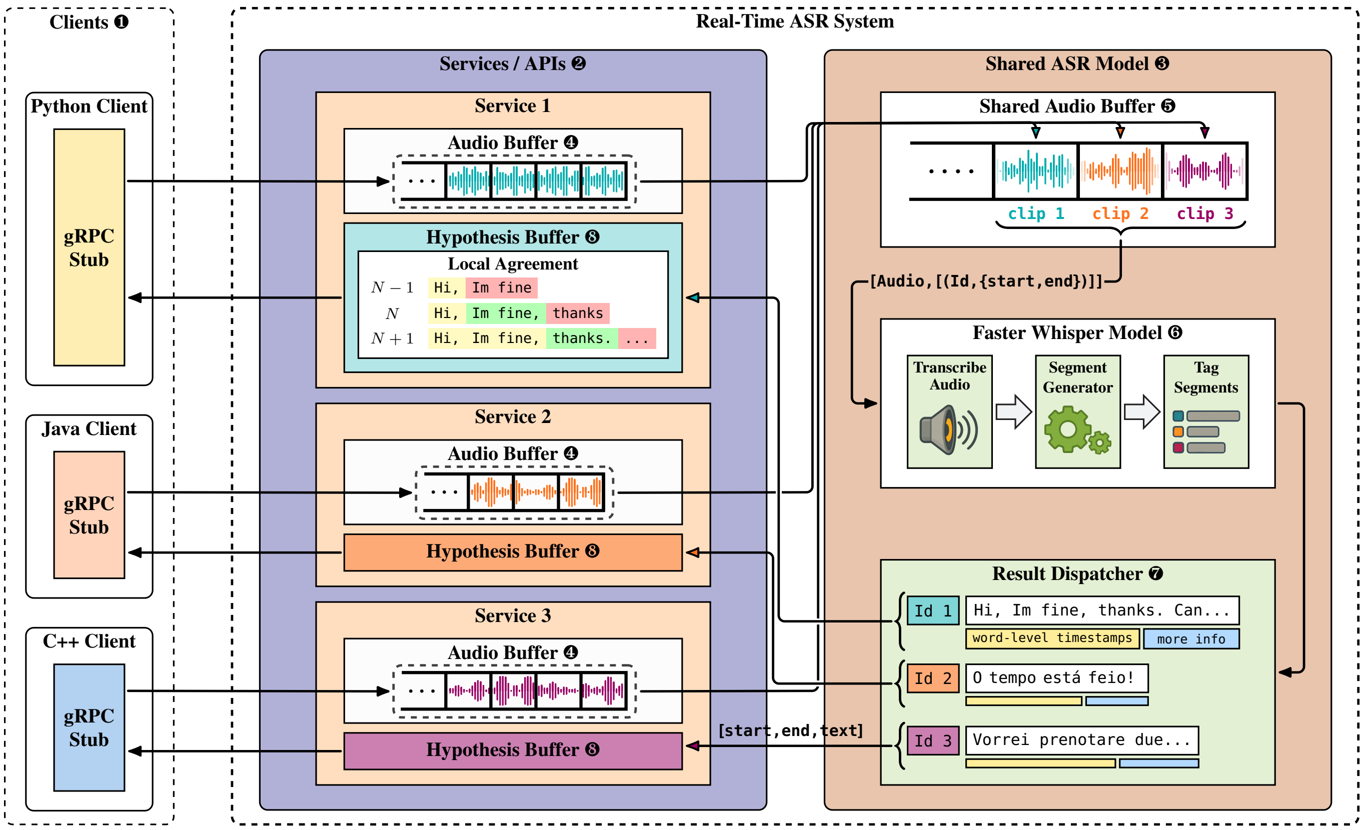


Figure 1: **SWIM** architecture. Multiple clients ❶ stream audio to dedicated *services* ❷, which forward it to a *shared ASR model* ❸ for parallel processing.

B. Clients

SWIM is a multi-client ASR system deployable on a remote server or locally (e.g., handling multiple audio sources on one machine). Clients (Fig. 1-❶) stream audio from audio-capturing devices (e.g., smartphones, computers, or boards like `Raspberry Pi` or `Arduino`) to the system. Client implementations are polyglot and platform agnostic, supporting any programming language compatible with the `gRPC`.[7] Transcriptions of the audio streams are returned via a *bidirectional* communication channel. Audio must be sampled at 16 kHz, as required by `Whisper`, to enable proper buffer handling (e.g., timestamp-based chunking and synchronization).

C. Services

Services (Fig. 1-❷) handle client audio streams. Each service is dedicated to a single `gRPC` client and builds an *audio buffer* from incoming chunks. These *audio buffers* (color-coded as ●, ●, and ● in Fig. 1-❷) are continuously sent to the `Whisper` model (arrows from ❷ to ❸) to ensure real-time processing. To reduce overhead, the audio buffer is kept short (typically 10–15 seconds). The oldest portion is trimmed once the buffer reaches its maximum size. Trimming affects only content already marked as *confirmed* in the *hypothesis buffer*[8] by the *local agreement* mechanism. Thanks to the *word-level* timestamp alignment in `faster-whisper`, the buffer can be segmented at word boundaries, avoiding splits. Each segment (Fig. 1-❼) is a continuous sequence of words with start/end timestamps and metadata such as full text and detected language. This design supports overlap resolution, insertion, and deletion in the *hypothesis buffer* (Fig. 1-❽). Specifically, the buffer tail (Fig. 1-❹) is truncated at the timestamp of the first confirmed word occurring before the buffer midpoint, i.e., $\ell < \frac{B}{2}$, where B is the total buffer length (e.g., 15 seconds). As shown in Fig. 1-❷ (Service 1), the confirmed portion of the *hypothesis buffer* is marked in *yellow* ● and *green* ● while the unconfirmed portion appears in *red* ●. When a segment is confirmed, it is marked *green* and returned to the client with its start and end timestamps. See Sect. IV-E for details on the local agreement policy.

D. Shared ASR Model

Shared Audio Buffer. **SWIM** is a multi-client ASR system in which services do not run dedicated `Whisper` instances. Instead, they rely on a *shared ASR model*[9] (Fig. 1-❸) shared across all services (Fig. 1-❷). The *shared model* operates on a single *shared audio buffer* (Fig. 1-❺), also shared among all services to enable parallel processing of multiple audio streams. It processes streams by continuously: 1) aggregating

[7]https://grpc.io

[8]The *hypothesis buffer* (Fig. 1-❽) stores transcriptions from the shared model's result dispatcher (Fig. 1-❼) and compares the latest confirmed transcription with the current one. Details in Sect. IV-E.

[9]The *shared ASR model* (or *shared model*) is a custom wrapper around `faster-whisper` providing methods that enable its use in multi-client setting.

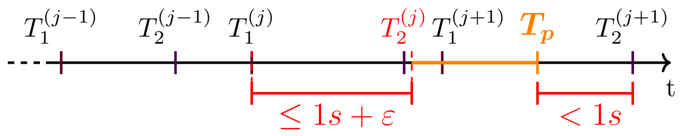


Figure 2: Audio chunk processing time diagram.

service *audio buffers* into the *shared audio buffer*, 2) running inference via a `faster-whisper` instance (Fig. 1-❻), and 3) tagging and returning results to the services (Fig. 1-❼).

The *shared audio buffer* is a single raw buffer built from the individual *audio buffers* of each service plus metadata to track their provenance. Alongside it, the system maintains pairs of: 1) the associated service's unique identifier, and 2) the start and end timestamps of its audio segment. The *shared audio buffer* is populated by the *shared ASR model* once all services are ready to process their *audio buffer* (i.e., after receiving a new audio chunk). To avoid deadlocks when one or more services are not ready, the system applies policy mechanisms. For instance, it may wait 1–2 seconds for a service to become ready; if not, it skips the service and either: 1) pauses or removes it from the system, or 2) excluding it from the current transcription while keeping it active for subsequent ones.

Each service operates autonomously, processing its own audio stream without interference from others. As a result, synchronization is implicitly handled by the system architecture. **`SWIM`** assumes a persistent audio stream transmitted in fixed-duration chunks, with each chunk having a duration $D \in (0, 1]$. As shown in Fig. 2, there always exist two consecutive chunks (D_1, T_1) and (D_2, T_2), where D_j denotes the chunk received by client C_j at time T_j. Let $(T, i) \leftarrow (\max\{T_1, T_2\}, \arg\max_{i\in\{0,1\}}\{T_i\})$, then:

$$T - T_{|i-1|} - \varepsilon \leq 1s$$

where:

1) $T_{|i-1|}$ is the timestamp of the last chunk from $C_{|i-1|}$, and
2) $\varepsilon = o(1)$ represents the network latency of T.

As shown in Fig. 2, this assumption ensures that the delay while waiting for all services to be ready is bounded by

$$\max(0, 1 - T_p) < 1s,$$

where T_p is the transcription processing time. Intuitively, this is the worst-case in which one service waits for another to finish processing and sending its chunk. The outer maximum with zero avoids negative delays: if the *shared ASR model* takes longer than 1 second to process all active services, the next chunk will already have arrived for every connected services, resulting in no extra waiting time.

Faster `Whisper` Model. In the adopted architecture, the entire *shared audio buffer* is sent to the model as a single, continuous waveform. Client contributions are identified by unique identifiers and temporal intervals `(start, end)`, labeled as `Timestamp` in Fig. 1 (from ❺ to ❻).

Inference is performed on the entire *shared buffer*, which partitions the audio into clips, each tagged with a unique client identifier. For each clip—shown as ●, ●, ● segments in Fig. 1-❺—the system extracts a corresponding transcription.

As noted in Sect. IV-C, service *audio buffer* are limited to 10–15 seconds. This reduces computational load by limiting the input per inference step and avoids excessively long clips, which are truncated beyond approximately 30 seconds.

Result Dispatcher. After inference, text segments are returned in processing order, allowing the system to associate each segment with its client identifier. Before returning them, the system adjusts the segment timestamps—including word-level timings—to match the original service audio buffer, since they are computed relative to the *shared audio buffer*, not to the service-specific *audio buffer*. Finally, resulting segments are sent to the corresponding service, which processes them and delivers the transcription to the client (arrow ❼→❽).

E. Hypothesis Buffer with Local Agreement

`SWIM` uses an adapted *hypothesis buffer*, inspired by `Whisper-Streaming` [27], that applies a *local agreement* strategy to ensure transcription reliability. Unlike prior work, **`SWIM`** relies on a *similarity metric* [28, 50] rather than strict token equality to govern the *hypothesis buffer* behavior. This design choice is motivated by preliminary studies without modifying the local agreement mechanism. **`SWIM`** occasionally encountered issues due to strict token equality checks, as minor transcription fluctuations—though less frequent than in `Whisper-Streaming`—caused equality to fail. To address this, **`SWIM`** uses the `QRatio` metric,[10], also known as normalized Bachmann's Indel distance, defined as:

$$QRatio(s1, s2) = \left(1 - \frac{d(s1, s2)}{|s1| + |s2|}\right) \times 100,$$

where $d(s1, s2)$ is the Indel distance [57, 58] between two strings $s1$ and $s2$—a Levenshtein distance [50] with substitution cost two—and $|s1|$ and $|s2|$ are string lengths. These changes serve two purposes: 1) handling punctuation, apostrophes, and minor discrepancies introduced by `Whisper` or individual services, and 2) populating the local agreement list (words with `(start, end, text)`) and retaining content slightly altered by the model. When a new segment arrives, the system replaces the previous one and inserts the first n consecutive tokens whose similarity ratio is at least 98% relative to the corresponding tokens in the previous segment. This overlap detection improves the accuracy and reliability of the hypothesis buffer. The local agreement mechanism confirms tokens by comparing the new and previous segments, using a 95% similarity threshold. This tolerates minor variations (e.g., punctuation changes) without extra iterations while preserving transcription accuracy. An example appears in the local agreement box under the first ❽ of Fig. 1. If no confirmation occurs under the local agreement policy, a fallback strategy is applied before the next iteration. The system takes the first half of the previous transcription (based on word count), generates its ordered prefixes, and compares them with prefixes of the new transcription, restricted to tokens whose end timestamps fall within that half. This limits matching to the relevant portion and avoids premature confirmation. The prefix with

[10]Later referred to as *similarity metric*, *similarity ratio*, or *similarity score*.

the highest similarity score is selected, and its tokens are confirmed individually adopting the same similarity metric as in the local agreement. This mechanism recovers from phrasing or tokenization shifts that could otherwise prevent confirmation until the audio buffer is trimmed. This local agreement mechanism is beneficial not only in multi-client scenarios but also in single-client settings, as it reduces latency. By confirming tokens based on similarity rather than strict equality, the system adapts to minor transcription variations, ensuring a smoother and more responsive user experience.

V. Evaluation

The evaluation of **SWIM** utilizes four datasets: the *Google Fleurs dataset* [46], the *Multilingual LibriSpeech dataset* [45], the *LibriSpeech-Long dataset* [44], and the *Italian Parkinson's Voice dataset* [59, 60, 61]. We use `large-v3-turbo`, a pruned and fine-tuned version of the pre-trained `large-v3` model, which offers significant improvements in efficiency. A replication package for the experiments is available at:

https://doi.org/10.5281/zenodo.15856828

A. Datasets

Description. The *Google Fleurs dataset* provides a multilingual benchmark with recordings from 102 languages, making it ideal for evaluating **SWIM**'s robustness across a wide range of speech types. The *Multilingual LibriSpeech dataset* consists of read audiobooks in eight European languages, supporting evaluation in moderately resourced multilingual settings. The *LibriSpeech-Long dataset* extends LibriSpeech by including longer audio segments, testing the model's ability for long-range temporal modeling. Finally, the *Italian Parkinson's Voice dataset* contains Italian speech samples from Parkinson's disease patients and control subjects, enabling assessment of **SWIM** in pathological speech recognition scenarios, which is critical for evaluating performance on impaired speech.

Preparation. We preprocess all datasets to align audio with transcriptions. Using the `Whisper large-v3` [13], we obtain word-level alignments for precise audio-text mapping. The resulting word-level timestamps are crucial to evaluate **SWIM** in real-time ASR scenarios requiring tight synchronization.

We chose `Whisper large-v3` for its balance of latency, accuracy, and speed. Its state-of-the-art performance offers a strong baseline for evaluating **SWIM** under our constraints and the limitations of the `large-v3 turbo` variant.

The prepared datasets enable evaluation of **SWIM** across multiple scenarios: 1) *Multilingual LibriSpeech* and *Google Fleurs* assess multilingual concurrent performance ($\approx$15 s and $\approx$10 s average lengths, respectively); 2) *LibriSpeech-Long* evaluates long-form audio processing ($\approx$5 min average); and 3) *Italian Parkinson's Voice* provides an intermediate case ($\approx$1 min segments) with pathological speech and Italian regional accents.

B. Experimental Setup

We conduct experiments on a server with: 1) an NVIDIA RTX 6000 Ada GPU (48 GB VRAM), 2) an Intel© Xeon© Gold 5412U CPU (24 cores, 48 threads), 3) 128 GB RAM, and 4) Ubuntu 22.04 LTS.

C. Experiments and Results

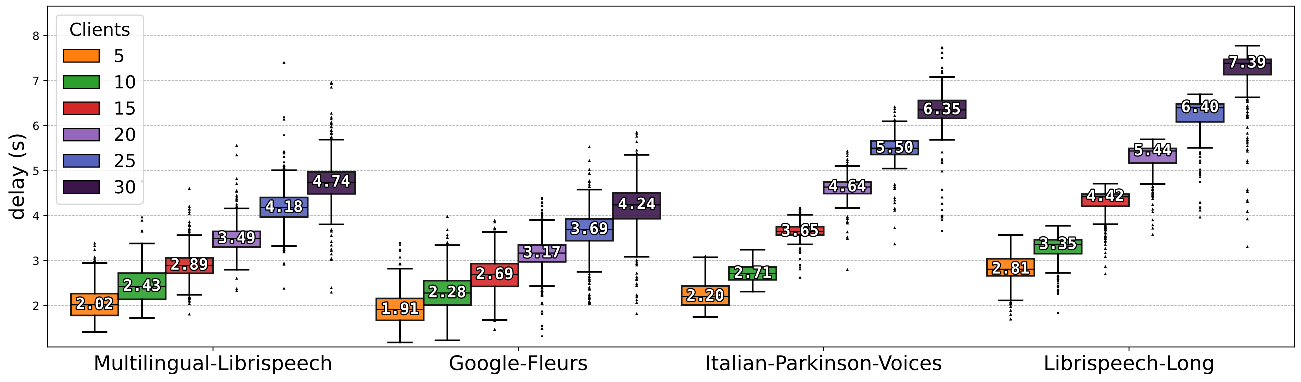
Delay Distribution Plots. Figures 3a and 3b show delay distributions by language under varying client loads, using chunk sizes of 1.0 s and 0.5 s, respectively. Delay is measured as the time between the end of a segment's response and the start of the corresponding input segment, reflecting user-perceived latency. Besides client load, delay depends on shared buffer utilization, which varies with average audio length. For instance, longer samples in *LibriSpeech-Long* increase delay compared to shorter samples in *Google Fleurs*, as the shared buffer remains occupied longer. In each plot, box groups are labeled by dataset on the x-axis, with delay (s) on the y-axis. Colors indicate the number of clients. Delays increase roughly linearly with concurrency, helping identify the point where performance degrades and scaling or offline transcription becomes necessary. With fewer clients, shorter chunks (0.5 s) reduce delay by enabling more frequent updates. At higher concurrency, however, 1.0 s chunks perform better due to lower processing overhead. These results show the importance of tuning chunk duration. With up to 15 clients, shorter chunks (0.5 s) reduce delay by about 11% compared to 1.0 s chunks. At higher loads (more than 15 clients), 1.0 s chunks lower delay by roughly 8% due to reduced overhead. Overall, **SWIM** dynamically adapts to workload changes, balancing throughput and latency as client count varies.

WER Distribution Plots. Figures 3c and 3d show WER distributions per language under varying client loads, using chunk sizes of 1.0 s and 0.5 s, respectively. Box groups are labeled by dataset on the x-axis, with WER on the y-axis. WER remains relatively stable across load levels, indicating that transcription quality is largely unaffected by parallel streams. The WER is computed as:

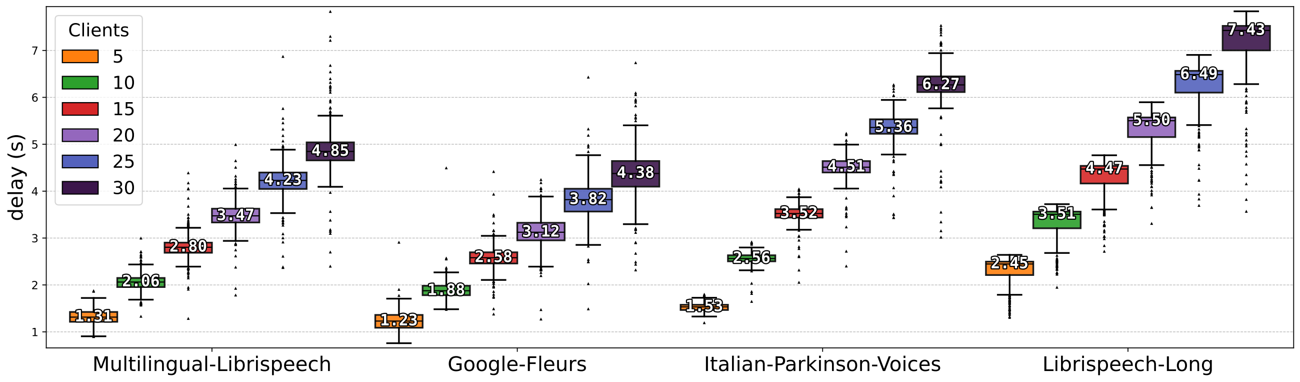
$$\text{WER} = \frac{S + D + I}{N}$$

where S, D, and I denote the number of substitutions, deletions, and insertions, and N is the number of words in the transcription. For fair comparison, WER is calculated over the aggregate transcription output of the full dataset for each run, regardless of client count, ensuring consistent data distribution and avoiding subsampling artifacts.

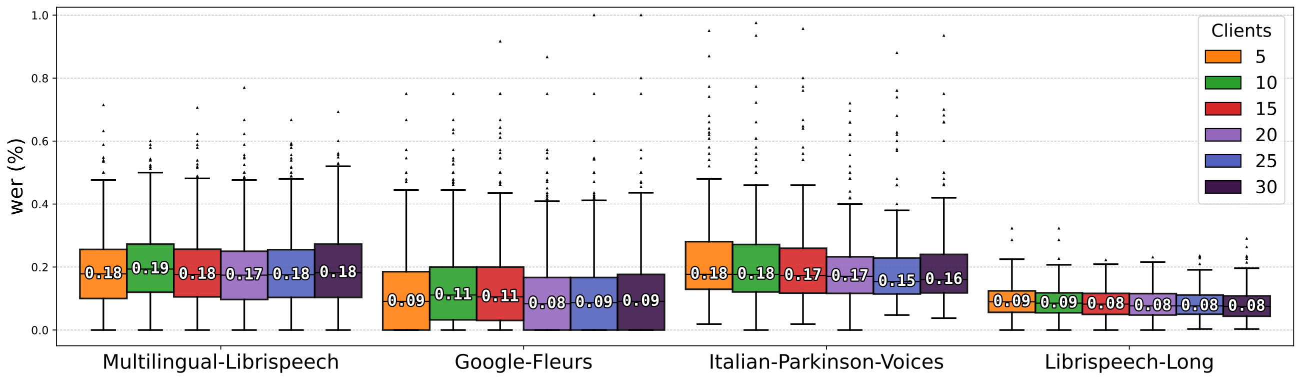
As expected, WER remains relatively stable across load levels, with only minor fluctuations. This suggests that accuracy depends more on architectural choices—particularly the shared buffer—than on the number of concurrent requests. **SWIM** consistently maintains low WER in multilingual settings, highlighting a key strength. Chunk duration only marginally affects WER values. The largest deviation appears in the *Italian Parkinson's Voice* dataset. Lower latency can trigger earlier segment confirmation, reducing contextual information before finalization of a transcription and mainly affecting punctuation and casing. For instance, 1.0 s chunks allow more linguistic context than 0.5 s chunks, which are more prone to punctuation and boundaries errors. This effect amplified in the Parkinson's dataset, where speakers often articulate isolated words with long pauses. The lack of continuous speech context, coupled with rapid segment finalization, leads to formatting errors inflating WER despite largely correct



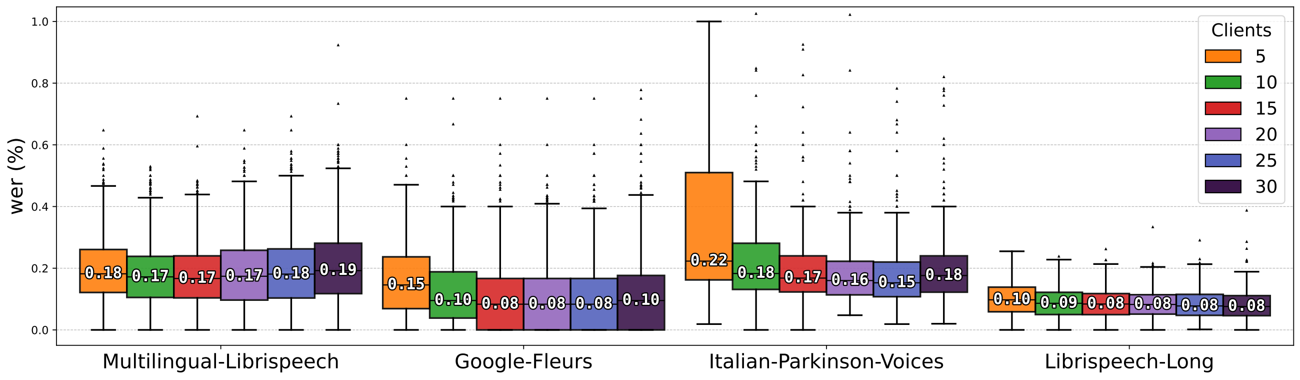
(a) Chunk duration of 1.0s



(b) Chunk duration of 0.5s

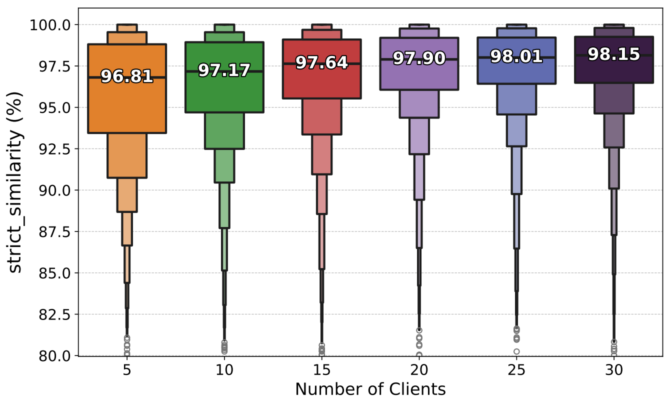


(c) Word Error Rate with chunk duration of 1.0s

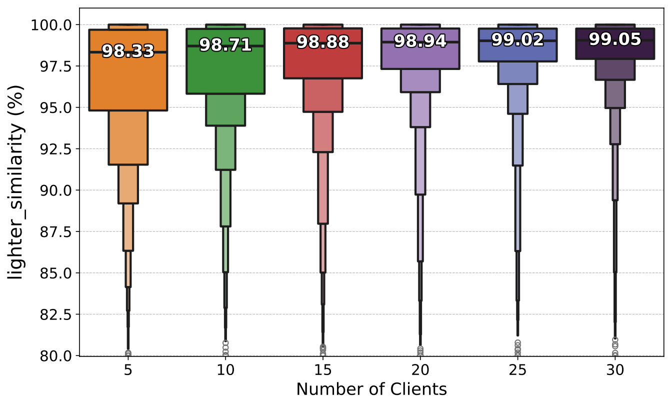


(d) Word Error Rate with chunk duration of 0.5s

Figure 3: Evaluation results with different chunk durations using box plots.



(a) Strict similarity



(b) Lighter similarity

Figure 4: The similarity score distribution per client across all datasets using boxen plots.

semantic content. Nevertheless, **SWIM** maintains a competitive WER across datasets and languages, demonstrating its robustness under high concurrency.

Similarity Distribution Plots. Figures 4a and 4b show similarity score distributions by client count across all datasets. Scores (in %) are visualized with boxen plots,[11] offering a detailed view of data distribution. Similarity scores are expected to exceed WER, since WER penalizes formatting differences (e.g., punctuation and casing) that do not affect semantics. Both similarity measures use `QRatio`. For each response, we extract the corresponding transcript segment using its timestamps and compute similarity against the generated text. The *strict similarity* applies standard `QRatio`, while the *lighter similarity* ignores punctuation and casing. As expected, strict scores are slightly lower, though the difference is small, indicating strong semantic accuracy even under stricter comparison. Similar to the trends in Figures 3c and 3d, similarity scores improve as the number of concurrent clients increases. This was expected because longer processing times allow more audio to accumulate in each service buffer, providing additional context for inference within the shared buffer. Finally, both similarity scores remain stable across datasets, with only minor variations. As shown in Fig. 4, median scores exceed 96% for the strict metric and 98% for the lighter one, regardless of client count. This confirms that **SWIM** maintains high semantic accuracy across diverse datasets and languages—a crucial requirement for real-time ASR systems.

Linguistic Delay Distribution Plots. Figures 5a and 5b show per-language delay distributions for the *Google Fleurs* and *Multilingual LibriSpeech* datasets. Both use violin plots, which combine box plots with kernel density estimation to visualize delay distributions. Due to **SWIM**'s client synchronization mechanism, delays remain consistent across languages, as they are mainly driven by processing time. Distributions are similar across languages with only minor median variations, and their narrow spread indicates stable, predictable latency—crucial for real-time use. These results confirm that **SWIM** generalizes well across languages without sacrificing responsiveness.

[11]Boxen plots extend box plots to better capture distribution shape.

Linguistic WER Distribution Plots. Figures 6a and 6b show WER distribution across languages in the same settings.

As expected, WER is slightly influenced by the linguistic characteristics of each language. English, for instance, typically yields the lowest WER, while languages with diacritics or complex orthographic rules may show higher error rates.

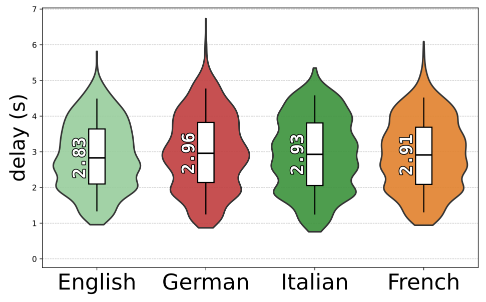
Nonetheless, the plots reveal that WER distributions remain consistent across languages, with only minor differences in median values. The small variations can be attributed to distribution tails and outliers. This indicates that languages rich in diacritics or with complex orthographies do not significantly degrade **SWIM**'s transcription accuracy.

Unlike `whisper-streaming`, which already demonstrates high accuracy, **SWIM** maintains comparable WER across languages—including those with complex phonetic structures—even as the number of concurrent clients increases.

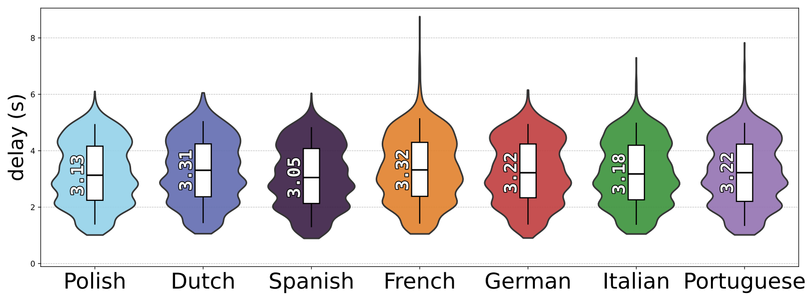
Hypothesis vs. Confirmation Delay Plot. Figure 7 shows the mean delays—visualized as a bar plot—for both hypothesis and confirmation responses on the *Multilingual LibriSpeech* dataset, using a fixed chunk duration of 1.0 s. The purpose of this plot is to illustrate that, beyond confirmed responses—which represent finalized, immutable transcriptions—hypothesis responses can provide preliminary transcriptions earlier in the pipeline.

The mean is used here because bar plots naturally encode an arithmetic average per category, and the goal of this figure is to highlight the *relative gap* between hypothesis and confirmation delays rather than to characterize their full distributions. It should be noted, however, that the resulting values may appear slightly different from those in Figure 3a, which reports the median via box plots: in our streaming context, delay distributions are often right-skewed, so the mean tends to exceed the median.

Regardless of the metric used, thanks to the local agreement mechanism, hypothesis responses serve as early approximations of the final confirmed outputs. These intermediate results are often sufficiently accurate to be used for immediate feedback to users. This distinction is especially valuable in real-time applications where low-latency feedback is critical, such as live captioning or when supplying incremental input to *large language models* in interactive agent systems.

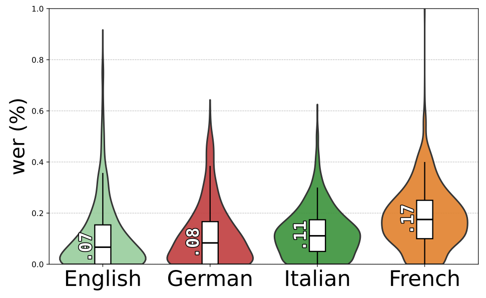


(a) Google Fleurs dataset

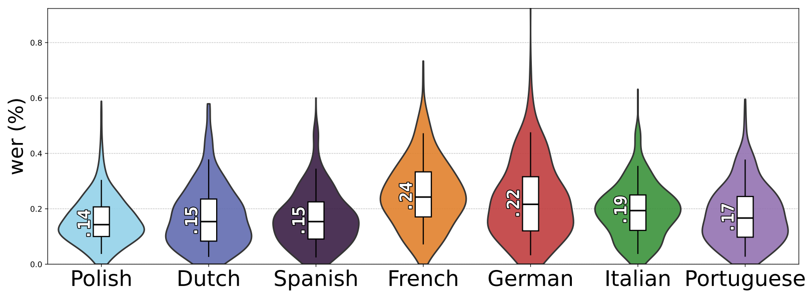


(b) Multilingual LibriSpeech dataset

Figure 5: Delay per language across the Multilingual LibriSpeech dataset and Google Fleurs dataset using violin plots.



(a) Google Fleurs dataset



(b) Multilingual LibriSpeech dataset

Figure 6: WER per language across the Multilingual LibriSpeech dataset and Google Fleurs dataset using violin plots.

D. Concluding Remarks

`Whisper-Streaming` [27], in an English-only setting with a single fixed client, achieves a WER of approximately 8.2% and an average transcription delay of about 3.4 seconds. In comparison, **SWIM** scales to 5 concurrent clients while maintaining a similar WER (see Tables 3c and 3d on the *LibriSpeech-Long dataset*) and reducing average delay to roughly 2.4 seconds. Even at 15 concurrent clients, **SWIM** sustains stable latency and accuracy comparable to `Whisper-Streaming`. Furthermore, **SWIM** maintains low delay and high transcription accuracy—measured by WER and similarity scores—across diverse datasets and multilingual scenarios with up to 20 concurrent clients. These results demonstrate **SWIM**'s effective scalability, significantly increasing throughput without sacrificing transcription quality.

E. Noisy, Real-Time Environments

Our current evaluation leverages clean and pre-segmented datasets with reliable word-level alignments, which enables controlled measurement of latency, WER, and similarity at scale. In real-time environments, audio streams are often affected by far-field capture, reverberation, background noise, overlapping speech, and variable network conditions (e.g., jitter, packet loss). These factors can degrade both hypothesis stability and confirmation latency by reducing local agreement and delaying segment finalization.

System behavior under noise. While **SWIM** includes mechanisms that are beneficial in noisy scenarios—such as an optional VAD preprocessor to filter non-speech segments and an adaptive hypothesis buffer that delays confirmation upon repeated local agreement failures—its performance has not been quantified on streaming datasets explicitly designed to represent noisy, time-varying conditions (e.g., telephony channels, public spaces, or multi-speaker overlap). In practice, we expect: 1) increased hypothesis churn (more frequent revisions) due to reduced acoustic confidence, 2) longer confirmation delays when the QRatio-based local agreement requires additional context to disambiguate noisy segments, and 3) minor WER inflation primarily from punctuation, casing, and segmentation decisions, with semantic similarity remaining more stable than strict WER.

Potential extensions. To improve robustness in noisy deployments, several extensions are natural and align with **SWIM**'s architecture:

1) front-end denoising and dereverberation (e.g., spectral subtraction or modern neural speech enhancement) prior to model inference,
2) online endpointing and diarization to better segment and attribute speech under overlap,
3) adaptive chunking and dynamic buffer policies driven by per-segment confidence and hypothesis stability, balancing latency against accuracy, and
4) resilience to network variability via jitter-aware scheduling and backpressure controls in the shared buffer.

Future work. Building or sourcing datasets that simulate realistic noisy, real-time conditions—ideally multilingual and covering diverse acoustic conditions—remains a key avenue for future work. We plan to evaluate **SWIM** with controlled noise injections (SNR sweeps, reverberation, and overlap) on existing corpora adapted for streaming and report a comprehensive set of real-time metrics, including stability and timestamp jitter, alongside WER and similarity. This will strengthen practical relevance and provide a more complete characterization of performance in noisy, real-time deploy-

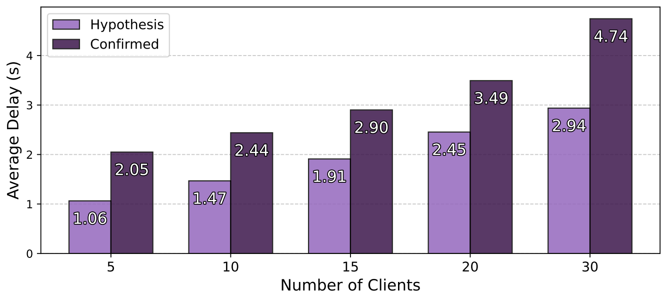


Figure 7: Hypothesis delay vs Confirmation delay with fixed chunk duration of 1.0s on Multilingual LibriSpeech dataset.

ments.

VI. Threats to Validity

In our discussion, we follow Wohlin et al. [62]'s taxonomy.

A. Construct Validity

Transcription Quality Metrics. Our evaluation also relies on standard ASR metrics such as WER to assess transcription quality. However, these metrics may not fully reflect the nuances of real-time multilingual transcription, especially in cases involving code-switching, domain-specific vocabulary, or varied audio conditions across clients.

Real-Time Performance Metrics. We use latency and throughput to evaluate real-time performance. However, the definition of "real-time" varies by application. Our processing thresholds may not meet the stricter requirements of scenarios like live broadcasting or emergency response systems, where sub-second latency is critical.

Scalability Assessment. Our evaluation measures scalability by the number of concurrent clients the system can support. This may not fully reflect real-world conditions, where clients vary in audio quality, language, and network stability.

Mitigation Strategies. To address these threats to construct validity, we adopted several strategies: 1) for transcription quality, we complement standard WER metrics with our QRatio-based local agreement mechanism and a fallback confirmation strategy within the adaptive hypothesis buffer. This enables more nuanced handling of multilingual and code-switched input, especially in noisy or ambiguous contexts; 2) for real-time performance, our latency threshold applies only to confirmed segments, while unconfirmed portions are updated dynamically using word-level indicators. This better reflects practical expectations for streaming ASR systems; nd 3) for scalability, **SWIM** was evaluated on four datasets (including two multilingual) recorded in both studio and real-world conditions, covering diverse speakers and varying audio quality. This helps ensure that our results generalize to real-world deployments rather than idealized settings.

B. Internal Validity

Whisper Model Limitations. The underlying Whisper model has known limitations when handling silent audio segments, occasionally producing hallucinated or repeated phrases. These artifacts can inflate error rates and may not accurately reflect the performance of the **SWIM** system architecture.

Audio Quality Variations. Inconsistent audio quality across test samples may affect our evaluation. Background noise, microphone fidelity, speaker accents, and recording conditions can introduce biases, favoring certain client configurations or audio types.

Mitigation Strategies. To address these threats to internal validity, we adopted several strategies: 1) to mitigate limitations of the Whisper model, we support optional voice activity detection as a preprocessing step. This filters out non-speech segments, reducing hallucinations caused by silence; and 2) to handle audio quality variations, **SWIM** includes a fallback mechanism that activates after repeated local agreement failures, ensuring reliable output even under challenging acoustic conditions.

C. External Validity

Dataset Limitations. Our evaluation relies on selected datasets that may not capture the full diversity of real-world audio streams. If these datasets predominantly feature clean, studio-quality recordings or limited language combinations, the results may not generalize to noisy environments, telephone audio, or underrepresented languages in Whisper's training data.

Client Configuration Constraints. Our experiments assume clients use uniform audio chunk durations ($\leq$ 1 second) and fixed sampling rates (16kHz). However, real-world deployments may involve more diverse configurations, including varying chunk lengths and audio formats, which could impact system performance.

Deployment Environment. Our evaluation takes place in controlled server environments that may not capture the constraints of real-world deployments, such as limited computational resources, fluctuating network conditions, and varying system loads.

Mitigation Strategies. To address threats to external validity, we implemented several strategies: 1) to overcome dataset limitations, our evaluation covers diverse audio conditions—including noisy environments, telephony audio, and multiple languages—beyond clean, studio-quality recordings to improve generalizability; 2) concerning client configuration constraints, **SWIM** supports variable chunk sizes, with some settings yielding better performance. Although a 16kHz sampling rate is currently required, we are extending support for automatic upsampling and downsampling to increase compatibility with different input formats; and 3) while testing occurred in controlled environments, **SWIM** is actively deployed in production by VoiSmart, serving both internal and external clients, confirming its real-world effectiveness.

D. Conclusion Validity

Baseline Comparison Limitations. Comparisons with existing systems such as Whisper-Streaming and Whispy may be affected by differences in implementation, evaluation settings, or optimization goals, which can make direct performance comparisons potentially misleading.

Statistical Analysis Limitations. Our analysis is limited by sample size, which may affect the reliability of our conclusions. Small samples can yield unstable performance estimates and may not fully capture true variability in system behavior.

Mitigation Strategies. To address threats to conclusion validity, we adopted several strategies: 1) although implementation differences can impact baseline comparisons, our primary contribution is a novel architectural approach to multi-client ASR, which can be adopted or adapted by other systems regardless of their implementation. This highlights methodological innovation over direct performance advantage; and 2) to strengthen statistical validity, we evaluated **`SWIM`** on datasets with long audio samples (up to 5 minutes) and hundreds of entries, tested across varying numbers of concurrent clients. This reduces small-sample effects and offers a more comprehensive assessment of system behavior under diverse load conditions.

VII. Conclusion

We present **`SWIM`**, a novel multi-client real-time ASR system that enables a single `Whisper` model instance to process multiple concurrent multilingual streams with low latency and high accuracy. Unlike multi-instance setups, **`SWIM`** achieves model-level parallelization through a shared buffer merging strategy, optimizing resource utilization while preserving transcription quality. Key innovations include an adaptive hypothesis buffer with `QRatio`-based agreement and a fallback confirmation mechanism to manage linguistic variability. **`SWIM`** outperforms existing single-client methods and provides scalable, cost-effective ASR suitable for enterprise deployment.

References

[1] S. Alharbi, et al., "Automatic Speech Recognition: Systematic Literature Review," *IEEE Access*, vol. 9, 2021.

[2] A. Ram, et al., "Conversational AI: The Science Behind the Alexa Prize,", arXiv:1801.03604, Jan. 2018.

[3] M. Gerosa, et al., "A Review of ASR Technologies for Children's Speech," WOCCI'09. ACM, Nov. 2009.

[4] A. Tjandra, et al., "Attention-Based wav2text with Feature Transfer Learning," ASRU. Jan. 2017, pp. 309–315.

[5] P. Dubey and B. Shah, "Deep Speech Based End-To-End Automated Speech Recognition (ASR) for Indian-English Accents," arXiv:2204.00977, pp. 1–6, Apr. 2022.

[6] R. Prabhavalkar, et al., "End-to-End Speech Recognition," *Trans. Audio, Speech and Lang. Process.*, 2023.

[7] J. Li, "Recent Advances in End-to-End Automatic Speech Recognition," *APSIPA Transactions on Signal and Information Processing*, vol. 11, no. 1, pp. 1–64, Apr. 2022.

[8] S. Furui, "50 Years of Progress in Speech and Speaker Recognition Research," *Trans. Comput. Inf. Technol.*, 1(2), pp. 64–74, Nov. 2005.

[9] M. Gales and S. Young, "The Application of Hidden Markov Models in Speech Recognition," *Foundations and Trends® in Signal Processing*, vol. 1, no. 3, 2008.

[10] H. Aldarmaki, A. Ullah, S. Ram, and N. Zaki, "Unsupervised Automatic Speech Recognition: A Review," *Speech Communication*, vol. 139, pp. 76–91, Apr. 2022.

[11] A. Baevski, et al., "wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations," NeurIPS'20, Dec. 2020, pp. 1–12.

[12] G. Chen, et al., "GigaSpeech: An Evolving, Multi-Domain ASR Corpus with 10,000 Hours of Transcribed Audio," INTERSPEECH'21, ISCA, 2021, pp. 4376–4380.

[13] A. Radford, et al., "Robust Speech Recognition via Large-Scale Weak Supervision," ICML'23. Honolulu, Hawaii: PMLR, Jul. 2023, pp. 28 492–28 518.

[14] A. Hannun, et al., "Deep Speech: Scaling up End-to-End Speech Recognition," arXiv:1412.5567, Dec. 2014.

[15] D. Amodei, et al., "Deep Speech 2," ICML'16, PMLR, Jun. 2016, pp. 173–182.

[16] A. Morris, et al., "Multi-Stream Adaptive Evidence Combination for Noise Robust ASR," *Speech Communication*, vol. 34, no. 1-2, pp. 25–40, 2001.

[17] R. Li, et al., "Multi-Stream End-to-End Speech Recognition," *Trans. Audio Speech Lang. Process*, vol. 27, no. 6, pp. 1026–1039, Jun. 2019.

[18] P. N. Garner, et al., "Real-Time ASR from Meetings," INTERSPEECH'09. ISCA, Sep. 2023, pp. 2119–2122.

[19] Arriaga, et al., "Evaluation of Real-Time Transcriptions Using End-to-End ASR Model," arXiv:2409.05674.

[20] A. V. Ivanov, et al., "Speed vs. Accuracy," IWSDS'16, Springer, Jan. 2016, pp. 1–12.

[21] S. Ahmed, et al., "Preech: A System for Privacy-Preserving Speech Transcription," USENIX Security, Aug. 2020, pp. 2703–2720.

[22] H. Abdullah, et al., "SoK: The Faults in our ASRs," SP'21, IEEE, May 2021, pp. 730–747.

[23] A. Geirgescu, et al., "Performance vs. HW Requirements in State-of-the-Art Automatic Speech Recognition," *Audio Speech Music Process.*, pp. 28:1–28:30, 2021.

[24] F. Ramirez, et al., "Anatomy of Industrial Scale Multilingual ASR," arXiv:2404.09841, 2024.

[25] H. Zhu, J. Wang, G. Cheng, P. Zhang, and Y. Yan, "Decoupled Federated Learning for ASR with Non-IID Data," INTERSPEECH'22, ISCA, 2023, pp. 2628–2632.

[26] S. Basak, et al., "Challenges and Limitations in Speech Recognition Technology," *Comput. Model. Eng. Sci.*, vol. 135, no. 2, pp. 1053–1089, Oct. 2022.

[27] D. Macháček, et al., "Turning Whisper into Real-Time Transcription System," CNLP'23, Nov. 2023, pp. 17–24.

[28] A. Bevilacqua, et al., "Whispy: Adapting STT Whisper Models to Real-Time Environ," *arXiv:2405.03484*, 2024.

[29] P. Polák, et al., "CUNI-KIT System for Simultaneous Speech Translation Task at IWSLT 2022,", pp. 277–285.

[30] V. Pratap, et al., "Scaling Up Online Speech Recognition Using ConvNets," INTERSPEECH, 2020.

[31] X. Cui, S. Lu, and B. Kingsbury, "Federated Acoustic Modeling for Automatic Speech Recognition," ICASSP'21, IEEE, Jun. 2021, pp. 6748–6752.

[32] Y. Zhang, et al., "Life After Speech Recognition," NDSS'19. Feb. 2019, pp. 1–15.

[33] K. Kuhn, et al., "Measuring the Accuracy of Auto-

matic Speech Recognition Solutions," *ACM Trans. Access. Comput.*, vol. 16, pp. 25:1–25:23, Jan. 2024.

[34] S. Latif, et al., "Speech Technology for Healthcare," *IEEE Rev. Biomed. Eng.*, vol. 14, pp. 342–356, 2021.

[35] D. Loakes, "Does ASR Have a Role in the Transcription of Indistinct Covert Recordings for Forensic Purposes?" *Front. Commun.*, vol. 7, Aug. 2022.

[36] L. R. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," *Proc. of IEEE*, vol. 77, no. 2, pp. 257–286, Feb. 1989.

[37] A. Gulati, et al., "Convolution-Augmented Transformer Speech Recognition," INTERSPEECH'20, pp. 536–540.

[38] Y. He, et al., "Streaming End-to-end Speech Recognition for Mobile Devices," ICASSP 2019, pp. 6381–6385.

[39] M. B. Hoy, "An Introduction to Voice Assistants," *Med. Ref. Serv. Q.*, vol. 37, no. 1, pp. 81–88, Jan. 2018.

[40] K. Han, et al., "Transformer in Transformer," NIPS'21, vol. 34, Dec. 2021, pp. 15 908–15 919.

[41] A. Vaswani, et al., "Attention Is All You Need," NIPS'17, Dec. 2017, pp. 6000–6010.

[42] V. Pratap, et al., "Scaling Speech Technology to 1,000+ Languages," *J. Mach. Learn. Res.*, 25, pp. 1–52. 2024.

[43] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: An ASR Corpus Based on Public Domain Audio Books," ICASSP'15, Apr. 2015, pp. 5206–5210.

[44] S. J. Park, et al., "Long-Form Speech Generation with Spoken Language Models," ICML'25. PMLR, Jul. 2025.

[45] V. Pratap, et al., "A Large-Scale Multilingual Dataset for Speech Research," INTERSPEECH 2020.

[46] A. Conneau, et al., "FLEURS: FEW-Shot Learning Evaluation of Universal Representations of Speech," SLT'22, Doha, Qatar: IEEE, Jan. 2023, pp. 798–805.

[47] D. Klakow and J. Peters, "Testing the Correlation of Word Error Rate and Perplexity," *Speech Communication*, vol. 38, no. 1–2, pp. 19–28, Sep. 2002.

[48] R. Sawata, Y. Kashiwagi, and S. Takahashi, "Improving Character Error Rate is Not Equal to Having Clean Speech," ICASSP'22, May 2022, pp. 991–995.

[49] S. Raybaud, D. Langlois, and K. Smaïli, "This Sentence Is Wrong," *Machine Translation*, vol. 25, pp. 1–34, 2011.

[50] Y. Li and B. Liu, "A Normalized Levenshtein Distance Metric," *Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 6, pp. 1091–1095, 2007.

[51] G.-X. Shi, et al., "Timestamp-Aligning and Keyword-Biasing End-to-End ASR Front-End for a KWS System," *Audio Speech Music Process.*, pp. 21:1–21:14, Jul. 2021.

[52] T. Han, W. Xie, and A. Zisserman, "Temporal Alignment Networks for Long-Term Video," CVPR'22, New Orleans, LA, USA: IEEE, Jun. 2022, pp. 2896–2906.

[53] J. Li, et al., "Neufa: Neural Network Based End-to-End Forced Alignment with Bidirectional Attention Mechanism," ICASSP'22, May 2022, pp. 8007–8011.

[54] X. Chu, et al., "TEGRA: Table Extraction by Global Record Alignment," SIGMOD 2015, pp. 1713–1728.

[55] R. Wang, Z. Xu, and F. X. Lin, "Efficient Whisper on Streaming Speech,"*arXiv:2412.11272*, 2024.

[56] M. Bain, et al., "WhisperX: Time-Accurate Speech Transcription of Long-Form Audio," INTERSPEECH 2023, pp. 4489–4493.

[57] M. Deza and E. Deza, *Encyclopedia of Distances*, 2009.

[58] P. H. Sellers, "On the Theory and Computation of Evolutionary Distances," *Appl. Math.*, 26(4):787–793, 1974.

[59] G. Dimauro, et al., "VoxTester, Software for Digital Evaluation of Speech Changes in Parkinson Disease," MeMeA'16. Benevento, Italy: IEEE, May 2016, pp. 1–6.

[60] G. Dimauro, et al., "Assessment of Speech Intelligibility in Parkinson's Disease Using a Speech-To-Text System," *IEEE Access*, vol. 5, pp. 22 199–22 208, Nov. 2017.

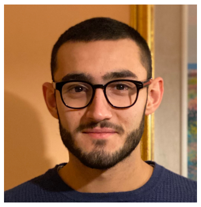
[61] G. Dimauro and F. Girardi, "Italian Parkinson's Voice and Speech," 2019.

[62] C. Wohlin, et al., *Experiment in SW Engineering*. 2012.

Federico Bruzzone (b. 2000) is currently a Ph.D. candidate in Computer Science at the Università degli Studi di Milano, Italy. He received his M.Sc. in Computer Science from the same institution. Currently, he is a researcher at the ADAPT Lab, where his work focuses on programming languages, compilers, and software maintenance and evolution. His research interests further extend to the intersection of software engineering and system optimization. He can be reached at federico.bruzzone@unimi.it.

Walter Cazzola is currently a Full Professor in the Computer Science Department of the Università degli Studi di Milano, Italy and the Chair of the ADAPT laboratory. He designed the mChaRM framework, @Java, [a]C#, Blueprint programming languages and he is currently involved in the designing and development of the Neverlang language workbench. He also designed the JavAdaptor dynamic software updating framework and its front-end FiGA. His research interests include software maintenance, evolution and comprehension, programming methodologies and languages. He served on the program committees or editorial boards of the most important conferences and journals about his research topics. He is associate editor for the Journal of Computer Languages published by Elsevier. More information are available at https://cazzola.di.unimi.it and he can be contacted at cazzola@di.unimi.it for any question.

Matteo Brancaleoni is a Senior Staff Engineer at VoiSmart Srl, Italy, where he develops unified communication systems with a focus on VoIP infrastructures and videoconferencing platforms. His expertise includes real-time communications, telephony protocols, WebRTC, and distributed architectures. His research and professional activities center on designing scalable, reliable communication frameworks and exploring their extension with artificial intelligence. He is particularly interested in protocol design, communication middleware, and AI integration into real-time distributed systems. He can be contacted at matteo.brancaleoni@voismart.it



Dario Pellegrino is currently a Master of Science student in Computer Science. He obtained his Bachelor's degree in Computer Science with a thesis project developed in collaboration with the ADAPT Lab and VoiSmart, focusing on the SWIM project. Recently, he started working as a Junior Software Engineer at VoiSmart, contributing to research and development activities on AI-based digital agents and assistants in the domain of telephony and PBX systems. His main research interests include software engineering and artificial intelligence, with a particular enthusiasm for both low-level aspects such as machine learning and the application of AI to software engineering. He can be contacted at dario.pellegrino@voismart.it