Hello, GenAI? Dissecting Human to Generative AI Calling

Ruizhi Cheng
Meta Platforms Inc. and
George Mason University
Menlo Park, CA, USA
ruizhicheng@meta.com

Surendra Pathak
George Mason University
Fairfax, VA, USA
spathak8@gmu.edu

Guowu Xie
Meta Platforms Inc.
Menlo Park, CA, USA
woo@meta.com

Matteo Varvello
Nokia Bell Labs
Murray Hill, NJ, USA
matteo.varvello@nokia.com

Songqing Chen
George Mason University
Fairfax, VA, USA
sqchen@gmu.edu

Bo Han
George Mason University
Fairfax, VA, USA
bohan@gmu.edu

Abstract

The rise of generative artificial intelligence (GenAI), powered by large language models, has led to the emergence of real-time, voice-based conversational applications that enable dynamic, multi-modal interactions for everyday tasks such as checking the weather or planning a trip. These *human-to-GenAI* calling applications blend speech processing, generative intelligence, and real-time communication, presenting new challenges in latency optimization, network infrastructure design, and resilience under load. Despite their growing popularity, little is known about the operational characteristics and performance of these applications. This paper conducts an empirical measurement of six human-to-GenAI calling applications from Google, Meta, Microsoft, and OpenAI, focusing on their input/output modalities, network behavior, latency metrics, and robustness. Our findings reveal key design choices and performance bottlenecks in these emerging applications. For example, the conversational latency often reaches several seconds, far exceeding the typical sub-second delays of human-to-human voice communication and potentially impairing interactivity. Moreover, voice-based GenAI traffic is inherently asymmetric: the uplink, carrying real-time human speech, benefits from streaming-based transmission, while the typically large downlink GenAI responses are better served through batch-based delivery.

CCS Concepts

• **Networks** → **Network measurement**; • **Computing methodologies** → **Artificial intelligence**.

Keywords

Network Measurement, Generative AI (GenAI), GenAI Calling, Real-time Human–GenAI Interaction

ACM Reference Format:
Ruizhi Cheng, Surendra Pathak, Guowu Xie, Matteo Varvello, Songqing Chen, and Bo Han. 2025. Hello, GenAI? Dissecting Human to Generative AI Calling . In *Proceedings of the 2025 ACM Internet Measurement Conference (IMC '25), October 28–31, 2025, Madison, WI, USA.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3730567.3764441

IMC '25, Madison, WI, USA
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1860-1/2025/10
https://doi.org/10.1145/3730567.3764441

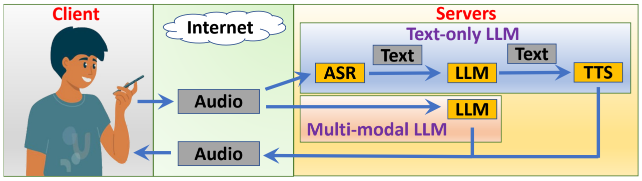

Figure 1: Workflow of human-to-GenAI calling (examined in §4.1). Servers rely on either text-only or multi-modal LLMs to process audio streams. ASR: Automatic speech recognition; LLM: Large language model; and TTS: Text to speech.

1 Introduction

Generative artificial intelligence (*GenAI*), exemplified by applications such as ChatGPT from OpenAI [81] and Gemini from Google [40], has rapidly evolved into a transformative tool for both personal and professional use. GenAI applications, powered by large language models (LLMs) [111], are capable of understanding, reasoning, and generating coherent dialogue with contextual awareness and task-oriented intelligence. These advancements have catalyzed a new wave of applications where GenAI agents engage in dynamic, interactive conversations with human users, facilitating tasks ranging from personal assistants [31, 52, 112] to code generation and debugging [56, 64, 85, 98].

Most recently, the fusion of conversational GenAI with real-time communication (RTC) has enabled emerging applications that blend audio interaction with generative intelligence. Notable examples include the calling features in applications such as ChatGPT and Gemini, marking a pivotal shift toward multi-modal, latency-sensitive conversational experiences. Different from traditional text-based services, human-to-GenAI calling (*GenAI calling*, for short) typically relies on complex multi-modal generative models [21] that involve capturing user inputs (*e.g.,* audio, text, or visual), invoking LLMs deployed on remote servers, and delivering dynamically generated responses in real time.

Despite the growing popularity, the performance characteristics of GenAI calling applications remain largely opaque, revealing critical gaps in our understanding of their real-time behavior and network demands, thus motivating the need for systematic measurement studies. For instance, characterizing the input modalities and the corresponding outputs is essential for capturing their end-to-end operational dynamics. Moreover, key latency metrics such as call setup time, which indicates the cold start delay, and time to first token (TTFT), which reflects user-perceived latency (§3), significantly influence user experience in interactive scenarios. In this paper, we aim to shed light on these dimensions through empirical measurement and detailed workflow analysis.

We measure six representative applications: **ChatGPT** [83] from OpenAI, **Copilot** [78] from Microsoft, **Gemini** [37] from Google, and **Instagram** [70], **Messenger** [72], and **WhatsApp** [73] from Meta. We exclude voice-assistant applications such as Apple Siri [11] as their current implementations do not incorporate GenAI capabilities [28]. We compare GenAI calling with voice-assistant applications in §4.6. We further exclude the mobile application of Claude [9] from Anthropic since, despite offering voice-based input, it currently does not support calling functionality for sustained, open-ended dialogue. Finally, we focus exclusively on mobile deployments and do not consider the Web-based versions of selected applications, as their capabilities may differ significantly due to variations in browser environments and access to device-level resources [22, 96]. We leave the measurement of additional applications, including Web-based counterparts, as promising directions for future work.

While GenAI calling is a nascent paradigm, it inherits and extends architectural principles from established real-time communication and content delivery systems. For instance, human-to-human calling applications such as Skype [79] have long addressed challenges of real-time audio streaming. Those challenges remain relevant in GenAI calling but are compounded by the need to *incorporate compute-intensive LLM inference into the communication loop*. Similarly, content distribution networks (CDNs) have laid the groundwork for scalable delivery of static and streaming content; however, GenAI calling demands *dynamic, per-session content generation that introduces new latency constraints*. Our study reveals both convergences and divergences from prior systems, offering novel insights into how emerging GenAI workloads are reshaping traditional expectations of latency, network usage, and architectural design for interactions between human users and AI agents.

We summarize our key findings as follows.

Finding-1. All six applications exhibit >1s TTFT, with the highest surpassing 8s (§4.4), well above the sub-second thresholds typically required for smooth, natural voice interaction [46]. Our measurements show that TTFT is jointly influenced by several factors, including the input modalities supported by the LLM backend, the geographic location of server infrastructure, the application's chosen transmission paradigms, and application-level optimizations such as incremental inference when handling long queries.

Finding-2. **Gemini** and the premium version of **ChatGPT** utilize multi-modal LLMs [21] that directly process audio input, whereas other applications rely on traditional text-only LLMs that require converting user speech to text before inference, and then converting the generated text back to speech for the response (§4.1). While multi-modal LLMs enable more seamless and expressive interactions by interpreting raw audio data, including non-verbal cues, they introduce additional processing overhead, adding >1s to the TTFT, compared to text-only LLMs.

Finding-3. In terms of network infrastructure (§4.2), the studied applications vary in the number of network sessions established during call setup. **Gemini** utilizes a single session and thus achieves the lowest setup time (∼1s), whereas **ChatGPT** shows non-deterministic behavior, opening 4–9 sessions to the same server, leading to setup times exceeding 3s. We also find that all applications deploy their servers primarily in major metropolitan areas, leading to a higher round-trip time (RTT) for users in less populated regions and potentially adding extra delay to the TTFT.

Finding-4. These applications adopt different design strategies for media delivery, resulting in varying protocol choices (§4.3). **ChatGPT**, **Instagram**, **Messenger**, and **WhatsApp** utilize UDP/RTP (Real-time Transport Protocol) for streaming-based transmission, delivering audio frame by frame in both uplink and downlink. In contrast, **Copilot** employs TCP with HTTP/2 (H2) to implement a batch-based strategy in uplink and downlink, where multiple audio frames are aggregated and transmitted in bursts. **Gemini** appears to be A/B testing with its uplink transmission strategy, alternating between streaming-based (QUIC with HTTP/3) and batch-based (TCP/H2) deliveries. In contrast, its downlink consistently relies on a batch-based transmission approach.

Finding-5. These transmission paradigms have a notable impact on both TTFT and bandwidth usage. For instance, streaming-based transmission enables shorter inter-arrival times than batch-based approaches, reducing TTFT by 150–250 ms. In contrast, batch-based transmission allows applications to retrieve the complete GenAI response quickly in the downlink, but at the cost of a substantial bandwidth surge, reaching up to 4 Mbps in the case of **Copilot**. On the other hand, streaming-based applications such as **Instagram**, **Messenger**, and **WhatsApp** maintain a low bandwidth footprint, typically operating at just 30–60 Kbps.

Finding-6. Through carefully crafted queries, we observe that **Instagram**, **Messenger**, and **WhatsApp** may employ incremental inference, progressively processing user audio as it is received (§4.4). For example, they may process speech in smaller steps as the user speaks, rather than waiting for the utterance to be completed before initiating inference. Our measurement results show that this optimization enables their TTFT for long queries to remain comparable to that for short queries. In contrast, other applications lack such optimizations, resulting in TTFT increases of more than 25%.

Finding-7. Our network disruption experiments (§4.5) reveal that **WhatsApp** supports fast failover with an average switching time of 1.2s, achieved by pre-establishing connections to three servers during call setup (§4.2). In contrast, **ChatGPT**, **Copilot**, and **Gemini** lack any failover mechanisms and lose functionality when the connection to the server becomes unavailable. Moreover, none of the evaluated applications implement adaptive bitrate streaming [42, 54, 93] to handle constrained bandwidth conditions. This limitation is particularly evident in **Copilot**, which fails to deliver smooth responses when downlink bandwidth falls below 1.5 Mbps. In comparison, other applications function reliably with bandwidth requirements of only a few hundred Kbps.

Our findings suggest that GenAI calling may necessitate a fundamentally new networking paradigm, one that moves beyond the one-size-fits-all strategies of traditional RTC or CDN infrastructures. Instead, it may require an intelligent integration of both approaches. This need arises from the inherently asymmetric nature of voice-based GenAI traffic. The uplink consists of real-time human speech, which is best served by RTC-style transmission. In contrast, while the downlink also transmits audio, the generation process of GenAI response is token-based; a single token can correspond to a substantial portion, or even the entirety, of the GenAI response. Consequently, the downlink exhibits a delivery pattern closer to on-demand streaming, for which CDN-style mechanisms provide a feasible complement to RTC-based approaches. In a recent paper [27], we further discuss design choices for GenAI calling motivated by the findings of this study. For instance, we explore QUIC-based transmission to support direction-dependent delivery, as well as dynamic switching between streaming-based and batch-based delivery in the downlink, depending on network conditions and the specific context of human–GenAI interactions.

2 Background

Human-to-GenAI Calling Applications consist of a client-side application (*e.g.,* a mobile app) and a remote LLM backend (*e.g.,* deployed in a cloud data center [92]). Figure 1 illustrates the workflow of a typical GenAI calling application, based on our measurement observations (§4.1). On the client side, the user speaks into a microphone, and the captured audio is transmitted over the Internet to the backend server. The server processes the input using either text-only LLMs or multi-modal LLMs [21]. When relying on a text-only LLM, the server first executes an automatic speech recognition (ASR) [119] module to extract scripts from audio. The textual query is processed by the text-based LLM to generate a response, which is then passed through a text-to-speech (TTS) [95] module to synthesize an audio response. In contrast, multi-modal LLMs can directly process raw audio and generate an audio response, without intermediate text conversion. Note that the LLM, ASR, and TTS modules may be deployed on different servers for scalability considerations.

Overview of Studied Applications. In this paper, we investigate six popular GenAI calling applications: **`ChatGPT`** [83] from OpenAI, **`Copilot`** [78] from Microsoft, **`Gemini`** [37] from Google, and **`Instagram`** [70], **`Messenger`** [72], and **`WhatsApp`** [73] from Meta. We choose these applications because they are developed by leading companies of LLMs [121]. Among them, **`ChatGPT`** and **`Gemini`** release paid tiers that provide access to more advanced models than their free versions. We refer to them as **`ChatGPT-P`**/**`ChatGPT-F`** and **`Gemini-P`**/**`Gemini-F`** *only* when their measurement results show distinguishable patterns. Thus, in total, our study covers eight distinct instances of these applications. In the following, we summarize the key features of the selected six applications.

- *OpenAI.* We evaluate **`ChatGPT-F`** and **`ChatGPT-P`** using the *GPT-4* [80] and *GPT-4o* [82] models, respectively, as they are the default deployments for these tiers and are optimized for fast response [84].
- *Microsoft*'s **`Copilot`** does not have a paid version. We evaluate its performance using the "*Quick response*" mode, which is described in the app as "*Best for everyday conversation*". The specific LLM backend used by **`Copilot`** is not disclosed.
- *Google.* We evaluate **`Gemini-F`** and **`Gemini-P`** using the *2.0 Flash* [36] and *2.5 Flash* [38] models, respectively, as they serve as the standard deployments for their respective tiers, tailored specifically to deliver low-latency responses [90].
- *Meta*'s three applications **`Instagram`**, **`Messenger`**, and **`WhatsApp`** employ the same LLaMA [107] model as the LLM backend. All of them are freely available and have no premium plan. Meta released a standalone application, *Meta AI*, on April 29, 2025 [71]. Based on our initial measurements, its network behavior closely resembles that of **`Messenger`**.

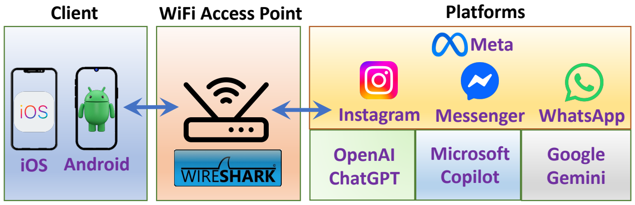

Figure 2: Measurement setup.

Audio Codec. Raw audio is a continuous analog signal that, in a typical voice call, can consume over 700 Kbps bandwidth [69]. To reduce this overhead, modern audio codecs segment the continuous waveform into discrete frames and apply efficient compression before transmission. For example, Opus [109], a widely adopted codec and the default one in WebRTC [108], supports frame durations ranging from 2.5 to 60 ms, with 20 ms being the default. It can aggregate multiple frames into a single packet. Additionally, Opus supports discontinuous transmission (DTX), which, when enabled, reduces bandwidth usage during silence or low-activity periods by encoding only one frame every 400 ms [109].

3 Testbed & Data Collection

Figure 2 shows our experimental setup. We conduct experiments on three smartphones: 1) Google Pixel 8a (Android), 2) OnePlus 10 Pro (Android), and 3) iPhone 14 Pro (iOS). We observe that device type and operating system have negligible impact on most of the measured metrics, as the majority of processing occurs on the server side (§4.1). Thus, unless otherwise specified, we report aggregated results across all three devices. We deploy the clients in nine locations across the western, middle, and eastern U.S., denoted as *Western-1/2/3*, *Middle-1/2/3*, and *Eastern-1/2/3*, respectively, when presenting location-specific results. For results unaffected by client geolocation, we report aggregated outcomes across all nine locations. The client devices connect via WiFi to an access point (AP) providing an average bandwidth exceeding 200 Mbps and a round-trip time (RTT) of ~10 ms between client and AP. We deploy Wireshark [113] on the AP to capture and analyze network traffic. We disable all background processes on client devices during testing and clear previous session data, if any.

Each experiment (*i.e.,* one call session) consists of multiple rounds of human–GenAI interaction, where each user query is randomly drawn from the dataset introduced below, and the duration of each call is at least 120s. At each location, for each device, and for each application, we conduct 20 experiments. We consider **`ChatGPT-P`**/**`ChatGPT-F`**, **`Gemini-P`**/**`Gemini-F`**, and **`Gemini-T`**/**`Gemini-Q`** (explained in §4.2), as distinct applications. Thus, we conduct 9 (locations) × 3 (devices) × 20 (experiments) × 10 (applications) = 5,400 calls. We also conduct additional calling sessions with specially designed queries to examine TTFT optimizations (§4.4) and with controlled network disruptions to evaluate application robustness (§4.5). For these calling sessions, conducted at the *Eastern-1* location, we perform 3 (devices) × 20 (experiments) × 10 (applications) = 600 calls. Thus, in total, our measurement study comprises 6,000 calls.

We primarily leverage the *Everyday Conversations for LLMs* dataset [33] for input queries. This dataset comprises a diverse set of multi-turn conversations on *everyday topics* and *elementary science questions*. They reflect two representative use cases of GenAI calling applications: *casual dialogue* and *knowledge retrieval*. We adopt this dataset for the majority of our experiments, except for workflow analysis (§4.1) and the investigation of TTFT optimizations (§4.4), where we use carefully crafted queries to isolate and/or examine specific behaviors. Each input query is converted into speech using ElevenLabs [32]. During each experiment (or *call*), the pre-recorded audio is replayed in a controlled, quiet environment as the user's input, ensuring consistent and reproducible interaction with the applications. At any given moment, either the user or the GenAI assistant is actively speaking. We also include a *mute* setting, where neither party speaks, and the user explicitly mutes the microphone by clicking the mute button in the app, except for **Gemini**, which does not offer this functionality.

We conducted all experiments between February and May 2025 and collected the following performance metrics.

- *Throughput.* We measure both uplink (from the user to GenAI) and downlink throughput during three phases: (i) user speaking, (ii) GenAI responding, and (iii) mute. Throughput is jointly influenced by inter-arrival times and packet size, which are introduced next.
- *Inter-arrival Time.* It refers to the time interval between consecutive packets. In streaming-based transmission, audio is typically encoded and sent frame-by-frame at regular intervals, for example, every 20 ms by default in Opus [109] (§2). However, we also observe batch-based delivery, where multiple audio frames are bundled and transmitted together as a batch. In such cases, we define the inter-arrival time as the time interval between successive batches. Inter-arrival time provides insight into how applications schedule audio delivery, which can influence how quickly audio data is received by either the user or the GenAI model.
- *Packet Size.* It offers valuable insights into how applications deliver audio data. Small packets typically indicate streaming-based delivery, as each audio frame itself is relatively small. Consistently large packets suggest batch-based transmission, where multiple audio frames are bundled together.
- *Time to First Token (TTFT).* Modern LLMs generate responses in a token-by-token manner [107]. We consider the time at which the GenAI calling application begins its response as the generation of the first token. We define TTFT as the time interval from when the user finishes speaking to when the GenAI starts responding. Note that most studies on LLM performance define TTFT as the time interval from when the inference request is received (often already processed/tokenized) to when the first output token is generated by the LLM [60, 118]. Thus, they focus on model-side latency and exclude client-side and networking factors such as audio capture and data transmission. In contrast, our definition of TTFT measures the user-perceived, end-to-end latency, representing responsiveness and conversational flow in real-time human–GenAI interactions.
- *Call Setup Time.* Upon initiating a call, there is a delay before user input can be accepted. We define this duration as the call setup time, which reflects the time required to initialize the voice-interaction pipeline and prepare the backend services. It measures the cold start time of an application, which is a key factor influencing quality of experience (QoE) in RTC [47]. Moreover, this delay adds to the waiting time for the user's first query, as the system must first establish the interaction pipeline before processing user input.

4 Measurement Results

4.1 Workflow Analysis

We first examine the workflow of typical GenAI calling applications by answering two key questions: 1) *What is the input modality of their underlying LLM backends?*, and 2) *What type of data is transmitted between the client and server?*

Input Modality of LLMs. With the source of user input data being audio, the underlying LLM backend can process this data through either a text-only LLM or a multi-modal LLM (§2). The key distinction between the two approaches lies in whether the model can capture and directly interpret information inherent in audio signals, such as tone or emotion, which would otherwise be lost during conversion to text. To determine what type of model each application employs, we design two sets of experiments by generating audio clips with: 1) sounds from various musical instruments (*e.g.,* guitar and drum), and 2) human voices expressing different emotions (*e.g.,* happiness and sadness). For example, to test instrument recognition, we use prompts such as: "*Listen to the sound that will be played and identify whether the musical instrument is a guitar or drum [sounds].*"

We observe that only **ChatGPT-P** and **Gemini** wait until the audio has fully played before responding and (correctly) identifying the instrument. In contrast, other applications respond immediately after the spoken portion of the prompt (*e.g.,* "guitar or drum"), with typical responses such as "*I am not able to identify ...*", even while the subsequent instrument sounds are still playing. This indicates that the backend model relies on ASR to transcribe spoken input, without the ability to interpret non-verbal audio content. Since instrument sounds contain no transcribable text, these systems treat the last transcribed word as the end of the user query.

For emotion recognition, **ChatGPT-P** and **Gemini** can accurately classify the emotional tone (*e.g.,* happy *vs.* sad) of spoken inputs, while other applications fail to do so. These results suggest that **ChatGPT-P** and **Gemini** are powered by multi-modal LLMs capable of audio understanding, whereas the remaining applications rely on text-only LLMs that cannot directly process/interpret audio signals.

What is Being Delivered? We next investigate the type of data transmitted between the client and the server. This cannot be obtained directly from the network traffic as all data is encrypted (§4.2). Since no intermediate conversion to text is required for applications using multi-modal LLMs, the transmitted data is audio in both uplink and downlink. In contrast, for applications using text-only LLMs, the placement of ASR and TTS modules determines the type of transmitted data: if ASR and TTS are on the server, the uplink/downlink data will be audio; if these components are on the client, the transmitted data would, instead, be text. Further, if ASR and/or TTS are locally executed in the client, we would expect an increase in CPU utilization during user speech (for ASR) and

Application	Protocols	# of Connections		Ent Svr Owner	Western-1	Middle-1	Middle-2	Middle-3	Eastern-1
		Setup	Calling						
Instagram	UDP/RTP	2	1 (2)	Meta	12.0/2.5	59.0/14	16.5/4.2	31.2/5.2	11.7/2.6
Messenger	UDP/RTP	2	1 (2)	Meta	11.8/2.7	58.7/14	16.8/4.1	31.7/5.4	11.9/2.8
WhatsApp	UDP/RTP	3	1 (2)	Meta	16.5/4.2	53.0/17	20.7/6.9	28.4/4.4	13.1/3.5
ChatGPT	UDP/RTP	4–9	4–6 (4)	Oracle	24.7/4.6	69.2/16	13.8/2.3	37.4/5.7	23.4/6.7
Copilot	TCP/H2	2	2	Akamai	15.1/5.6	51.8/15	11.1/0.7	33.1/5.3	13.1/4.7
Gemini	TCP/H2 QUIC/H3	1	1	Google	13.0/4.6	56.0/17	11.5/2.3	26.4/6.1	12.8/4.9

Table 1: Summary of (1) the transport/application layer protocols for media data exchange (H2 and H3 stand for HTTP/2 and HTTP/3, respectively), (2) the number of connections during call setup and active calling (numbers in parentheses for RTP-based applications denote the number of RTP streams for media data transmission), (3) the entry server owner for each application, and (4) the round-trip time (RTT) measured between the client and the entry server from five different client locations across the U.S., shown as average/standard deviation in milliseconds.

GenAI response (for TTS). Thus, resource usage on mobile devices is a key indicator of where ASR/TTS runs.

Our measurements show that, regardless of the application, the client-side CPU usage remains stable during interactions between users and GenAI, indicating that both ASR and TTS are handled on the server side. To further validate this, we execute Android's ASR [7] and TTS [8] APIs on Google Pixel 8a and observe that ASR/TTS causes a CPU utilization increase of 156/166%, on average.

Summary: With the above analysis, we can reconstruct the workflow of mainstream human-to-GenAI calling applications, as shown in Figure 1. We observe that ChatGPT-P and Gemini have adopted multi-modal LLMs that can directly process audio input from users. This design enables more expressive and targeted responses in specific scenarios, such as identifying sounds or recognizing emotional tone in users' speech, compared to text-only approaches. Nonetheless, as we will show in §4.4, this design choice may extend the TFFT, exhibit a trade-off between interaction richness and user-experienced latency.

4.2 Network Infrastructure

We next examine the network infrastructure supporting these applications. Our analysis is structured into three parts: 1) their transport and application layer protocols, 2) the number of network connections established during call setup and maintained during active calling, and 3) the characteristics of the *entry server*, defined as the IP address that directly exchanges media data with the client. This server often functions as a relay node, acting as an entry or exit point for traffic to or from the provider's backend infrastructure. Importantly, the entry server is not necessarily responsible for tasks such as LLM inference. We believe this definition is broadly applicable to measurement studies of black-box commercial systems, where the internal service architecture is typically opaque. To further validate this, we cross-examine the traces collected from our prior measurements of social virtual reality platforms [26] and find that the same entry-server IPs observed for Instagram and Messenger also appeared in traffic from Meta's Horizon Worlds.

Network Protocols. Table 1 shows the transport and application layer protocols used by these applications. ChatGPT, Instagram, Messenger, and WhatsApp utilize UDP and RTP, widely adopted for real-time audio communication [17, 25]. These applications also incorporate Session Traversal Utilities for NAT (STUN) for network address translation (NAT) traversal and Datagram Transport Layer Security (DTLS) to secure media exchanges. Copilot uses TCP with HTTP/2 for media transport and TLS for securing data exchange. Gemini supports two distinct setups. The first one uses TCP with HTTP/2 (H2) and TLS for encryption, while the second one uses QUIC with HTTP/3 (H3) and DTLS. In the following, we refer to them as Gemini-T and Gemini-Q, respectively, *only* when their measurement results show distinguishable patterns. These configurations appear to be deployed via A/B testing, as we observe Gemini alternates between them across different days without requiring client-side updates.

Number of Network Connections. We analyze the number of network connections each application establishes during call setup and maintains during active calling. Connections are identified by the 5-tuple: source IP address, source port, destination IP address, destination port, and transport protocol [13]. For RTP-based applications, we further distinguish individual media streams using the SSRC (synchronization source) field, which uniquely labels RTP streams when they share the same IP and port pair [99]. Table 1 summarizes connection and RTP stream counts (where applicable).

- WhatsApp initiates three connections via STUN messages with three servers during call setup. Following connection establishment, the client proceeds to exchange media data with only one of these servers. Based on the consistent naming convention and observed behavior, we infer that all candidates serve as potential entry servers, which enables fast failover when the link to the entry server in use becomes unavailable (§4.5). During an active call, the client and the entry server maintain two RTP streams for uplink and downlink, both using the same IP/port pair. In parallel, the client periodically exchanges STUN messages with the entry server every 3s over the same connection used for RTP, avoiding the creation of an extra network session.
- Instagram and Messenger initially exchange STUN messages with both a dedicated STUN server and a designated entry server. All media and signaling traffic is routed through the entry server. The connection behavior with the entry server mirrors that of WhatsApp (*i.e.,* one connection for two streams), except that STUN messages are exchanged every 8s, rather than every 3s.

- `ChatGPT` presents a non-deterministic connection setup pattern. After a call is initiated, the client exchanges STUN messages over 4–9 client/server port pairs with the same entry server. These exchanges occur *sequentially* over a span of 357 ms on average (SD: 112 ms). During active calling, 4–6 connections remain alive, each used to exchange periodic STUN messages every 2–3s. One of these connections also carries media data via RTP, with one uplink and three downlink streams. Among the downlink streams, only one carries the actual audio data, as evidenced by its high throughput during GenAI responses and low throughput when the user speaks. The remaining two streams exhibit identical packet transmission patterns and maintain consistently low data usage (∼5 Kbps), even during GenAI responses. These observations suggest that the two additional downlink streams are likely redundant and may be reserved for other purposes.
- `Copilot` maintains two TCP connections. Media data is transmitted over a single connection to its entry server. A second connection to `mobile.events.data.microsoft.com`, as indicated in the Server Name Indication (SNI) from the TLS handshake, is used for data telemetry [41]. This telemetry connection is only active when the user speaks.
- `Gemini` multiplexes all traffic through a single connection to its entry server, regardless of whether the client uses QUIC or TCP, and regardless of the free or paid tier.

Entry Server. We next examine the entry servers used by each application to exchange media data with the client. The owners of these entry servers are summarized in Table 1. `Gemini`, `Instagram`, `Messenger`, and `WhatsApp` rely on servers operated by their respective companies. `ChatGPT` utilizes entry servers hosted by Oracle. For `Copilot`, the entry server is hosted by Akamai, but the SNI field reveals the server name as `copilot.microsoft.com`. This is different from other Microsoft applications, such as AltspaceVR [76] and Teams [77], which rely on Microsoft's infrastructure for entry servers, as shown by prior measurement studies [25, 26]. This decision is likely to provide more entry servers closer to end-users across geolocations, which we examine next.

Geolocations. We study the geographic distribution of entry servers by deploying clients across the U.S. (§3). We measure the round-trip time (RTT) between the client and its assigned entry server using ICMP pings. With clients in different locations, we also confirm that none of the evaluated applications use *anycast* routing [55, 67] by following the approach adopted in prior work [26]. This allows us to reliably infer the geolocation of entry servers from their IP addresses. For geolocation lookup, we use *MaxMind* [66] and *ipinfo.io* [43]. Both tools yield consistent results for all servers in terms of location.

Table 1 presents RTT measurements for one representative location on the west and east coasts, respectively, as well as three locations in the middle U.S. We observe that RTTs from other west and east coast locations are similar to the locations reported. Across all applications, we find a consistent deployment strategy, with entry servers placed in major metropolitan areas. This results in low RTTs for clients on the coasts, where more entry servers are available. In contrast, clients in certain central regions may experience high RTTs. For example, the *Middle-1* location, despite being in a mid-sized city, is consistently assigned servers over 1,000 miles away, resulting in RTTs exceeding 50 ms, on average, for all applications. This suggests that commercial GenAI calling services are still in the early stages of deployment and may not yet be fully optimized for users in less-populated regions.

We next conduct per-application analysis. While `WhatsApp` pre-establishes connections to three candidate entry servers distributed across various regions (§4.2), its RTTs remain comparable to other applications. For instance, it achieves 53 ms RTT from *Western-1*, which is the second lowest across six applications. This suggests that `WhatsApp` benefits from a broader and more distributed server infrastructure. `ChatGPT` exhibits higher RTTs when the client is located on the west coast, east coast, or in *Middle-1*, compared to other applications, for example, over 10 ms higher than `Instagram` and `Messenger`. However, in these cases, `ChatGPT`'s entry server is located in the same metropolitan areas on the west or east coast as those of the other two. This observation may suggest room for improvement in the network performance and server capacity of Oracle's infrastructure supporting `ChatGPT` in these regions. We also observe that `Copilot`, hosted on Akamai's infrastructure, achieves comparable RTTs to other applications across the U.S.

Summary: Our analysis reveals that the evaluated applications adopt diverse network infrastructures. For instance, UDP, TCP, and QUIC, three of the most widely used transport protocols for RTC [48], are all represented among these applications. As we discuss next in §4.3, these protocol choices are closely tied to the underlying transmission paradigms adopted by each application. We also observe distinct differences in how applications manage network connections during the call. For example, `Gemini` stands out for its lightweight design, maintaining a single connection throughout the call, which contributes to its short call setup time (§4.4). Meanwhile, `WhatsApp` pre-establishes connections to multiple entry server candidates, enabling fast failover when the connection is disrupted during calling (§4.5).

4.3 Transmission Behavior

In this section, we analyze the transmission paradigms employed by the studied applications.

Transmission Patterns during Calling. Figure 3 presents the uplink and downlink throughput of each application during a call. `ChatGPT`, `Instagram`, `Messenger`, and `WhatsApp` exhibit a similar bi-directional streaming pattern, where bandwidth usage on the uplink or downlink corresponds closely to when the user is speaking or GenAI is responding. During idle periods, including when the user or GenAI is silent or when the session enters the mute stage, these applications maintain minimal background traffic. An exception is `ChatGPT`, which shows slightly elevated uplink throughput when GenAI is responding. We also observe that `WhatsApp` demonstrates two distinct transmission behaviors during the mute stage, switching modes ∼10s after mute is enabled.

In contrast, `Copilot` and `Gemini` continuously transmit data in the uplink regardless of whether the user is speaking or not. For the downlink, both applications display bursty traffic when GenAI begins responding, after which the transmission quickly ceases, even if GenAI continues creating the response. This suggests that the burst is caused by downloading the entire response up front, while playback proceeds concurrently. Additionally, both `Copilot`

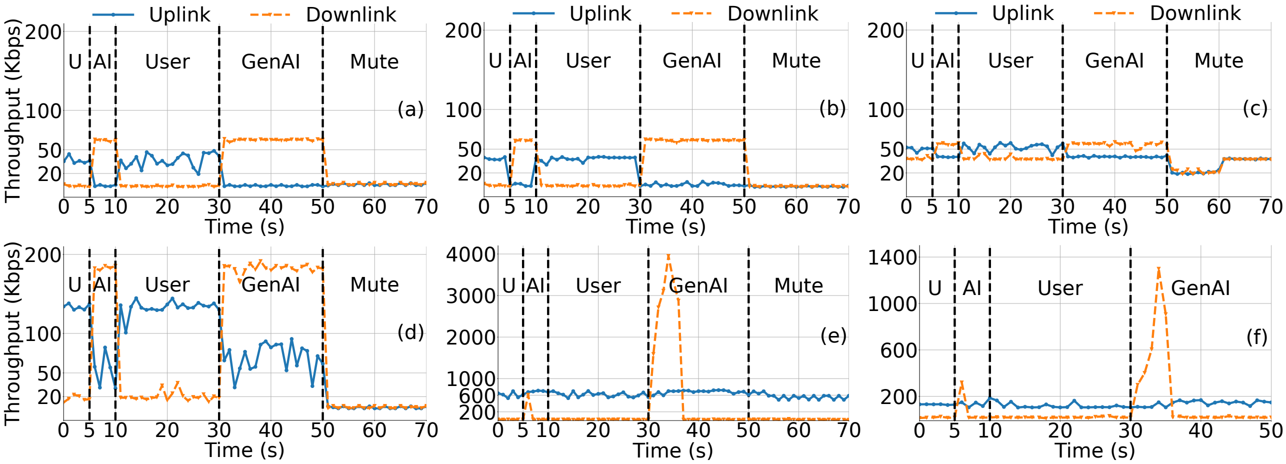

Figure 3: The throughput of (a) Instagram, (b) Messenger, (c) WhatsApp, (d) ChatGPT, (e) Copilot, and (f) Gemini-Q (QUIC) during a calling session. Gemini-T (TCP) exhibits a similar pattern to Gemini-Q. The session consists of five parts: user speaking (0–5s), GenAI responding (5–10s), user speaking again (10–30s), GenAI responding again (30–50s), and finally, a mute stage (50–70s). Gemini does not support mute functionality. Note that the scales for (e) and (f) are different from the others.

and **Gemini** exhibit two distinct throughput modes: a *low-throughput mode* for *short responses* and a *high-throughput mode* for *long responses*. Our experiments finds that the threshold between these modes is ~5s of response length.

We next perform a detailed analysis of the transmission behaviors exhibited by these applications.

Uplink. Figure 4 shows the throughput, inter-arrival time, and packet size distributions during: (1) user speaking, (2) GenAI responding, and (3) user mute (§3) for the uplink. The box plot shows the 95th, 75th, 25th, and 5th percentiles, median, and mean (blue dots). In the following analysis of the uplink, we use (a)–(i) to denote the subfigures in Figure 4.

User Speaking (a)–(c): We begin by focusing on the user speaking phase, where uplink traffic dominates bandwidth usage (Figure 3). Based on the inter-arrival time (b) and packet size (c) distribution, we can distinguish two transmission approaches: *streaming-based*, where audio is delivered frame-by-frame, and *batch-based*, where multiple frames are grouped and sent together. **ChatGPT**, **Gemini-Q**, **Instagram**, **Messenger**, and **WhatsApp** adopt streaming-based transmission, characterized by short inter-arrival times (<60 ms) and small packet sizes (*e.g.,* over 80% of packets are smaller than 500 bytes). In contrast, **Copilot** and **Gemini-T** adopt batch-based delivery, with evidence of packet sizes clustering near the 1500-byte MTU (maximum transmission unit) for more than 60% of packets. This batching behavior also introduces larger inter-arrival times, for instance, over 40% of packets in **Gemini-T** exhibiting inter-arrival times exceeding 300 ms.

Streaming-based applications maintain throughput below 150 Kbps. Our later examination finds that **ChatGPT**, **Instagram**, **Messenger**, and **WhatsApp** may utilize the Opus codec [109]. Therefore, their throughput variations are likely caused by different codec configurations. Considering **ChatGPT** as the baseline, it adheres to Opus's default 20 ms inter-arrival time (§2). **Instagram** and **Messenger** reduce throughput by increasing the inter-arrival time to ~60 ms. **WhatsApp**, on the other hand, largely retains the 20 ms inter-arrival time but uses smaller packet sizes. They ultimately achieve uplink throughput in the range of 36–52 Kbps, on average, which is more than 80 Kbps lower than that of **ChatGPT**, likely at the cost of reduced audio quality (given they use the same codec). For batch-based transmission, **Gemini-T** achieves even lower throughput than **ChatGPT**, primarily due to its larger inter-arrival times, at the cost of increased TTFT (§4.4). **Copilot** attempts to maintain moderate inter-arrival times (100–180 ms), but this, coupled with its large packet size, leads to a significantly higher uplink throughput, exceeding 600 Kbps on average, over 10× higher than Meta's applications.

GenAI Responding (d)–(f): During the GenAI response phase, when the user is silent and the uplink traffic is expected to resemble the mute stage (g), we observe that only **Instagram** and **Messenger** achieve this behavior. Specifically, they shift half of their packets to a larger inter-arrival time of ~360 ms and reduce their packet sizes (f) compared to the user speaking phase (c). **ChatGPT** exhibits a similar 360 ms inter-arrival time, resulting in its uplink throughput during GenAI response (d) being roughly half of that during user speaking (a).

In contrast, **Copilot** and **Gemini-Q/T** exhibit similar, or even higher, uplink throughput during GenAI response than during user speaking. Given the comparable inter-arrival times and packet sizes observed in both phases, this behavior may indicate that the applications continue capturing and transmitting background audio even when the user is silent, raising potential privacy concerns. Another potential explanation is that these applications transmit dummy packets to maintain session activity rather than actual audio content. The root cause of this phenomenon warrants further investigation to determine whether it indeed poses a privacy risk.

Mute (g)–(i): During the mute stage, **ChatGPT**, **Instagram**, and **Messenger** effectively reduce uplink bandwidth consumption to <10 kbps. This is achieved by shifting half of the packets to a ~400 ms inter-arrival time (h), likely enabling the DTX mode of the Opus

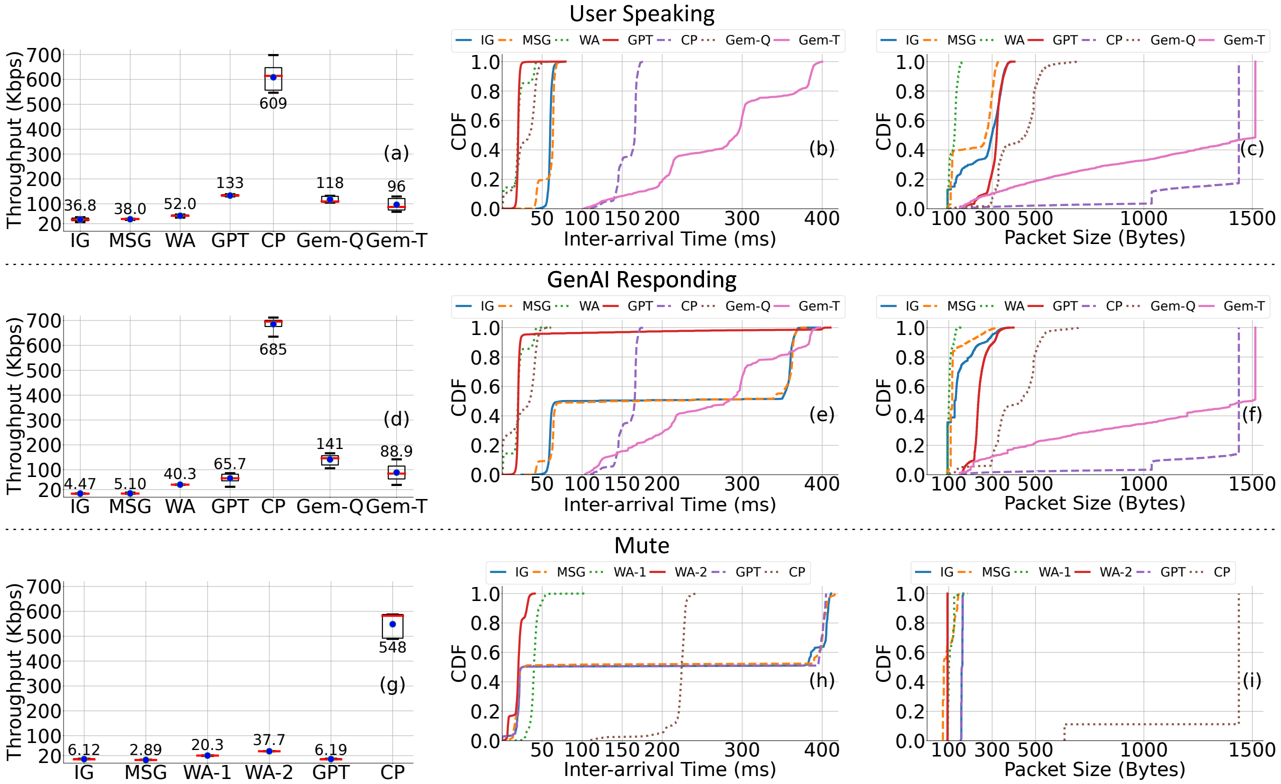

Figure 4: The throughput, inter-arrival time, and packet size of uplink (from left to right) when (1) user speaking (top), (2) GenAI responding (middle), and (3) mute (bottom). WA-1/2 in the mute stage refers to the first/second stage of `WhatsApp`.

codec (§2) that they may employ, while also reducing packet sizes to fewer than 100 bytes (i). Different from other applications, **`WhatsApp`** does not shift to 400-ms inter-arrival times after mute is activated. About 10s after enabling mute, **`WhatsApp`** transitions to a secondary transmission mode characterized by the continuous transmission of small 94-byte packets with short inter-arrival times (80% <20 ms), resulting in uplink bandwidth usage exceeding 35 Kbps (g). The rationale behind this design choice remains unclear. We also observe that **`Copilot`** continues to consume over 500 Kbps of uplink bandwidth even when muted, suggesting that it lacks optimizations to reduce bandwidth usage in low-activity periods.

Downlink. Figure 5 shows the downlink throughput and packet size distributions when GenAI is responding. We do not report inter-arrival times because for batch-based methods, data is delivered in a short burst, making the inter-arrival time less informative. We observe that batch-based approaches consume significantly higher bandwidth than streaming-based ones, with peak throughput reaching ~4 Mbps in the *high-throughput mode* of **`Copilot`**, as shown in Figure 5(a). This behavior reflects a download-while-playback strategy. This batch-based delivery is characterized by large packet sizes (*e.g.*, 1514/1292 bytes for **`Copilot`**/**`Gemini`**), as shown in Figure 5(b).

By comparing Figures 4(a) and 5(a), we also find that streaming-based applications have a higher downlink throughput during GenAI responding than their uplink during user speaking. This increase is achieved through different strategies. **`Instagram`** and **`Messenger`** increase packet sending rates (not shown), while **`ChatGPT`** increases packet sizes. This finding suggests that, even when adopting the streaming-based paradigm in both uplink and downlink, these applications apply different rate control strategies, likely reflecting different design goals for the delivery of user input versus GenAI response.

A key factor underlying the different choices for uplink and downlink deliveries is the inherently asymmetric nature of uplink and downlink behavior in GenAI calling applications. The uplink carries live human speech that must be streamed in real time, making it well-suited to RTC protocols such as UDP/RTP for low-latency delivery. In contrast, the downlink transmits GenAI responses, which are often generated in full before playback and can be efficiently delivered using batch-based transmission over protocols such as TCP/H2. This asymmetry mirrors the classic RTC and CDN paradigms, respectively. However, it also presents protocol design challenges, as both UDP/RTP and TCP/H2 typically enforce symmetric transmission strategies. **`Gemini`** appears to address this tension by adopting QUIC/H3, which enables streaming-like behavior for the uplink while supporting efficient batch delivery in the downlink.

Summary: We find that the applications studied adopt different strategies for audio delivery, resulting in distinct data exchange patterns across different stages of a call. Moreover, even among applications utilizing similar transmission paradigms, we observe

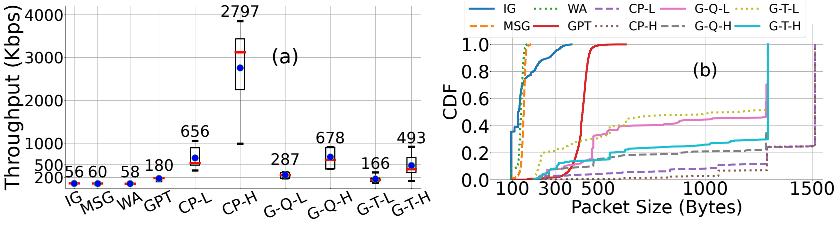

Figure 5: Throughput (a) and packet size (b) in downlink during GenAI responding. L/H: low/high throughput mode.

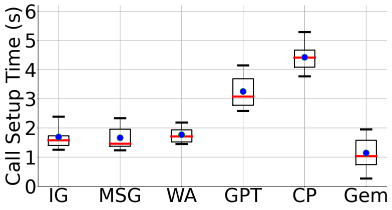

Figure 6: Call setup time of six applications.

variations in design details such as inter-arrival times and packet sizes. This suggests that identifying the optimal transmission configurations for GenAI calling remains an open problem. Moreover, we reveal potential inefficiencies in the current system designs. For example, `Copilot` and `Gemini` continuously transmit data in the uplink regardless of whether the user is actively speaking, indicating further optimization opportunities.

4.4 Latency Analysis

In this section, we analyze and compare the latency performance of the studied applications. We evaluate the call setup time of these applications, their TTFT across different queries and client locations, and potential application-level optimizations to reduce TTFT. The results reported in this section are based on measurements conducted at the *Eastern-1* location, except for the experiments that explicitly examine TTFT across client locations.

Call Setup Time. We begin by measuring the call setup time (*i.e.,* cold start latency) of each application. To enable accurate and large-scale measurements, we develop an automated testing tool, detailed in Appendix B. Figure 6 summarizes the results across six applications. We observe that their call setup time can vary from as low as 0.25s (5th percentile of `Gemini`) to as high as 5.34s (95th percentile of `Copilot`). `Gemini` achieves the lowest setup time, averaging ~1s. This efficiency is likely due to its lightweight network behavior during call initiation, requiring the establishment of only a single network connection (§4.2). `Copilot` exhibits the longest call setup time, >4s on average. Our experiments on network disruption in §4.5 reveal that this is due to the extended time required to establish network connections to its server infrastructure. Notably, after the user initiates a call, `Copilot` presents a greeting message before accepting any user input, a behavior not observed in other applications. This design likely serves to hide the perceived call setup time by occupying the user's attention while the system completes its initialization. `ChatGPT` has the second-highest call setup time, >3s on average. This delay may stem from its *sequential* establishment of 4–9 network connections to the entry server (§4.2). By contrast, while `WhatsApp` also builds connections to its three entry server candidates, it is done in parallel, avoiding significant setup delays, which is evidenced by its average call setup time of ~2s, similar to `Instagram` and `Messenger`.

A simple yet effective optimization strategy is to overlap call setup with user input capture, allowing the user to start speaking while the session is still being established. While this approach does not essentially reduce the call setup time, it decreases the perceived waiting time by the user. However, none of the evaluated applications appear to adopt this strategy. Instead, we observe that during setup, users are typically presented with a transitional interface, such as a blank screen or a greeting prompt used by `Copilot`, during which user input cannot be accepted.

TTFT across Queries. Figure 7 presents the TTFT for six applications using input queries drawn from *everyday topics* and *elementary science questions* (§3) with experiments conducted in *Eastern-1* (§4.2). We find that all applications exhibit similar TTFT across both query types. More importantly, the measured TTFT remains considerably high across all applications, with the 5th percentile exceeding 1s. The worst-case TFFT can reach up to 5s, as observed at the 95th percentile for `ChatGPT-F`, falling short of the seamless, sub-second latency expected in natural voice conversations [46]. Additionally, we observe no evidence of response caching. Even when users repeat identical queries within the same session, these applications consistently reprocess the input, resulting in second-level TTFT for repeated requests.

The input modalities of LLM backends lead to notable impacts on TFFT. Specifically, applications leveraging multi-modal LLMs that directly process audio input, including `ChatGPT-P` and `Gemini`, have higher TTFT, averaging over 3s. In contrast, applications relying on text-only LLMs, including `Copilot`, `Instagram`, `Messenger`, and `WhatsApp`, achieve lower TTFT below 2s, on average. We exclude `ChatGPT-F` from this comparison, as our later investigation of TTFT optimizations reveals that OpenAI appears to assign a weaker model to it compared to `ChatGPT-P`, making direct comparisons less meaningful since model complexity, rather than input modality, may dominate TTFT in this case.

We further find that the transmission paradigm influences TTFT. For example, `Gemini-Q` adopts streaming-based uplink transmission, resulting in a short inter-arrival time of ~40 ms at the 50th percentile; while `Gemini-T` uses batch-based delivery with a longer inter-arrival time of ~300 ms at the 50th percentile (Figure 4). This difference leads to a measurable latency gap, with `Gemini-T` consistently showing 150–250 ms higher TTFT than `Gemini-Q` for the same query and service tier. For example, for the paid tiers, the average TTFT for `Gemini-T` on the *everyday topic* is 3.45s, 250 ms higher than the 3.20s TTFT observed for `Gemini-Q`.

TTFT across Client Locations. Figure 8 presents the TTFT measured when the client is located in *Western-1* and *Middle-1*. The results are for queries from *everyday topics*, and those from *elementary science questions* show similar trends. Comparing Figures 7 and 8, we can observe the impact of entry server geolocation on

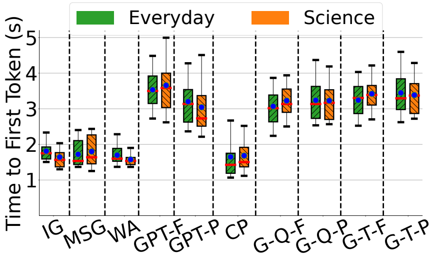

Figure 7: Time to first token (TTFT) with input queries from `everyday topics` and `elementary science questions` of *Everyday Conversations for LLMs* dataset [33]. F/P denotes free/paid tier.

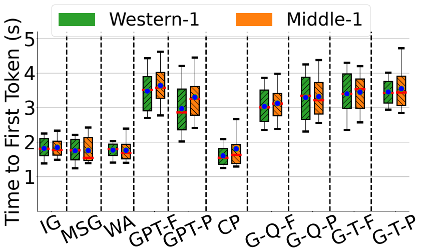

Figure 8: TTFT for six applications with input queries from `everyday topics` when the client is in *Western-1* and *Middle-1*. Results from `elementary science questions` are similar.

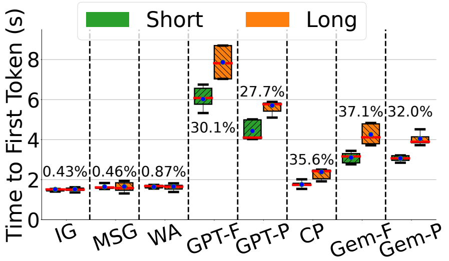

Figure 9: TTFT for short and long queries designed to reveal potential optimizations. The numbers shown for each application indicate the percentage (%) increase in the average TTFT from short to long queries.

TTFT. Specifically, the TTFT results for *Western-1* and *Eastern-1* are comparable across all applications, consistent with their similar RTTs to the entry servers in these regions (§4.2). In contrast, measurements from *Middle-1* show increased TTFT, ranging from 30 to 300 ms on average, potentially revealing the impact of TTFT introduced by longer physical distance to the entry server (§4.2). Note that the increase in RTT between the client and the entry server is only ~40 ms (Table 1). Any additional increase in TTFT beyond that may be attributed to potential changes in the backend triggered by client relocation, which are not visible to us.

The latency incurred in the network may account for only a small fraction of the total TTFT. For instance, as previously reported, for **`Gemini`**, the latency introduced by transmission mechanisms is 150–250 ms, while its TTFT is ~3s. Thus, the dominant contributor to TTFT likely lies in LLM inference, which is influenced by factors such as input length and model complexity [60]. To reduce model complexity, model compression [62] is a promising solution. Moreover, performing system-level optimizations, such as key-value (KV) caching [60, 118], can contribute to TTFT reduction.

TTFT Optimizations. The results above reveal that the TTFT for these applications can exceed 3s, on average, even for input queries from simple everyday conversations. Naturally, more complex queries (*e.g.,* with longer user inputs) are expected to incur even higher TTFT [60]. This raises the question of whether these applications implement optimizations to reduce TTFT. Considering the LLM backend receives a continuous stream of speech in real time, one potential optimization is *incremental inference* (*e.g.,* starting to process partial input as it is received and repeating the inference multiple times during the user's speech) to reduce TTFT.

To explore whether this optimization is adopted, we design a controlled experiment with two query types: a *short query* serving as the baseline and a *long query* intended to reveal such optimizations if present. Both queries ask the applications to complete the same task, which, in our case, is to write an essay. The long query is structured to progressively provide additional context and requirements after stating initially the same prompt as the short query. If no optimization exists, we expect the TTFT for the long query to be higher than that of the short one [60]. Conversely, if incremental inference is applied, it can reduce the final workload by enabling a shorter TTFT compared to processing the full query at once, potentially achieving a latency close to that of the short query. We provide the details of the short query and long query in Appendix C.

Figure 9 presents the TTFT results of six applications for both queries. We observe that for **`Instagram`**, **`Messenger`**, and **`WhatsApp`**, the TTFT of the long query increases by less than 1% on average compared to that of the short query, consistently remaining under 2s. In contrast, other applications exhibit TTFT increases exceeding 25% on average, with the most pronounced result observed in **`ChatGPT-F`**, reaching an average TTFT of 8s, >6s longer than Meta's three applications. Given that **`Instagram`**, **`Messenger`**, and **`WhatsApp`** rely on the same LLM backend (§2), these results indicate that Meta may implement incremental inference optimizations that process user input progressively during the speaking phase, reducing TTFT for long queries. While this technique lowers TTFT, it incurs additional computational and financial costs [20], since inference will be executed multiple times for a single user query.

We also observe that **`ChatGPT-F`** exhibits higher TTFT compared to **`ChatGPT-P`** by ~2s on average for both the short and long queries. This suggests that OpenAI may provision a less capable LLM model for the free tier. On the other hand, across the TTFT measurements in Figures 7, 8, and 9, we observe no significant difference between **`Gemini-F`** and **`Gemini-P`**.

Summary: Our measurements reveal that both call setup time and TTFT of evaluated applications remain at the second level, highlighting the need for further optimization. For reducing call setup time, **`Gemini`** demonstrates a promising approach by minimizing the number of network connections required to establish the session. Regarding TTFT, our results indicate that network-level factors such as uplink transmission strategy (*e.g.,* streaming vs. batching) and server proximity can provide measurable improvements. Our measurements further show that Meta may mitigate the impact of long inputs through incremental inference, which reduces processing delay by handling partial inputs progressively.

4.5 Network Disruptions

In this section, we evaluate the robustness of GenAI calling applications under adverse network conditions. Specifically, we investigate their behavior under 1) complete connection loss and 2) constrained bandwidth scenarios. We select queries from the

Application	Backup Entry Server		Recovery (s)
	Number	Failover (s)	
Instagram	1	4.03/0.44	1.18/0.12
Messenger	1	3.98/0.38	1.21/0.15
WhatsApp	2	1.19/0.17	1.10/0.11
ChatGPT	✗		✗
Copilot	✗		1.72/0.42 (4.23/1.12)*
Gemini	✗		✗

Table 2: Summary of application behaviors under connection blocking and unblocking. We first block the connection between the client and the entry server to assess whether the application supports failover to a backup server, and if so, report the number of backup servers and failover time. We then unblock the connection to measure the time required to restore functionality. *: Upon recovery, Copilot does not resume the original conversation but instead initiates a new session; values outside (inside) parentheses indicate recovery to a previously established (newly initiated) entry server, respectively. All time-related results are reported as average/standard deviation.

Everyday Conversations for LLMs dataset [33] that trigger the *long response*, which activate the *high-throughput mode* observed in **Copilot** and **Gemini** (§4.3).

Connection Blocking. We use `iptables` [97] to block uplink and downlink traffic between the client and the entry server while the GenAI is actively responding. We monitor whether the application attempts to fail over to a backup server and, if so, continue blocking subsequent connections until the application ceases to function. After that, we unblock the connection to assess whether the application can be restored. We define *failover time* as the duration from blocking the connection to the resumption of smooth, uninterrupted GenAI responses, and *recovery time* as the duration from unblocking the connection to the resumption of smooth, uninterrupted GenAI responses.

Bandwidth Throttling. We apply bandwidth constraints on uplink and downlink traffic with `tc-netem` [50]. For each application and each direction of traffic, we gradually reduce the available bandwidth from the 95th percentile to the 5th percentile, with steps of 5 percentile intervals, based on the throughput distributions shown in Figures 4(a) and 5(a).

Connection Blocking. Table 2 summarizes the reaction of each application after the connection is blocked/unblocked.

- **WhatsApp** demonstrates fast failover by switching to one of its four other candidate entry servers with pre-established connections (§4.2) when the current entry server is blocked. This process continues when other candidate servers are blocked until none are available, at which point the application stops functioning. Because these connections are already established prior to failure, the *failover time* is relatively short, with 1.19s on average. Once all candidate servers are blocked, functionality can only be restored by unblocking either the initially selected server or the last used backup server. **WhatsApp** will not attempt to establish new connections with other servers beyond the initial three. Note that when functionality is restored in this manner, **WhatsApp** preserves the ongoing conversation context, allowing the GenAI to resume its response without restarting the calling session. The *recovery time* in these cases is also short, averaging 1.10s.
- **Instagram** and **Messenger** rely on a Traversal Using Relays around NAT (TURN) [63] server from Meta to maintain functionality after their entry server is blocked, a typical usage of the TURN server [63]. However, since this involves establishing a new network connection, both applications have an average *failover time* of ∼4s, which is ∼3s longer than that of **WhatsApp**. If the TURN server is subsequently blocked, both applications lose functionality entirely. It can be restored by unblocking either the entry server or the TURN server, with a *recovery time* of ∼1s on average, as the client has already established connections with both servers. Similar to **WhatsApp**, this restoration preserves the conversation context.
- **ChatGPT** maintains a server that the client connects to using TCP/H2 after its primary entry server is blocked. However, this connection fails to preserve functionality. Thus, this TCP/H2 server may be intended merely to maintain a control channel and wait passively for the recovery of the original server, rather than actively supporting media delivery. However, we find that even after the entry server is unblocked, **ChatGPT** still cannot recover, with the purpose of this additional TCP/H2 server remaining unknown.
- **Copilot** does not utilize any backup entry server. Once the initial entry server becomes unreachable, the application stops functioning. When the connection is unblocked, **Copilot** can recover functionality but does not resume the interrupted conversation. Instead, it restarts a new session from scratch. In other words, it re-establishes connectivity but loses the conversational context. This behavior contrasts with Meta's three applications, which preserve the ongoing conversation state after restoration. This difference likely stems from the underlying transport protocols. TCP employed by **Copilot** requires maintaining an active connection with periodic acknowledgments to keep the session alive. In contrast, UDP employed by Meta's applications is connectionless and stateless, allowing clients to continue sending packets to the same connection once unblocked.

Upon reconnection, **Copilot** shows two distinct, non-deterministic behaviors. One is to re-establish the connection with the original entry server, with a mean recovery time of 1.72s. The other is to initiate a connection with a new server, which takes 4.23s on average. This additional delay of 2.51s reflects the extra overhead of establishing a new connection to the entry server, which can potentially explain the long call setup time of **Copilot** (§4.4).

- **Gemini** does not employ any automatic failover mechanism (*e.g.*, backup server) to maintain the session once the entry server is blocked. After losing connection, it presents a fallback screen stating that "*something went wrong*" along with a user-facing "*try again*" button. Once the entry server is unblocked, it requires explicit user interaction by clicking "*try again*", which allows the user to re-establish a new session but not restore the existing one.

Bandwidth Throttling. In the uplink, all applications, regardless of the underlying transport protocol or employed transmission method (as examined in §4.2 and §4.3), fail to function when the available bandwidth becomes insufficient. This observation also applies to the downlink for applications that utilize streaming-based transmission, including **ChatGPT**, **Instagram**, **Messenger**, and

`WhatsApp`. These results indicate that none of the evaluated applications implement adaptive bitrate streaming [42, 54, 93], which dynamically switches between different transmission rates, for instance, by adjusting audio encoding quality, to accommodate varying network conditions.

One might argue that bandwidth-driven adaptations are less critical for low-bitrate services such as Meta's three applications, which consume ~50 Kbps on average (§4.3). However, this assumption is challenged by Meta's recent development of *MLow* [69], a low-bitrate audio codec capable of operating at bitrates as low as 6 Kbps. Despite this advancement, our measurements reveal no evidence that Meta's GenAI calling applications have adopted it to enhance robustness against bandwidth limitations.

For `Copilot` and `Gemini` that employ the batch-based mechanism on the downlink, we find that they can maintain functionality only when the available downlink bandwidth exceeds 1500 Kbps and 300 Kbps, respectively. When bandwidth drops below these levels, audio playback begins to lag and eventually breaks up, indicating that the download rate cannot keep up with the playback speed. This result shows that `Copilot` requires a minimum of 1500 Kbps to function properly, significantly higher than the few hundred Kbps needed by other applications, thereby limiting its usability on constrained network connections.

Summary: Our experiments reveal that the evaluated applications generally lack robust mechanisms to handle connection failures or constrained bandwidth conditions. Among them, `WhatsApp` stands out as the only one with proactive failover support, pre-establishing connections to multiple backup servers during call initiation. This design allows it to quickly switch over in the event of a connection failure. However, none of the applications exhibit adaptive behavior in response to limited bandwidth. These findings suggest that developers may currently deprioritize handling poor networking conditions, perhaps viewing them as rare or non-critical corner cases. Nonetheless, building resilience to variable and constrained network environments is crucial for GenAI calling, especially given that users may often access the service on mobile devices over wireless networks, where connectivity is inherently unstable [68, 110].

4.6 Comparison with Voice Assistants

Finally, we measure traditional voice-assistant applications that do not employ LLMs, which can be considered as predecessors to GenAI calling, and compare them with our studied GenAI calling applications. We use the same input queries (§3) as in our measurements for GenAI calling to enable a fair comparison. Specifically, we use a Google Pixel 8a to access and measure Google Assistant [39], and an iPhone 14 Pro to access and measure Apple Siri [10]. Both applications require Internet connectivity to function, meaning that user voice inputs are transmitted to backend servers for processing. Thus, their uplink transmission content is similar to that of GenAI calling applications. We observed that Google Assistant employs QUIC/H3 in the uplink with a streaming-based transmission strategy, similar to `Gemini`, while Apple Siri uses TCP/H2 in the uplink with batch-based transmission similar to `Copilot` (§4.3). In the downlink, however, these voice assistants differ fundamentally from GenAI calling. Since they do not employ LLMs, they cannot provide open-ended responses to diverse queries. Instead, they typically return non-informative fallback replies such as "I cannot answer this", or "here is the information I searched' and redirect the query to a web search engine and display the results. Consequently, in addition to a short audio message, these applications may transmit supplemental data such as search results or external links, rather than generating a fully conversational response. This highlights the most significant distinction between traditional voice assistants and GenAI calling applications.

5 Discussion

Quality of GenAI Responses. Our measurement reveals that the response quality of GenAI calling applications is intricately linked to a fundamental system design trade-off between latency and response quality: low TTFT may enhance interactivity but at the expense of response depth, content richness, or even accuracy. For example, `ChatGPT` exhibited a noticeably higher TTFT compared to `Instagram`, `Messenger`, and `WhatsApp`. However, `ChatGPT` consistently returned comprehensive answers (*e.g.,* detailed, structured essays), suggesting that it may prioritize output quality over responsiveness. Moreover, response quality is inherently subjective and difficult to quantify. Existing metrics such as ROUGE [58], BLEU [86], and inform rate [15, 117] are widely used in natural language generation tasks, but they often fail to capture contextual relevance or user satisfaction. Additionally, the audio input modality used by `ChatGPT-P` and `Gemini` may introduce further latency while aiming to preserve nuanced cues for better comprehension. These observations highlight the need for user studies or task-specific evaluation frameworks to complement existing metrics and better assess the quality of GenAI responses.

Large-scale Crowd-sourced Experiments. Although our measurement study covers 6,000 calls (§3), further insights could be gained through large-scale crowd-sourced experiments, which would enable the evaluation of GenAI calling applications from more diverse perspectives. For example, such experiments could measure performance across different countries and track changes over time to reveal potential design improvements introduced by these service providers. In addition, we can further distinguish between client network types (*e.g.,* cellular vs. WiFi) and collect network condition metrics (*e.g.,* RTT and packet loss) to analyze how varying network conditions affect GenAI calling performance. To this end, we are developing an open-source measurement framework, extending our existing tools for quantifying call setup time (§4.4), in order to support large-scale crowd-sourced studies and facilitate the evaluation of GenAI calling applications in the wild.

Inspecting Encrypted Traffic. Given that all studied applications encrypt their traffic (*e.g.,* TLS over TCP/H2 and QUIC/H3, or DTLS over UDP/RTP, as presented in §4.2), we are unable to perform direct inspection of the payload. However, analyzing encrypted traffic could help us better understand how these applications function, for instance, the specific audio codec employed and the exact batching strategy for aggregating audio frames. As TLS/DTLS are designed for end-to-end encryption [102], simply utilizing the man-in-the-middle attack cannot get the TLS/DTLS certificate, and thus it is challenging to directly decrypt the traffic. Given these constraints, we can turn to what can be inferred from side channels. Prior studies have shown that analyzing IP headers [101] and packet transmission patterns [75] may help better understand the media traffic, which is also applicable to GenAI calling that delivers audio over the network. Moreover, a recent effort from Ahmed *et al.* [3] exploited encrypted network traffic from voice-assistant applications to implement a voice fingerprinting attack that revealed certain information, such as user activities. By analogy, similar fingerprinting methods may help characterize GenAI calling traffic.

Privacy & Ethical Concerns. The deployment of human-to-GenAI calling on mobile platforms raises important privacy and ethical concerns, as these applications require continuous access to the microphone and capture not only the user’s voice but also ambient sounds. While such data is essential for delivering real-time, context-aware responses, it introduces risks related to unauthorized storage, prolonged retention, and potential third-party sharing [3, 45, 57, 100, 115]. Moreover, the potential integration of multi-modal inputs, such as camera-based context, further amplifies these concerns, as visual data may inadvertently reveal sensitive information about the user’s environment [1, 94, 106, 114]. Future research should explore privacy-preserving schemes to reduce exposure while maintaining functionality in GenAI calling.

Early Stage. Human-to-GenAI calling applications are in their infancy and are expected to evolve dramatically in the near future. The rapid development of underlying large language models [80, 82, 107, 111], improvements in real-time inference efficiency [2, 34, 51, 60, 120], and growing integration of multi-modal inputs [14, 30, 59, 74, 80, 89] suggest that both the capabilities and system architectures of these services are in flux. As service providers experiment with new features, user experience and network behaviors will likely shift accordingly. Consequently, this work represents only a snapshot of a fast-changing landscape. Future studies will be needed to track the evolution of these applications and revisit our key findings as existing platforms evolve.

Other Applications. While our study focuses on six representative applications, it is important to acknowledge that this selection is not exhaustive. Human-to-GenAI calling applications are rapidly growing, with new services and features continually emerging. As a result, our findings represent a valuable yet partial view of the current ecosystem. Expanding future measurements to include a broader and more diverse set of applications will be essential for developing a comprehensive understanding of this emerging space.

6 Related Work

Human-to-human Calling. Applications such as FaceTime, Skype, and Telegram have been extensively studied from various perspectives, including network performance, traffic patterns, and protocol behavior [4, 16, 18, 49, 53, 65, 87, 105]. For example, Chen *et al.* [18] proposed a novel model to quantify Skype user satisfaction using network trace data, thereby eliminating the need for cumbersome user studies. In contrast, our work investigates the emerging paradigm of human-to-GenAI calling, where users interact with AI agents via voice. This shift introduces new performance challenges, such as LLM inference latency, which do not present in conventional calling applications.

Measurement of other RTC Applications. Extensive prior work has examined the performance of video conferencing platforms such as Zoom, Webex, and Microsoft Teams [17, 75, 101, 110]. Other studies have focused on immersive telepresence systems [25] and social virtual reality platforms in the metaverse [23, 24, 61], investigating their latency, bandwidth demands, and resiliency. Additionally, a growing body of work examines the performance of cloud-based gaming platforms [44, 116]. In contrast to these efforts, we measure a new class of GenAI-powered RTC applications that enable real-time, voice-based interactions.

Voice Assistant. Several prior studies have investigated voice assistants such as Apple Siri and Google Assistant, driven by concerns in the security and privacy community [3, 5, 12, 45, 57, 88, 100, 115]. For example, Ahmed *et al.* [3] proposed a voice fingerprinting attack that can reveal user activities based on the network traffic. Iqbal *et al.* [45] developed a measurement framework to analyze data collection, usage, and sharing behaviors in voice assistants. In contrast, our work measures the performance of voice-enabled, LLM-powered applications that support real-time interactions.

LLM Serving has garnered significant attention from both the systems and machine learning communities. A number of recent studies have proposed system-level optimizations to improve LLM inference efficiency [2, 6, 34, 35, 51, 60, 118, 120, 122]. For example, PagedAttention introduced a virtual memory- and paging-inspired attention mechanism to support high-throughput LLM serving [51]. Concurrently, the machine learning community has explored efficient attention mechanisms for LLM serving [19, 29, 91, 103, 104]. In contrast to these efforts, which focus on backend optimizations, our work centers on measuring the end-to-end performance of GenAI calling applications.

7 Conclusion

In this paper, we presented the first comprehensive measurement study of human-to-GenAI calling applications that integrate voice interactions with LLMs to support dynamic, real-time conversations. By analyzing six representative applications, we shed light on their end-to-end operational workflows, including input modalities, network behavior, latency metrics, and resilience to adverse network conditions. For example, our measurements reveal significant latency overheads (*e.g.,* several seconds for a complete conversational exchange), far exceeding the responsiveness expected in traditional human-to-human voice communication. Our findings highlight the need for future system designs that prioritize low-latency performance. As GenAI calling matures into a mainstream communication modality, we hope our work serves as a foundational reference for building scalable, responsive, and resilient voice-driven AI experiences.

Acknowledgment

We thank the anonymous reviewers and our shepherd for their insightful feedback, which has greatly strengthened this work. We also thank Bo Chen and Nan Wu for their assistance in setting up experiments across different locations. No Meta resources were used for this paper. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of Meta. This work was partially supported by the National Science Foundation under Grants CNS-2212296, NeTS-2402992, and NeTS-2437203.

References

[1] Paarijaat Aditya, Rijurekha Sen, Peter Druschel, Seong Joon Oh, Rodrigo Benenson, Mario Fritz, Bernt Schiele, Bobby Bhattacharjee, and Tong Tong Wu. 2016. I-Pic: A Platform for Privacy-Compliant Image Capture. In *Proceedings of ACM Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*. https://doi.org/10.1145/2906388.2906412

[2] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. 2024. Taming Throughput-Latency Tradeoff in LLM Inference With Sarathi-Serve. In *Proceedings of USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. https://www.usenix.org/conference/osdi24/presentation/agrawal

[3] Dilawer Ahmed, Aafaq Sabir, and Anupam Das. 2023. Spying Through Your Voice Assistants: Realistic Voice Command Fingerprinting. In *Proceedings of USENIX Security Symposium (USENIX Security)*. https://www.usenix.org/conference/usenixsecurity23/presentation/ahmed-dilawer

[4] Waqas Ahmed, Faisal Shahzad, Abdul Rehman Javed, Farkhund Iqbal, and Liaqat Ali. 2021. WhatsApp Network Forensics: Discovering the IP Addresses of Suspects. In *Proceedings of IFIP International Conference on New Technologies, Mobility and Security (NTMS)*. https://doi.org/10.1109/NTMS49979.2021.9432677

[5] Ahmad Alhilal, Kirill Shatilov, Gareth Tyson, Tristan Braud, and Pan Hui. 2023. Network Traffic in the Metaverse: The Case of Social VR. In *Proceedings of IEEE International Conference on Distributed Computing Systems Workshops (ICDCS Workshops)*.

[6] Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, et al. 2022. Deepspeed-Inference: Enabling Efficient Inference of Transformer Models at Unprecedented Scale. In *Proceedings of ACM International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*.

[7] Android. 2025. Android Developers: SpeechRecognizer. https://developer.android.com/reference/android/speech/SpeechRecognizer. [accessed on 09/30/2025].

[8] Android. 2025. Android Developers: TextToSpeech. https://developer.android.com/reference/android/speech/tts/package-summary. [accessed on 09/30/2025].

[9] Anthropic. 2025. Claude. https://claude.ai. [accessed on 09/30/2025].

[10] Apple. 2025. Siri. https://www.apple.com/siri/. [accessed on 09/30/2025].

[11] Apple Inc. 2025. Siri. https://www.apple.com/siri/. [accessed on 09/30/2025].

[12] Mehdi Assefi, Mike Wittie, and Allan Knight. 2015. Impact of Network Performance on Cloud Speech Recognition. In *Proceedings of IEEE International Conference on Computer Communication and Networks (ICCCN)*. https://doi.org/10.1109/ICCCN.2015.7288417

[13] Marcelo Bagnulo, Philip Matthews, and Iljitsch van Beijnum. 2011. RFC 6146: Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers. https://www.rfc-editor.org/rfc/rfc6146. [accessed on 09/30/2025].

[14] Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, et al. 2025. Qwen2. 5-VL Technical Report. https://doi.org/10.48550/arXiv.2502.13923. [accessed on 09/30/2025].

[15] Yejin Bang, Samuel Cahyawijaya, Nayeon Lee, Wenliang Dai, Dan Su, Bryan Wilie, Holy Lovenia, Ziwei Ji, Tiezheng Yu, Willy Chung, et al. 2023. A Multitask, Multilingual, Multimodal Evaluation of ChatGPT on Reasoning, Hallucination, and Interactivity. In *Proceedings of International Joint Conference on Natural Language Processing and Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*. https://doi.org/10.18653/v1/2023.ijcnlp-main.45

[16] Dario Bonfiglio, Marco Mellia, Michela Meo, Dario Rossi, and Paolo Tofanelli. 2007. Revealing Skype Traffic: When Randomness Plays With You. In *Proceedings of ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*. https://doi.org/10.1145/1282380.1282386

[17] Hyunseok Chang, Matteo Varvello, Fang Hao, and Sarit Mukherjee. 2021. Can You See Me Now? A Measurement Study of Zoom, Webex, and Meet. In *Proceedings of ACM IMC*. https://dl.acm.org/doi/abs/10.1145/3487552.3487847

[18] Kuan-Ta Chen, Chun-Ying Huang, Polly Huang, and Chin-Laung Lei. 2006. Quantifying Skype User Satisfaction. In *Proceedings of ACM SIGCOMM*. https://doi.org/10.1145/1151659.1159959

[19] Lequn Chen, Zihao Ye, Yongji Wu, Danyang Zhuo, Luis Ceze, and Arvind Krishnamurthy. 2024. Punica: Multi-Tenant LoRA Serving. In *Proceedings of Machine Learning and Systems (MLSys)*. https://proceedings.mlsys.org/paper_files/paper/2024/file/054de805fcceb78a201f5e9d53c85908-Paper-Conference.pdf

[20] Lingjiao Chen, Matei Zaharia, and James Zou. 2023. FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance. https://doi.org/10.48550/arXiv.2305.05176. [accessed on 09/30/2025].

[21] Qian Chen, Yafeng Chen, Yanni Chen, Mengzhe Chen, Yingda Chen, Chong Deng, Zhihao Du, Ruize Gao, Changfeng Gao, Zhifu Gao, et al. 2025. MinMo: A Multimodal Large Language Model for Seamless Voice Interaction. https://arxiv.org/abs/2501.06282. [accessed on 09/30/2025].

[22] Zhiyang Chen, Yun Ma, Haiyang Shen, and Mugeng Liu. 2025. WeInfer: Unleashing the Power of WebGPU on LLM Inference in Web Browsers. In *Proceedings of ACM Web Conference (WWW)*. https://doi.org/10.1145/3696410.3714553

[23] Ruizhi Cheng, Nan Wu, Songqing Chen, and Bo Han. 2022. Reality Check of Metaverse: A First Look at Commercial Social Virtual Reality Platforms. In *Proceedings of IEEE Workshop for Building the Foundations of the Metaverse (Metabuild), co-located with IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*.

[24] Ruizhi Cheng, Nan Wu, Songqing Chen, and Bo Han. 2022. Will Metaverse be NextG Internet? Vision, Hype, and Reality. *IEEE Network* 36 (2022), 197–204.

[25] Ruizhi Cheng, Nan Wu, Matteo Varvello, Eugene Chai, Songqing Chen, and Bo Han. 2024. A First Look at Immersive Telepresence on Apple Vision Pro. In *Proceedings of ACM IMC*.

[26] Ruizhi Cheng, Nan Wu, Matteo Varvello, Songqing Chen, and Bo Han. 2022. Are We Ready for Metaverse? A Measurement Study of Social Virtual Reality Platforms. In *Proceedings of ACM IMC*.

[27] Ruizhi Cheng, Guowu Xie, and Bo Han. 2025. Real-time Human and Generative AI Interaction: Network Challenges and Opportunities. In *arXiv*.

[28] CNBC. 2025. Apple delays Siri AI improvements to 2026. https://www.cnbc.com/2025/03/07/apple-delays-siri-ai-improvements-to-2026.html. [accessed on 09/30/2025].

[29] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and Memory-Efficient Exact Attention With IO-Awareness. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*. https://proceedings.neurips.cc/paper_files/paper/2022/file/67d57c32e20fd0a7a302cb81d36e40d5-Paper-Conference.pdf

[30] Google DeepMind. 2025. Gemini 2.5. https://blog.google/technology/google-deepmind/gemini-model-thinking-updates-march-2025/. [accessed on 09/30/2025].

[31] Xin Luna Dong, Seungwhan Moon, Yifan Ethan Xu, Kshitiz Malik, and Zhou Yu. 2023. Towards Next-Generation Intelligent Assistants Leveraging LLM Techniques. In *Proceedings of ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. https://dl.acm.org/doi/abs/10.1145/3580305.3599572

[32] ElevenLabs. 2025. ElevenLabs: Text to Speech & AI Voice Generator. https://elevenlabs.io/. [accessed on 09/30/2025].

[33] Hugging Face. 2024. Everyday Conversations for LLMs. https://huggingface.co/datasets/HuggingFaceTB/everyday-conversations-llama3.1-2k.

[34] Yao Fu, Leyang Xue, Yeqi Huang, Andrei-Octavian Brabete, Dmitrii Ustiugov, Yuvraj Patel, and Luo Mai. 2024. ServerlessLLM: Low-Latency Serverless Inference for Large Language Models. In *Proceedings of USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. https://www.usenix.org/conference/osdi24/presentation/fu

[35] Bin Gao, Zhuomin He, Puru Sharma, Qingxuan Kang, Djordje Jevdjic, Junbo Deng, Xingkun Yang, Zhou Yu, and Pengfei Zuo. 2024. Cost-Efficient Large Language Model Serving for Multi-Turn Conversations With CachedAttention. In *Proceedings of USENIX Annual Technical Conference (USENIX ATC)*.

[36] Google. 2024. Gemini 2.0 Flash. https://cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-0-flash. [accessed on 09/30/2025].

[37] Google. 2025. Gemini. https://gemini.google.com/. [accessed on 09/30/2025].

[38] Google. 2025. Gemini 2.5 Flash. https://cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-5-flash. [accessed on 09/30/2025].

[39] Google. 2025. Google Assistant. https://assistant.google.com/. [accessed on 09/30/2025].

[40] Google DeepMind. 2024. Gemini. https://gemini.google.com/app. Accessed: 2025-05-01.

[41] hagezi. 2023. VS Code Telemetry. https://github.com/StevenBlack/hosts/issues/2301#issuecomment-1518564889. [accessed on 09/30/2025].

[42] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. 2014. A Buffer-Based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service. In *Proceedings of ACM SIGCOMM*.

[43] ipinfo.io. 2024. https://ipinfo.io/. [accessed on 09/30/2025].

[44] Hassan Iqbal, Ayesha Khalid, and Muhammad Shahzad. 2021. Dissecting Cloud Gaming Performance with DECAF. In *Proceedings of ACM Measurement and Analysis of Computing Systems (SIGMETRICS)*.

[45] Umar Iqbal, Pouneh Nikkhah Bahrami, Rahmadi Trimananda, Hao Cui, Alexander Gamero-Garrido, Daniel J Dubois, David Choffnes, Athina Markopoulou, Franziska Roesner, and Zubair Shafiq. 2023. Tracking, Profiling, and Ad Targeting in the Alexa Echo Smart Speaker Ecosystem. In *Proceedings of ACM IMC*. https://doi.org/10.1145/3618257.3624803

[46] The International Telecommunication Union Telecommunication Standardization Sector (ITU-T). 2003. One-way Transmission Time for the General Recommendations on the Transmission Quality for an Entire International Telephone Connection. https://www.itu.int/rec/t-rec-g.114-200305-i.

[47] Junchen Jiang, Shijie Sun, Vyas Sekar, and Hui Zhang. 2017. Pytheas: Enabling Data-Driven Quality of Experience Optimization Using Group-Based Exploration-Exploitation. In *Proceedings of USENIX NSDI*.

[48] Arash Molavi Kakhki, Samuel Jero, David Choffnes, Cristina Nita-Rotaru, and Alan Mislove. 2017. Taking a Long Look at QUIC: An Approach for Rigorous Evaluation of Rapidly Evolving Transport Protocols. In *Proceedings of ACM IMC*.

[49] Filip Karpisek, Ibrahim Baggili, and Frank Breitinger. 2015. WhatsApp Network Forensics: Decrypting and Understanding the WhatsApp Call Signaling Messages. *Digital Investigation* 15 (2015). https://doi.org/10.1016/j.diin.2015.09.002

[50] Michael Kerrisk. 2000. tc-netem(8) — Linux manual page. https://man7.org/linux/man-pages/man8/tc-netem.8.html. [accessed on 09/30/2025].

[51] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving With Pagedattention. In *Proceedings of Symposium on Operating Systems Principles (SOSP)*. https://doi.org/10.1145/3600006.3613165

[52] Sunjae Lee, Junyoung Choi, Jungjae Lee, Munim Hasan Wasi, Hojun Choi, Steve Ko, Sangeun Oh, and Insik Shin. 2024. MobileGPT: Augmenting LLM with Human-Like App Memory for Mobile Task Automation. In *Proceedings of ACM Annual International Conference on Mobile Computing and Networking (MobiCom)*. https://dl.acm.org/doi/10.1145/3636534.3690682

[53] Li Li, Ke Xu, Dan Wang, Chunyi Peng, Kai Zheng, Haiyang Wang, Rashid Mijumbi, and Xiangxiang Wang. 2017. A Measurement Study on Skype Voice and Video Calls in LTE Networks on High Speed Rails. In *Proceedings of IEEE/ACM International Symposium on Quality of Service (IWQoS)*. https://doi.org/10.1109/IWQoS.2017.7969110

[54] Weihe Li, Jiawei Huang, Wenjun Lyu, Baoshen Guo, Wanchun Jiang, and Jianxin Wang. 2022. RAV: Learning-Based Adaptive Streaming to Coordinate the Audio and Video Bitrate Selections. *IEEE Transactions on Multimedia* 25 (2022), 5662–5675.

[55] Zhihao Li, Dave Levin, Neil Spring, and Bobby Bhattacharjee. 2018. Internet Anycast: Performance, Problems, & Potential. In *Proceedings of ACM SIGCOMM*.

[56] Xinyu Lian, Yinfang Chen, Runxiang Cheng, Jie Huang, Parth Thakkar, Minjia Zhang, and Tianyin Xu. 2024. Large Language Models as Configuration Validators. In *Proceedings of IEEE/ACM International Conference on Software Engineering (ICSE)*. https://www.computer.org/csdl/proceedings-article/icse/2025/056900a204/215aWCaXlSg

[57] Song Liao, Christin Wilson, Long Cheng, Hongxin Hu, and Huixing Deng. 2020. Measuring the Effectiveness of Privacy Policies for Voice Assistant Applications. In *Proceedings of Annual Computer Security Applications Conference (ACSAC)*. https://doi.org/10.1145/3427228.3427250

[58] Chin-Yew Lin. 2004. ROUGE: A Package for Automatic Evaluation of Summaries. In *Proceedings of Text Summarization Branches Out*. https://aclanthology.org/W04-1013/

[59] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023. Visual Instruction Tuning. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*. https://proceedings.neurips.cc/paper_files/paper/2023/file/6dcf277ea32ce3288914faf369fe6de0-Paper-Conference.pdf

[60] Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, et al. 2024. CacheGen: KV Cache Compression and Streaming for Fast Large Language Model Serving. In *Proceedings of ACM SIGCOMM*.

[61] Minzhao Lyu, Rahul Dev Tripathi, and Vijay Sivaraman. 2023. Metavradar: Measuring Metaverse Virtual Reality Network Activity. In *Proceedings of ACM Measurement and Analysis of Computing Systems (SIGMETRICS)*. https://doi.org/10.1145/3626786

[62] Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. LLM-Pruner: On the Structural Pruning of Large Language Models. *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*.

[63] Rohan Mahy, Philip Matthews, and Jonathan Rosenberg. 2010. RFC 5766: Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN). https://www.rfc-editor.org/rfc/rfc5766. [accessed on 09/30/2025].

[64] Yacine Majdoub and Eya Ben Charrada. 2024. Debugging With Open-Source Large Language Models: An Evaluation. In *Proceedings of ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. https://dl.acm.org/doi/abs/10.1145/3674805.3690758

[65] Natalia M Markovich and Udo R Krieger. 2010. Statistical Analysis and Modeling of Skype VoIP Flows. *Computer Communications* 33 (2010). https://doi.org/10.1016/j.comcom.2010.04.029

[66] MaxMind. 2024. https://www.maxmind.com/en/home. [accessed on 09/30/2025].

[67] Trevor Mendez, Walter Milliken, and Dr. Craig Partridge. 1993. Host Anycasting Service. RFC 1546. https://www.rfc-editor.org/info/rfc1546 [accessed on 09/30/2025].

[68] Jiayi Meng, Jingqi Huang, Y Charlie Hu, Yaron Koral, Xiaojun Lin, Muhammad Shahbaz, and Abhigyan Sharma. 2023. Modeling and Generating Control-Plane Traffic for Cellular Networks. In *Proceedings of ACM IMC*.

[69] Meta. 2024. MLow: Meta's low bitrate audio codec. https://engineering.fb.com/2024/06/13/web/mlow-metas-low-bitrate-audio-codec/. [accessed on 09/30/2025].

[70] Meta. 2025. Instagram. https://www.instagram.com/. [accessed on 09/30/2025].

[71] Meta. 2025. Introducing the Meta AI App: A New Way to Access Your AI Assistant. https://about.fb.com/news/2025/04/introducing-meta-ai-app-new-way-access-ai-assistant/. [accessed on 09/30/2025].

[72] Meta. 2025. Messenger. https://www.messenger.com/.

[73] Meta. 2025. WhatsApp. https://whatsapp.com/. [accessed on 09/30/2025].

[74] Meta Platforms, Inc. 2025. The Llama 4 Herd: The Beginning of a New Era of Natively Multimodal AI Innovation. https://ai.meta.com/blog/llama-4-multimodal-intelligence/. [accessed on 09/30/2025].

[75] Oliver Michel, Satadal Sengupta, Hyojoon Kim, Ravi Netravali, and Jennifer Rexford. 2022. Enabling Passive Measurement of Zoom Performance in Production Networks. In *Proceedings of ACM IMC*.

[76] Microsoft. 2022. AltspaceVR. https://altvr.com/. [accessed on 09/30/2025].

[77] Microsoft. 2024. Teams. https://www.microsoft.com/en-us/microsoft-teams/group-chat-software. [accessed on 09/30/2025].

[78] Microsoft. 2025. Copilot. https://copilot.microsoft.com/. [accessed on 09/30/2025].

[79] Microsoft Corporation. 2025. Skype. https://www.skype.com/en/. [accessed on 09/30/2025].

[80] OpenAI. 2023. Gpt-4 Technical Report. https://arxiv.org/abs/2303.08774. [accessed on 09/30/2025].

[81] OpenAI. 2024. ChatGPT. https://openai.com/. Accessed: 2025-05-01.

[82] OpenAI. 2024. Gpt-4o System Card. https://arxiv.org/abs/2410.21276. [accessed on 09/30/2025].

[83] OpenAI. 2025. ChatGPT. https://openai.com/index/chatgpt/. [accessed on 09/30/2025].

[84] OpenAI. 2025. What is the ChatGPT model selector? https://help.openai.com/en/articles/7864572-what-is-the-chatgpt-model-selector. [accessed on 09/30/2025].

[85] Victor Alexandru Padurean, Paul Denny, and Adish Singla. 2025. BugSpotter: Automated Generation of Code Debugging Exercises. In *Proceedings of ACM Technical Symposium on Computer Science Education*. https://dl.acm.org/doi/10.1145/3641554.3701974

[86] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: A Method for Automatic Evaluation of Machine Translation. In *Proceedings of ACM Annual Meeting on Association for Computational Linguistics*. https://doi.org/10.3115/1073083.1073135

[87] Nayankumar Patel, Swapnil Patel, and Wee Lum Tan. 2018. Performance Comparison of WhatsApp Versus Skype on Smart Phones. In *Proceedings of International Telecommunication Networks and Applications Conference (ITNAC)*. https://doi.org/10.1109/ATNAC.2018.8615445

[88] Surendra Pathak, Sheikh Ariful Islam, Honglu Jiang, Lei Xu, and Emmett Tomai. 2022. A Survey on Security Analysis of Amazon Echo Devices. *High-Confidence Computing* 2 (2022), 100087. https://www.sciencedirect.com/science/article/pii/S2667295222000393

[89] Anthropic PBC. 2024. Claude 3.5 Sonnet. https://www.anthropic.com/news/claude-3-5-sonnet. [accessed on 09/30/2025].

[90] Android Police. 2025. Google Gemini vs. Gemini Advanced: All the key differences explained. https://www.androidpolice.com/google-gemini-vs-gemini-advanced/. [accessed on 09/30/2025].

[91] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2023. Efficiently Scaling Transformer Inference. In *Proceedings of Machine Learning and Systems (MLSys)*. https://proceedings.mlsys.org/paper_files/paper/2023/file/c4be71ab8d24cdfb45e3d06dbfca2780-Paper-mlsys2023.pdf

[92] Kun Qian, Yongqing Xi, Jiamin Cao, Jiaqi Gao, Yichi Xu, Yu Guan, Binzhang Fu, Xuemei Shi, Fangbo Zhu, Rui Miao, et al. 2024. Alibaba HPN: A Data Center Network for Large Language Model Training. In *Proceedings of ACM SIGCOMM*.

[93] Yanyuan Qin, Subhabrata Sen, and Bing Wang. 2019. ABR Streaming with Separate Audio and Video Tracks: Measurements and Best Practices. In *Proceedings of ACM International Conference on Emerging Networking Experiments And Technologies (CoNEXT)*.

[94] Eric D Ragan, Hye-Chung Kum, Gurudev Ilangovan, and Han Wang. 2018. Balancing Privacy and Information Disclosure in Interactive Record Linkage With Visual Masking. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI)*. https://doi.org/10.1145/3173574.3173900

[95] Yi Ren, Yangjun Ruan, Xu Tan, Tao Qin, Sheng Zhao, Zhou Zhao, and Tie-Yan Liu. 2019. FastSpeech: Fast, Robust and Controllable Text to Speech. In *Proceedings of Conference on Neural Information Processing Systems (NeurIPS)*.

[96] Charlie F Ruan, Yucheng Qin, Xun Zhou, Ruihang Lai, Hongyi Jin, Yixin Dong, Bohan Hou, Meng-Shiun Yu, Yiyan Zhai, Sudeep Agarwal, et al. 2024. WebLLM: A High-Performance In-Browser LLM Inference Engine. https://arxiv.org/abs/2412.15803. [accessed on 09/30/2025].

[97] Rusty Russell. [n. d.]. iptables - administration tool for IPv4 packet filtering and NAT.

[98] Fardin Ahsan Sakib, Saadat Hasan Khan, and AHM Rezaul Karim. 2024. Extending the Frontier of ChatGPT: Code Generation and Debugging. In *Proceedings of International Conference on Electrical, Computer and Energy Technologies (ICECET)*. https://ieeexplore.ieee.org/abstract/document/10698405

[99] Henning Schulzrinne, Stephen L. Casner, Ron Frederick, and Van Jacobson. 2003. RTP: A Transport Protocol for Real-Time Applications. RFC 3550. https://rfc-editor.org/rfc/rfc3550.txt [accessed on 09/30/2025].

[100] William Seymour, Xiao Zhan, Mark Coté, and Jose Such. 2023. A Systematic Review of Ethical Concerns with Voice Assistants. In *Proceedings of AAAI/ACM Conference on AI, Ethics, and Society*. https://doi.org/10.1145/3600211.3604679

[101] Taveesh Sharma, Tarun Mangla, Arpit Gupta, Junchen Jiang, and Nick Feamster. 2023. Estimating WebRTC Video QoE Metrics Without Using Application Headers. In *Proceedings of ACM IMC*. https://doi.org/10.1145/3618257.3624828

[102] Yaron Sheffer, P Saint-Andre, and T Fossati. 2022. RFC 9325: Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS). https://www.rfc-editor.org/rfc/rfc9325.html. [accessed on 09/30/2025].

[103] Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, Joseph E. Gonzalez, and Ion Stoica. 2024. S-LoRA: Serving Thousands of Concurrent LoRA Adapters. https://arxiv.org/abs/2311.03285. [accessed on 09/30/2025].

[104] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. 2023. Flexgen: High-Throughput Generative Inference of Large Language Models With a Single GPU. In *Proceedings of International Conference on Machine Learning (ICML)*. https://proceedings.mlr.press/v202/sheng23a.html

[105] C Shubha, SA Sushma, and KH Asha. 2019. Traffic Analysis of WhatsApp Calls. In *Proceedings of International Conference on Advances in Information Technology (ICAIT)*. https://doi.org/10.1109/ICAIT47043.2019.8987315

[106] Abigale Stangl, Kristina Shiroma, Nathan Davis, Bo Xie, Kenneth R Fleischmann, Leah Findlater, and Danna Gurari. 2022. Privacy Concerns for Visual Assistance Technologies. *ACM Transactions on Accessible Computing (TACCESS)* 15 (2022), 1–43. https://doi.org/10.1145/3517384

[107] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and Efficient Foundation Language Models. https://arxiv.org/pdf/2302.13971. [accessed on 09/30/2025].

[108] Jean-Marc Valin and Cary Bran. 2016. RFC7874: WebRTC Audio Codec and Processing Requirements. https://datatracker.ietf.org/doc/html/rfc7874. [accessed on 09/30/2025].

[109] Jean-Marc Valin, Koen Vos, and Tim Terriberry. 2012. Definition of the Opus Audio Codec. RFC 6716. https://rfc-editor.org/rfc/rfc6716.txt [accessed on 09/30/2025].

[110] Matteo Varvello, Hyunseok Chang, and Yasir Zaki. 2022. Performance Characterization of Videoconferencing in the Wild. In *Proceedings of ACM IMC*. https://doi.org/10.1145/3517745.3561442

[111] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All You Need. In *Proceedings of Conference on Neural Information Processing Systems (NeurIPS)*.

[112] Hao Wen, Yuanchun Li, Guohong Liu, Shanhui Zhao, Tao Yu, Toby Jia-Jun Li, Shiqi Jiang, Yunhao Liu, Yaqin Zhang, and Yunxin Liu. 2024. Autodroid: LLM-Powered Task Automation in Android. In *Proceedings of ACM Annual International Conference on Mobile Computing and Networking (MobiCom)*. https://dl.acm.org/doi/10.1145/3636534.3649379

[113] Wireshark. 1998. https://www.wireshark.org/. [accessed on 09/30/2025].

[114] Nan Wu, Ruizhi Cheng, Songqing Chen, and Bo Han. 2025. PIPE: Privacy-preserving 6DoF Pose Estimation for Immersive Applications. In *Proceedings of ACM Conference on Embedded Networked Sensor Systems (SenSys)*. https://doi.org/10.1145/3715014.3722069

[115] Fuman Xie, Yanjun Zhang, Chuan Yan, Suwan Li, Lei Bu, Kai Chen, Zi Huang, and Guangdong Bai. 2022. Scrutinizing Privacy Policy Compliance of Virtual Personal Assistant Apps. In *Proceedings of IEEE/ACM International Conference on Automated Software Engineering*. https://doi.org/10.1145/3551349.3560416

[116] Xiaokun Xu and Mark Claypool. 2022. Measurement of Cloud-based Game Streaming System Response to Competing TCP Cubic or TCP BBR Flows. In *Proceedings of ACM IMC*. https://doi.org/10.1145/3517745.3561464

[117] Yunyi Yang, Yunhao Li, and Xiaojun Quan. 2021. Ubar: Towards Fully End-to-End Task-Oriented Dialog System With GPT-2. In *Proceedings of AAAI Conference on Artificial Intelligence*. https://doi.org/10.1609/aaai.v35i16.17674

[118] Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. 2025. CacheBlend: Fast Large Language Model Serving for RAG With Cached Knowledge Fusion. In *Proceedings of European Conference on Computer Systems (EuroSys)*. https://doi.org/10.1145/3689031.3696098

[119] Dong Yu and Lin Deng. 2016. *Automatic Speech Recognition*. Vol. 1. Springer.

[120] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. Orca: A Distributed Serving System for Transformer-Based Generative Models. In *Proceedings of USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. https://www.usenix.org/conference/osdi22/presentation/yu

[121] Zapier. 2025. The best large language models (LLMs) in 2025. https://zapier.com/blog/best-llm/. [accessed on 09/30/2025].

[122] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. 2024. DistServe: Disaggregating Prefill and Decoding for Goodput-Optimized Large Language Model Serving. In *Proceedings of USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. https://www.usenix.org/conference/osdi24/presentation/zhong-yinmin

Appendix

A Ethics

This work does not raise any ethical issues.

B Automated Testing Tool for Measuring Call Setup Time

This section describes our method for measuring call setup time (§4.4). A key challenge in this measurement is that applications do not explicitly expose when they are ready to accept user input. As a result, we must probe this readiness indirectly. Our goal is to identify the earliest point when user input could be successfully processed after call initiation. To achieve this, we design a probing workflow that gradually adjusts the timing of the first input query. To enable large-scale automated experiments, we develop an Android-based testing tool with the following three capabilities: (1) automatically pressing the button to initiate a call, (2) waiting for a configurable duration, and (3) playing a pre-recorded audio query. By varying the wait duration in step (2), we can probe when the system becomes ready to accept input.

We detect whether the GenAI calling application has accepted the query and given the response by monitoring the network traffic because if the application generates a response, we can detect a corresponding increase in downlink traffic, as playback of the AI-generated audio triggers additional network activity (see Figure 3). We validate this detection method by recording the experiments and manually verifying the accuracy of response detection.

In practice, we observe that the earliest timing that consistently triggers a successful response remains stable across repeated trials conducted within a short time window (*e.g.,* five minutes). To ensure measurement accuracy, we perform pre-experiments by repeating the probing process multiple times. Once the variation of call setup times converges to within 100 ms across five successful sessions, we proceed with data collection for the next five-minute window.

We repeat this procedure at different times of day over multiple days. The variation in the call setup times reported in Figure 6 comes from results captured at different times. We hypothesize that this variability may result from fluctuating server-side conditions, such as increased server load or network congestion during peak usage periods compared to off-peak times, such as late at night.

C Queries for Evaluating TTFT Optimization

This section presents the *short query* and *long query* we designed to evaluate whether the studied applications apply optimizations to reduce TTFT (§4.4). Both queries ask the GenAI to write an eight-hundred-word essay, which we determine to be the longest response length that all tested platforms can support. This ensures that the task is sufficiently challenging while keeping the evaluation fair across all platforms. The full content of the *short query* and *long query* is provided below.

Short Query:

> *Write an eight-hundred-word essay that compares three major interpretability techniques for large language models, which are feature attribution, mechanistic interpretability, and behavioral probing.*

Long Query:

> *Write an eight-hundred-word essay that compares three major interpretability techniques for large language models, which are feature attribution, mechanistic interpretability, and behavioral probing. I will provide some information related to this topic. Interpreting how large language models work is essential for improving their transparency, reliability, and safety. Among the many techniques proposed, three stand out as particularly influential: feature attribution, mechanistic interpretability, and behavioral probing. Feature attribution methods attempt to identify which parts of the input most influenced the model's prediction. Tools like saliency maps or attention visualizations fall into this category. They are intuitive and often used for debugging, but can be unstable and may not reflect true causality. Mechanistic interpretability takes a deeper approach by analyzing internal components such as neurons or circuits. Techniques like activation patching or circuit tracing can provide detailed insights, though they are often resource-intensive and difficult to scale. Behavioral probing evaluates how models respond to controlled input changes or activation modifications. It focuses on observable behavior and is useful for uncovering latent knowledge or biases. However, it doesn't always explain why those behaviors emerge. Each of these techniques offers unique advantages and limitations. Together, they help researchers develop a more comprehensive understanding of how LLMs operate. Now, give me the essay that I mentioned to you with eight hundred words.*

The long query repeats the same initial task description as the short query but adds additional background and context before asking the model to proceed. This design enables us to isolate whether the system waits for the entire input or begins processing as soon as the task is recognizable, thereby revealing potential optimizations aimed at reducing TTFT for long queries.