IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 29 March 2024, accepted 26 April 2024, date of publication 6 May 2024, date of current version 18 June 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3397053

RESEARCH ARTICLE

Assessing Performance of Cloud-Based Heterogeneous Chatbot Systems and A Case Study

GANESH REDDY GUNNAM, DEVASENA INUPAKUTIKA, RAHUL MUNDLAMURI, SAHAK KAGHYAN, AND DAVID AKOPIAN, (Senior Member, IEEE)
Electrical and Computer Engineering Department, The University of Texas at San Antonio, San Antonio, TX 78249, USA
Corresponding author: Ganesh Reddy Gunnam (gunnamganeshreddy@gmail.com)

ABSTRACT Recently, human-machine digital assistants gained popularity and are commonly used in question-and-answer applications and similar consumer-supporting domains. A class of more sophisticated digital assistants (chatbots) employing more extended dialogs follows the trend. Chatbots have become increasingly popular in recent years. Nowadays, chatbot deployments in the cloud have become a common practice because of their benefits, including flexibility, scalability, reliability, security, remote working, cost, and power outages. However, measuring the cloud-based chatbot systems' performance is challenging as human-machine information exchanges are performed in heterogeneous environments such as cloud hosting platforms, information processing units, and several machine-to-machine and human-machine communication channels. This paper investigates different methodologies for assessing the performance of such heterogeneous deployments and identifies performance metrics for evaluating the performance of cloud-based chatbot deployment. The study employs chatbot performance measurements with both real users (human) and automated (simulated) users. The experimental results discuss communication metrics such as response time, throughput, and load testing (connection loss) through the performance assessment of a case study deployment that utilizes an automated protocol chatbot development framework. The findings presented in this paper can further help practitioners to better understand the performance characteristics of a cloud-based chatbot and assist in making informed decisions related to the chatbot development and deployment options.

INDEX TERMS Chatbot, cloud computing, performance assessment methodology.

I. INTRODUCTION

A chatbot is an artificial intelligence (AI) software that can simulate a conversation with a user in natural language through messaging applications, websites, and mobile apps [1]. Chatbots can be classified into various categories, such as Rule-based chatbots and Intent-based chatbots [2]. The rule-based approach defines the chatbot's response based on specified conditions or rules, typically used for narrowly defined conversation protocols. However, rule-based chatbots could not adequately respond beyond the protocol conditions and rules.

In contrast, the Intent-based approach relies on the chatbot's ability to learn and gather information. To achieve this, the chatbot must be trained in natural language processing (NLP) using datasets containing conversation dialogs to extract the combination of conversation elements, including intent, context, and entity [3]. This approach enables more human-like conversations between the user and the chatbot. Many modern tools, such as IBM Watson [4], Api.ai or Dialogflow [5], and Wit.ai [6], follow this approach.

Chatbots are typically deployed on dedicated servers or cloud platforms, depending on cost, control, security, and

The associate editor coordinating the review of this manuscript and approving it for publication was Vlad Diaconita.

customization requirements. In recent years, many companies have been migrating their on-premises-server applications to the cloud due to the increasing availability of cognitive services offered by cloud providers. These services include machine learning, translation, and analytics, which serve as building blocks for developing AI applications [7]. Maintaining applications on-premises can lead to significant challenges, including the cost of physical servers, susceptibility to power outages, and limited accessibility from external locations. Cloud computing offers customers the flexibility to scale their IT infrastructure and applications dynamically based on current demand in a shared and scalable environment, using a pay-as-you-go billing model [8]. Cloud infrastructure eliminates the need for upfront capital investments in hardware, data centers, and maintenance. Instead, pay-as-you-go models are becoming increasingly popular and can be more cost-effective, particularly for small or medium-sized businesses. Additionally, businesses can save on ongoing expenses related to power, cooling, physical security, and equipment upgrades.

In cloud environments, intent-based chatbots can utilize cloud-supported machine-learning tools to train with diverse datasets and scale up applications to serve more users. Most cloud providers offer scalable options in their databases, complete with regular backup features to maintain data integrity even in the event of application crashes. Recently, there has been significant growth among cloud providers such as Amazon Web Services (AWS), Microsoft Azure, and Google. Chatbots must leverage the strengths of different cloud platforms through heterogeneous cloud architecture to meet the increasing demands of businesses and users, ensuring efficiency and a wide range of features. This architecture integrates public and private components from multiple cloud vendors. This paper presents an empirical study of the performance of cloud-based heterogeneous chatbot systems, leveraging the benefits of cloud infrastructure in terms of overall cost, scalability, and availability.

Measuring the performance of cloud-based heterogeneous chatbot systems presents a multifaceted challenge, prompting an in-depth exploration of varied methodologies in this paper. Performance is assessed through real-time testing with both real users (humans) and automated simulations (simulated users). Including real users ensures a holistic performance assessment, capturing the fine distinction of natural interactions and offering valuable insights into the system's responsiveness and user satisfaction. Simultaneously, using automated simulated users allows for controlled testing scenarios, enabling the systematic exploration of the chatbot's capabilities under predefined conditions.

In addition to practical testing, this paper reviews existing performance measurement methodologies. This review aims to clarify essential insights into the cloud-based chatbot deployment performance landscape, finding key metrics crucial for a thorough assessment. By combining practical experimentation and a critical analysis of existing methodologies, this research contributes to a fine understanding of the challenges and opportunities inherent in assessing the performance of cloud-based heterogeneous chatbot systems.

The major contributions of this paper are as follows:

1. First, a performance measurement methodology is proposed for cloud-based chatbot systems in heterogeneous environments for measuring response time, and throughput and performing load testing.
2. Second, an extensive assessment and statistical analysis of the response times and throughput under stress with different real-time (human) testing and automated (simulated users) cloud-chatbot communication configurations are conducted.

The remainder of this paper is organized as follows. Section II covers a background and literature survey of chatbots, cloud, and microservice-based chatbot applications, followed by a review of performance metrics. Section III provides the experimental settings covering the on-premises server, AWS cloud server, and RDS database and their detailed specifications. Section IV covered all the performance metrics used for assessments in this paper. Section V presents the performance results and provides concluding remarks in the following section.

II. BACKGROUND

Integrating chatbot applications within a microservices architecture on cloud platforms like AWS impacts operation performance in heterogeneous environments. Virtualization, a foundational technology in modern computing, plays a crucial role in enabling the creation of virtual environments across hardware, storage, operating systems, and networks. Within cloud platforms, virtualization allows for the efficient allocation and management of resources, providing the underlying infrastructure needed to support microservices-based applications. By leveraging virtualization technologies, microservices can be deployed and scaled dynamically, optimizing resource utilization, and enhancing flexibility in cloud environments. Thus, virtualization serves as the backbone for hosting microservices, enabling the development and deployment of scalable and resilient chatbot applications in cloud-based architectures.

In cloud computing, virtualization serves to centralize memory, storage, and bandwidth, facilitating efficient computing [9]. This technology enables users to maintain their applications in the cloud, accessible from any internet-connected computer. According to NIST (National Institute of Standards and Technologies), cloud computing is typically categorized into three models: SaaS (Software as a Service), PaaS (Platform as a Service), and IaaS (Infrastructure as a Service) [18]. Virtualization manifests in various forms, including hardware virtualization, desktop virtualization, nested virtualization, and containerization [10], [11], [12], [13], [14], [15]. Hardware virtualization involves creating virtual machines (VMs) with their operating systems, while desktop virtualization, such as virtual desktop infrastructure (VDI), enhances data security for companies by separating logical desktops from physical servers [14]. Nested

virtualization enables the implementation of virtualization within a VM, allowing for diverse application compatibility.

Containerization, exemplified by platforms like Docker, facilitates the packaging of applications and their dependencies into standardized, self-contained units known as containers [15], [16], [17]. Containerization, particularly with Docker, has emerged as a lightweight and scalable solution for deploying microservice-based applications. Microservices, characterized by their small, autonomous nature, offer flexibility and scalability in system design. These services can be deployed independently, supported by deployment and orchestration frameworks, thus enabling the integration of heterogeneous services with minimal communication overhead [1].

This study uses Docker containers to implement a microservice-based chatbot application on the AWS Cloud. This architecture integrates external communication channels like Facebook Messenger, Twilio, and WhatsApp to facilitate seamless data transfer essential for the chatbot's functionality. In the next section, we will delve into the specific performance metrics used to evaluate the effectiveness of our chatbot application in this cloud-based microservices environment.

III. RELATED WORK: PERFORMANCE ASSESSMENT OF CLOUD-BASED HETEROGENEOUS SYSTEMS

In human-machine dialogue systems, particularly with the cloud-based chatbot deployment, there has been a notable surge in research driven by such benefits as flexibility, scalability, reliability, etc. State-of-the-art chatbot systems commonly adopt heterogeneous deployment environments, integrating services from various cloud-based platforms with end-user mobile devices. These services include natural language processing (NLP), hosting conversation protocols, infrastructure for messaging channels, complementary technologies like serverless architectures and cloud APIs to enhance efficiency and extend functionalities, etc. Heterogeneous environments enable multi-modal interactions, ensuring cross-platform compatibility and establishing connectivity with diverse devices, thereby requiring adaptability to various data types and user preferences. Assessing system performance metrics in these diverse deployments faces numerous challenges.

The performance of conventional cloud deployments has been thoroughly addressed in the literature as shown in Table 1. Ataş and Güngör [20] explore statistical methods to measure response time, average total operation time, and memory bandwidth. Khanghahi and Ravanmehr [21] delve into metrics like average response time, average time per data center, total cost, and cost per VM of the Data Center, providing an overall perspective on cloud evaluation criteria and highlighting it with the help of simulation. Rak et al. [22] present a technique to evaluate the trade-off between costs and performance of cloud applications by employing benchmarks and simulation, where they gauge overall response time, throughput histograms, and resource usage histograms. Stantchev [23] assess response time, transaction rate, and availability, and introduce an approach for performance evaluation of advanced computing infrastructures. Papadopoulos et al. [19] establish principles for reproducible performance evaluation in cloud computing, emphasizing throughput and response time. Lastly, Bahga and Madisetti [24] scrutinize average throughput and response time, describing a generic performance evaluation methodology for complex multi-tier applications deployed in cloud computing environments. Mohammad et al. [25] review over twenty papers and benchmarks, considering metrics such as throughput, response time, and the number of application instances. These studies significantly contribute valuable insights into intra-cloud performance issues, focusing on the dynamics within a single cloud environment and enhancing the understanding and assessment of cloud systems' performance at the system level. However, as cloud computing evolves and becomes more intricate, these studies do not address the more complicated challenges presented by inter-cloud and heterogeneous environments.

TABLE 1. Key performance metrics of cloud application.

Source	Key Performance Metrics
Performance evaluation of cloud computing platforms using statistical methods [20]	• Response time • Average total operation time • Memory bandwidth
Cloud Computing Performance Evaluation: Issues and Challenges [21]	• Average response time • Average time per data center • The total cost • Cost per VM of Data Center
Cost/performance evaluation for cloud applications using simulation [22]	• Overall response time • Throughput histogram • Resource usage histogram
Performance Evaluation of Cloud Computing Offerings [23]	• Response time, • Transaction rate, • Availability
Methodological Principles for Reproducible Performance Evaluation in Cloud Computing [19]	• Throughput • Response time
Performance Evaluation Approach for Multi-Tier Cloud Applications [24]	• Average Throughput • Response time
Performance evaluation metrics for cloud, fog, and edge computing: A review, taxonomy, benchmarks, and standards for future research [25]	• Throughput • Response time

Recent state-of-the-art research explored the performance aspects of heterogeneous cloud systems as shown in Table 2. The recent study [26] focused on the execution time, request processing time, CPU usage, and Memory usage. In another study [27], data transfer times between different cloud

functions were measured in a heterogeneous cloud system. The performance metrics chosen for the comparison are the schedule length (Make span), total cost, and task rejection rate [28]. Joel Scheuner and Philipp Leitner [29] reviewed eighty-two relevant studies on cloud performance studies which include heterogeneous systems and listed the most often used metrics, such as latency, CPU, throughput, and network. Li et al. [30] investigate execution time, startup delay, deadline miss rate, and incurred cost in video transcoding. Lastly, Singh et al. [31] assess make span and resource utilization cost, contributing to the evolving discourse on scheduling efficiency in heterogeneous cloud environments. Mahmoud et al. [41] evaluated the performance of large-scale heterogeneous IoT data, and the main metrics in their experiments were the response time and the database size. These studies primarily focus on metrics at the general system level. However, the crucial nuances and intricacies involved in the communication aspects of chatbots, especially in heterogeneous cloud environments, remain unexplored in the existing literature. Therefore, there is a notable gap in addressing the unique challenges of chatbot systems operating in diverse and interconnected cloud environments.

TABLE 2. Key performance metrics of heterogeneous cloud services.

Source	Key Performance Metrics
Benchmarking heterogeneous cloud functions [26]	• Execution time • Request Processing time • CPU usage • Memory usage
Performance evaluation of heterogeneous cloud functions [27]	• Data Transfer time
Budget-constraint stochastic task scheduling on heterogeneous cloud systems [28]	• Schedule Length • Total cost • Cost Rejection Rate
Function-as-a-service performance evaluation: A multivocal literature review [29]	• Latency • Throughput • CPU • Network
Cost-Efficient and Robust On-Demand Video Transcoding Using Heterogeneous Cloud Services [30]	• Execution Time • Startup Delay • Deadline Miss Rate • Incurred Cost
Metaheuristics for scheduling of heterogeneous tasks in cloud computing environments: Analysis, performance evaluation, and future directions [31]	• Make Span • Resource Utilization Cost
Performance Evaluation of IoT Data Management Using MongoDB Versus MySQL Databases in Different Cloud Environments [41]	• Response Time • Database Size

The performance metrics related to chatbot systems focus on Machine-Learning (ML) aspects, exploring metrics such as response time, average total operation time, memory bandwidth, average time per data center, total cost, cost per VM, throughput histograms, resource usage histograms, fallback rate, accuracy, precision, recall, F1-score, and conversation interruption as shown in Table 3.

Forkan et al. [43] propose a tool for empirical evaluation, introducing metrics like Average Response time, Fallback Rate, Comprehensive Rate, Accuracy, Precision, Recall, and F1-score. Zubani et al. [44] compare performance using metrics like F1-score, error rate, response time, and robustness. Carolyn F. Salazar [45] evaluates a Pedagogical Conversational Agent using a Cloud-Based Chatbot Builder, employing a black-box method and a checklist to assess adherence to international standards. Jaing et al. [46] and [47], [48] use metrics based on communication principles, including specificity, relevance, response clarity, informativeness, response length, expansiveness, user engagement, and conversational interruption. However, the identified gap lies in the fact that these assessments predominantly revolve around the algorithmic aspects and ML capabilities of chatbots. They often fall short in providing a holistic assessment that integrates both the system-level and algorithmic-level performances of chatbot systems. The need for a more thorough assessment that considers the synergies between the underlying systems and the implemented algorithms becomes clear in the existing literature.

TABLE 3. Key performance metrics of cloud chatbot applications.

Source	Key Performance Metrics
Echo: a tool for empirical evaluation cloud chatbots [43]	• Average Response Time • Fallback Rate • Comprehensive Rate • Accuracy • Precision • Recall • F1-Score
Performance Evaluation of IoT Data Management Using MongoDB Versus MySQL Databases in Different Cloud Environments [44]	• F-Score • Error Rate • Response Time • Robustness
CommunityBots: Creating and evaluating a Multi-Agent chatbot platform for public input elicitation [46-48]	• Specificity • Relevance • Response Clarity • Informativeness

These papers collectively contribute to our understanding of performance metrics, offering valuable insights for practitioners in cloud-based chatbot development and deployment. Having analyzed studies assessing cloud chatbot systems, observed a reliance on machine learning or artificial intelligence in performance assessment methods. Conversely, heterogeneous cloud systems evaluated performance at the system level, not specifically applied to chatbot systems. This paper bridges this gap, proposing a performance measurement methodology tailored for cloud-based chatbot systems

in heterogeneous environments, addressing the performance assessment of chatbots with and without integrated machine learning.

We emphasize communication performance metrics like response time, connection loss, and throughput with a case study that excludes machine learning aspects. Response time is crucial in chatbot-based applications since users will expect a chatbot response quickly, and it is understandable that nobody wants to wait longer to get a reply. Throughput measures how many user requests or interactions a chatbot can handle in a given time. Additionally, the connection loss metric is important in chatbot performance assessment. Connection loss refers to the interruption or breakdown in communication between the user and the chatbot. Both throughput and connection loss metrics directly affect the user experience and the effectiveness of the conversation as well as overall chatbot performance.

The contributions include proposing a novel performance measurement methodology and conducting an extensive assessment and statistical analysis of response times and throughput under stress conditions with both real (human) and automated (simulated users) testing configurations.

IV. PERFORMANCE ASSESSMENT METHODOLOGY

Case Study Platform (Dash-Messaging)

A case-study messaging system referred to as Dashmessaging [32] in the following, is developed and deployed on the AWS cloud to explore the performance assessment aspects. Figure 1 shows the architecture of the performance assessment methodology used in this work. The AWS RDS database is attached to the system. This chatbot is a representative implementation that can be accessed from third-party messaging applications such as Facebook, WhatsApp, Twilio, etc. These messaging applications can be connected to the test system through the webhook [33]. In this work, two evaluation approaches are used to address Human testing and Automated testing.

Dash-Messaging hosts interactive automated messaging protocols [32] and is designed for long-spread conversations. In this system, the conversation will be triggered by keywords from the participants. From there, the conversation follows the respective branch based on the participant's responses [34]. While the presented performance evaluation methodology was applied and tested using the abovementioned test system Dashmessaging, it can apply to any type of chatbot hosted in cloud environments.

A. TESTING METHODOLOGIES

1) HUMAN TESTING

The round-trip response time between the User and the Chatbot was calculated for this testing. The communication was enabled by actual messaging channels such as WhatsApp and Messenger. The response time was received by sending a message request to the case study chatbot through these messaging channels. The split-window method was used to calculate this response time. On the computer, two windows opened side by side: one for WhatsApp/ Messenger and another for the current time with milliseconds website [35]. Both windows were recorded on one screen with multiple numbers of requests by the user. Once the experiment finished, data was extracted manually by pausing the video frame by frame (with milliseconds).

2) AUTOMATED TESTING

To test deployments on a large scale, requests need to be sent from Facebook/WhatsApp users, but this will be possible for only 5-10 users in real time because it is challenging to create more than 5000 Facebook accounts and send requests concurrently. If Twilio/WhatsApp is used to test our platform, more than 5000 phone numbers need to be bought, each number priced at $1.15 per month and each message priced at $0.0079 [36]. These requests need to be performed multiple times for multi-threading. To overcome the above issues, the chatbot was tested with simulated users. Next, random 10-digit numbers with US country codes were generated, as shown in Algorithm 1. The chatbot webhook is designed to respond to any phone number with a US country code, irrespective of whether it is a valid number. With this, incoming and outgoing messages can be seen in the database and connection logs. To stress the chatbot, multiple requests were sent simultaneously with different threads (max workers in the algorithm). An external webhook was used to measure performance assessment in this approach. Random users were simulated using scripting, and requests were sent to the case study platform.

3) SEMI-AUTOMATED TESTING

The DM cloud chatbot experimented with semi-automated testing, where real user messages were used alongside automated scripts to calculate response time one way at a time. Initially, messages were sent to the Cloud chatbot with the script from the real user's cellphone number, and their one-way response time was recorded. Subsequently, another automated script was employed to send messages from the chatbot to the real users, and their response time was recorded. Finally, these two response times were combined.

B. PERFORMANCE METRICS

Reviewing cloud applications and heterogeneous cloud systems, we consider response time, connection loss, and throughput performance metrics in this work. The assessment methodology involved two deployments discussed in the experimental section: on-premises servers and cloud servers, which were used to obtain these metrics. To assess the performance, a required number of random users was simulated using a script, and requests were sent to the chatbot via the webhook endpoint. Given that chatbots often need to respond to multiple users simultaneously, we employed multi-threading to evaluate this aspect.

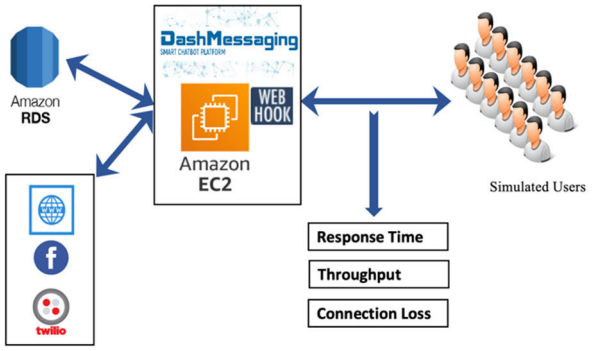


FIGURE 1. Assessment methodology architecture.

The case study chatbot responds through different platforms such as Facebook, WhatsApp, SMS, etc. The case study platform chatbot does have webhooks for these platforms and hosted medical applications on these platforms. On Facebook, users can communicate through Messenger, and for SMS and WhatsApp, Twilio, a consumer engagement party was used.

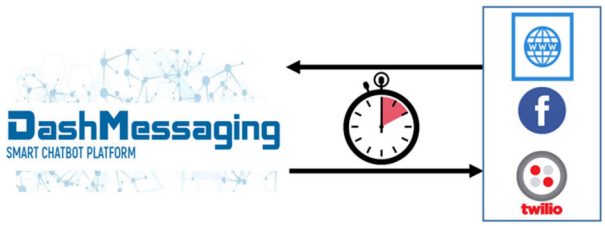


FIGURE 2. Response time.

Response Time is the time taken to receive a response to a given request. Several combinations of requests were sent to both the on-premises server and the cloud server, and the below equation was used to calculate the average response time. Most often, response time is affected by network bandwidth, the volume of users and requests submitted, and average think time.

1) RESPONSE TIMES

Response time [37] refers to the time the Enterprise Server takes to return the results of a request to the user as shown in Figure 2. The response time is affected by network bandwidth, the number of users, the number and type of requests submitted, and the average response time. The faster the response time, the more requests per minute are being processed. However, as the number of users on the system increases, the response time also increases, even though the number of requests per minute declines, as shown in the below equation where n is the total number of users and r is the number of concurrent users.

In this work, multiple simulated user requests were sent to the chatbot, and their response time was recorded to get back to the users with a reply. The different sizes of multi-threads were used while sending requests. Response time is crucial in chatbot-based applications since users will expect a chatbot response quickly, and it is understandable that nobody wants to wait longer to get a reply.

$$\text{Average Response Time} = \frac{\text{Sum of All Response Times}}{\text{Number of Iterations}}$$

2) THROUGHPUT

Throughput measures how many user requests or interactions a chatbot can handle in a given time. It is an important performance metric for chatbots, as it can impact the user experience and overall effectiveness of the chatbot. It is an important performance metric for chatbots, as it can affect the user experience and overall effectiveness of the chatbot. Several factors can impact the throughput of a chatbot, including the complexity of user requests, the responsiveness of the chatbot, and the available computing resources. Generally, a higher throughput is desirable, as the chatbot can handle more user requests and provide more timely responses.

Throughput is calculated as requests/units of time. The time is calculated from the first sample's start to the last sample's end. This includes any intervals between samples, as it is supposed to represent the load on the server [38].

In this chatbot model, total time refers to serving all the requested simulated users. Also, we have used multi threads, which are concurrent simulated users. This paper considers throughput per 1000 requests in calculations. Throughput in this assessment will provide a rate at which a chatbot completes its task. The seconds per 1000 requests are the metrics for throughput used in this work.

$$\text{Throughput} = \frac{\text{Total time taken for job}}{\text{Total requests}}$$

3) CONNECTION LOSS

The percent of simulated users who define connection loss in this work did not receive a response from the chatbot, as shown in Figure 3. Also, many concurrent users will expect an answer from the chatbot without any loss. So, Connection loss will provide a proper estimation of a chatbot with a huge number of concurrent users.

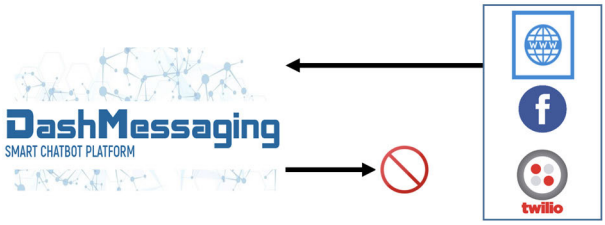


FIGURE 3. Connection loss.

V. EXPERIMENTAL SETUP

To assess chatbot performance across cloud and on-premises environments, we utilized the case study platform Dash Messaging Chatbot application developed at The University of

Algorithm 1 Algorithm to Send User Requests to Chatbot

```
FUNCTION random_number ():
    GENERATE RANDOM USERS
FUNCTION webhook_request(number):
    SET response TO requests.post('https://example.com/twilio', headers=headers, data=data)
    SET output TO response.elapsed.total_seconds()
    OPEN (f'output.csv', 'a').write(str(output) + '\n').close()
    RETURN response.status_code
SET number_list TO []
FOR i IN range(1000):
    number_list.append(random_number())
FUNCTION main ():
    SET threads TO []
    WITH ThreadPoolExecutor(max_workers=100) as executor:
        FOR number IN number_list:
            threads.append(executor.submit(webhook_request, number))
Main()
```

Texas at San Antonio [38] to aid users in quitting smoking by delivering inspirational messages over 5 to 6 months. Thus, the Dash Messaging platform serves as a representative heterogeneous cloud environment for assessing chatbot performance in this study. The on-premises server is housed at The University of Texas at San Antonio, whereas Amazon Web Services (AWS) serves as the chosen platform for cloud deployments. This section explains experimental setups for both on-premises server and cloud implementation, including all the hardware and software specifications, as shown in Table 4.

In experimental setup, including both on-premises-server and cloud environments was deliberate and aimed at capturing a complete view of chatbot performance under different infrastructural paradigms. While it may seem that on-premises and cloud environments share similarities, the distinction arises from the underlying microservice architecture employed in the cloud setting. Microservices offer a modular and scalable approach to software design, enabling the deployment of individual, independently scalable components in the cloud. This architecture inherently affects communication patterns, load distribution, and resource utilization, influencing the performance characteristics of the chatbot system.

Moreover, the differences in database connections between the on-premises and cloud environments contribute to the experimental diversity. The cloud environment often involves database services that are inherently scalable and managed by the cloud provider, influencing the way data is stored, retrieved, and processed compared to a traditional on-premises database setup. These variations introduce unique challenges and opportunities regarding communication efficiency, response times, and overall system adaptability.

In essence, by including both on-premises server and cloud environments, the target was to capture the impact of distinct architectural and infrastructure choices on chatbot communication performance. This allows for a better understanding of how these environments handle user requests, process data, and interact with various components, contributing to a more realistic performance assessment of the chatbot system in heterogeneous settings.

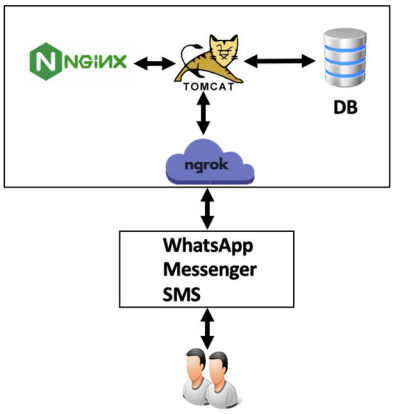


FIGURE 4. Chatbot on-premises server deployment setup.

The On-premises server has the following setup, as shown in Figure 4. The chatbot application is deployed on the Windows server using Tomcat with an MSSQL database connected to it. With the help of Nginx [39], the connection was exposed to the outside world. We connected our chatbot to Facebook Messenger and SMS/WhatsApp (Twilio) using webhooks.

Webhooks are widely used to connect chatbots to messaging platforms such as Facebook Messenger and WhatsApp. Webhooks provide a way for chatbots to receive real-time updates and messages from these platforms. When a user sends a message or performs an action, the platform sends an HTTP request to a webhook URL provided by the chatbot, triggering the chatbot to process the request and respond accordingly.

Webhooks advantages for chatbot integration as listed below:

- Real-time Communication: Webhooks enable real-time communication between the chatbot and the platform so that the chatbot receives updates and messages instantly, allowing for timely responses.
- Event-Driven Architecture: With webhooks, the chatbot is notified only when an event occurs, such as a new message or a user action which reduces the need for continuous polling, making the integration more efficient.
- Asynchronous Processing: Webhooks allow the chatbot to handle requests asynchronously which receive multiple requests simultaneously and process them independently, enabling scalability and efficient resource utilization.
- Flexibility: Webhooks provide flexibility in handling incoming requests and defining custom logic for processing them.

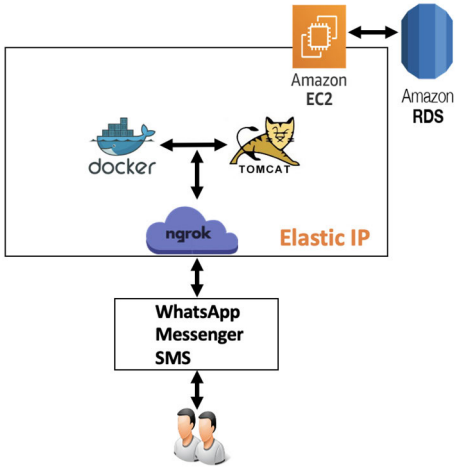


FIGURE 5. Chatbot cloud deployment setup.

Cloud is a dense framework of virtualized servers with software installed to serve the client's needs. The servers may be placed in massive data centers or clusters separated across geographies and connected with high-speed Internet. Cloud computing delivers services installed on the cloud infrastructure and resources to the end client in a scalable and pay-per-use manner. It is a distributed system employing utility computing to provide services [8]. Cloud setup is in AWS EC2 instance and has the following specifications, shown in Table 4 and followed by features.

AWS EC2 T2 medium instance features include Burstable CPU, governed by CPU Credits, consistent baseline performance, Low-cost general purpose instance type, Balance of computing, memory, and network resources. This instance is with the Linux operating system (Ubuntu) and deployed our chatbot application using Tomcat microservices (Docker) and connected to AWS RDS MSSQL database and attached elastic IP (Don't want to change IP in case of start/stop). We have used Ngrok to expose the connection to the outside world, like on-premises. Like on-premises servers, webhooks were used to connect messaging platforms, as shown in Figure 5.

TABLE 4. Server specifications.

Deployment server	On-premises server	Cloud server
Hardware	DELL	Haswell E5-2676 v3 or
Processor	Intel® Xeon® CPU E5 -26200@2.00GHz	Up to 3.3 GHz Intel Xeon Scalable processor
vCPU	4	2
Memory	4 GiB	4 GiB
Operating system	Windows	Linux

The AWS RDS T2 tier database was used for this experiment, and the specifications are shown in Table 4. T2 instances are burstable general-purpose performance instances that provide a baseline level of CPU performance with the ability to burst above the baseline. T2 instances are a good choice for various database workloads, including micro-services and test and staging databases. CPU Credits govern the baseline performance and ability to burst. T2 instances receive CPU Credits continuously at a set rate depending on the instance size, accumulating CPU Credits when idle and consuming CPU credits when active. Features and specifications of the T2 medium model include:

- High-frequency Intel Xeon processors
- Burstable CPU, governed by CPU Credits, and consistent baseline performance.
- Balance of compute, memory, and network resources
- Two vCPU, 24 CPU Credits/hour, and 4 GiB Memory
- Low to Moderate Network performance

VI. RESULTS

This section evaluated the performance of both On-premises server and Cloud Chatbot deployments with two testing methodologies, as mentioned in the previous sections.

A. HUMAN TESTING

The average response times for the On-Premises server and Cloud Chatbot over three communication channels such as Messenger, WhatsApp, and SMS are shown in Figure 6.

Firstly, 100 user messages were sent one at a time to the On-Premises server Chatbot over WhatsApp, resulting in an average response time of 1.61 seconds, with a minimum of 0.89 seconds and a maximum of 2.78 seconds. Subsequently, testing with Messenger yielded an average response time of approximately 2.2 seconds, with a minimum of 1.53 seconds and a maximum of 2.87 seconds. Similarly, the cloud chatbot over the WhatsApp channel was tested, achieving an average response time of 1.75 seconds, with a minimum of 1.42 seconds and a maximum of 2.78 seconds. Next, the cloud chatbot over Messenger was tested, resulting in an average response time of 2.14 seconds, with a minimum of 1.92 seconds and a maximum of 2.49 seconds.

Finally, the cloud chatbot over SMS was tested, receiving an average response time of 1.8 seconds, with a minimum

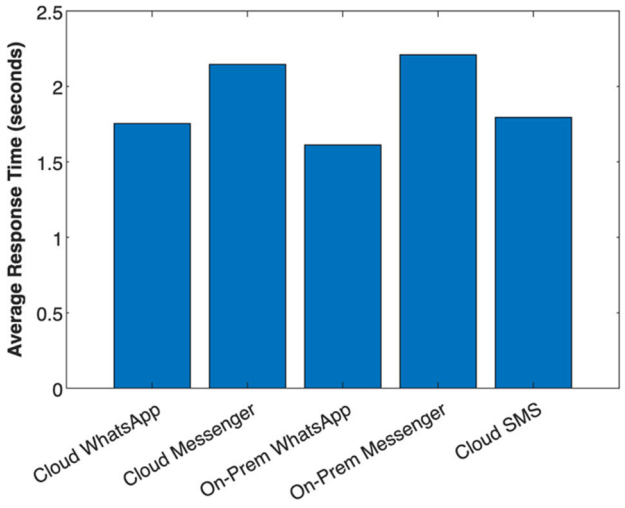


FIGURE 6. Average response time.

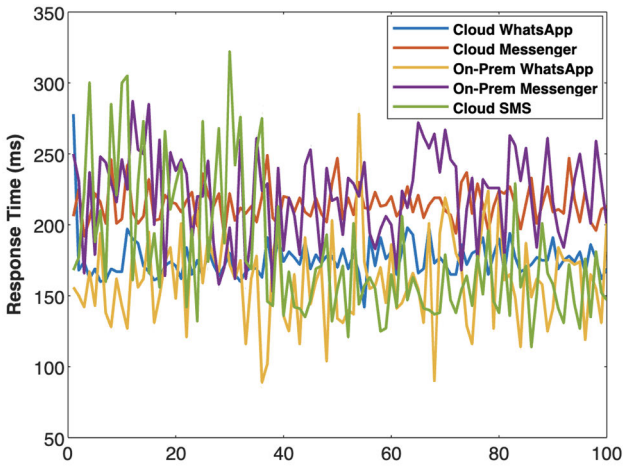


FIGURE 7. Response times of 100 user messages.

of 1.14 seconds and a maximum of 3.22 seconds, as shown in Table 5. The average response of these five scenarios is depicted in Figure 6, while the minimum and maximum values are listed in Table 5. The average response times of Messenger with both On-Premises server and Cloud chatbots are almost identical, at 2.2 and 2.14 seconds, respectively. Similarly, the average response times of WhatsApp with the On-Premises server and Cloud Chatbot and SMS with Cloud Chatbot response times were similar, at 1.6, 1.7, and 1.8 seconds, respectively. In these three scenarios, Twilio is the common channel.

TABLE 5. Response time (in seconds) for 100 messages.

	On-Premises Server		Cloud Server		
	WhatsApp	Messenger	WhatsApp	Messenger	SMS
Average	1.61	2.20	1.75	2.14	1.79
Min	0.89	1.53	1.42	1.92	1.14
Max	2.78	2.87	2.78	2.49	3.22

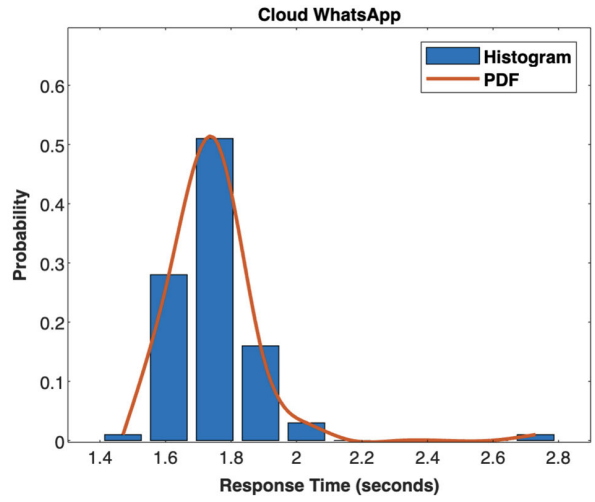
Figure 7 illustrates the distribution of response times for the respective chatbots over WhatsApp, Messenger, and SMS. Most response times in these five scenarios fell within the range of approximately 1.5 to 2.5 seconds. Additionally, the average response times across the above five scenarios were also closely aligned.

Figure 8 presents histogram plots, a commonly utilized tool for visualizing the distribution of response times in a chatbot system. Histograms depict the frequency of response times within different bins or ranges, with each bar's height representing the probability of response times falling within that range. Additionally, a probability distribution plot (smooth line atop histogram bars) is employed to illustrate the distribution of chatbot response times. The X-axis denotes response time in seconds, while the Y-axis indicates the frequency or probability of those values, reflecting the analysis of the five scenarios with 100 user messages. In Figure 8(a), about Cloud WhatsApp, most messages exhibit response times between 1.6 and 1.9 seconds, with over half of them approaching around 1.8 seconds. In Figure 8(b), concerning Cloud Messenger, most messages display response times ranging from 2 to 2.3 seconds. Figure 8(c) depicts On-Premises server WhatsApp, revealing that over 90% of messages exhibit response times varying between 1 and 2 seconds. In Figure 8(d), corresponding to the On-Premises server Messenger, total message response times span between 1.6 and 2.8 seconds, with over 25% of messages clocking in at 2.3 seconds. Lastly, in Figure 8(e) depicting Cloud SMS, over half of the message response times fall within the 1.4 to 1.6 seconds range.

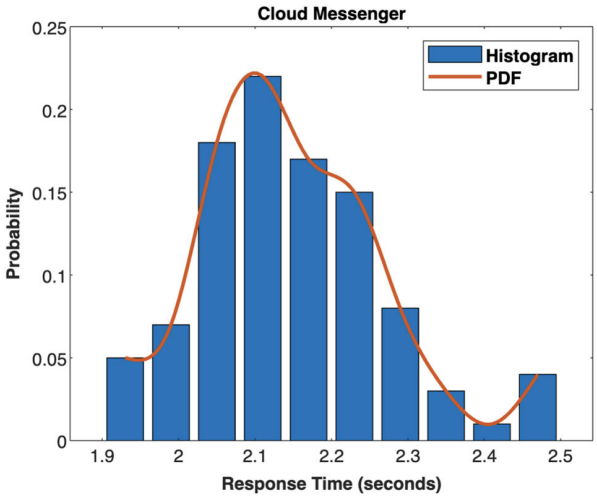
B. AUTOMATED TESTING

This section covers communication performance assessment of on-premises servers and cloud chatbots with automated and stress testing. Figure A shows the response time of the on-premises server versus the cloud comparison. First sent, 1000 simulated users separately, resulting in an average response time of 390 milliseconds with the on-premises server and 136 milliseconds with the cloud server. Thus, the cloud is three times faster than the on-premises server. The cloud took 136 seconds to process all 1000 users and one user at a time, and at the same time, the on-premises server took 390 seconds to process all 1000 users and one user at a time.

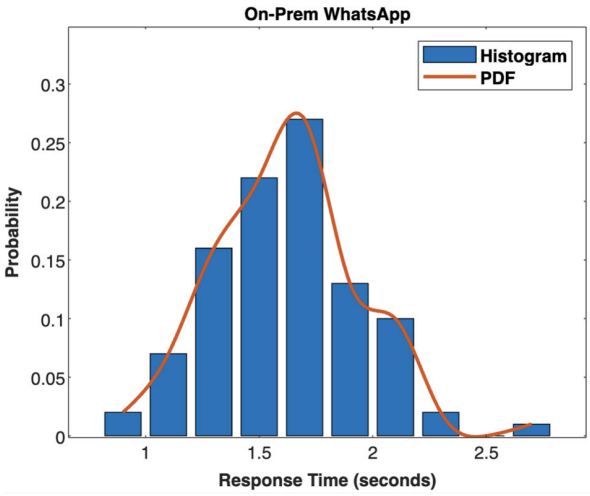
Next, with 100 threads (each sending 100 users at a time), we observed that the on-premises server took an average of 1578 milliseconds to process 100 requests and 15780 milliseconds to process all 1000 requests with 100 threads. Similarly, the cloud server took an average of 997 milliseconds to process each request and 9974 milliseconds to process all 1000 requests with 100 threads. Subsequently, with 200 threads, the on-premises server took an average of 2869 milliseconds to process each request and 14345 milliseconds to process all 1000 requests. Conversely, the cloud



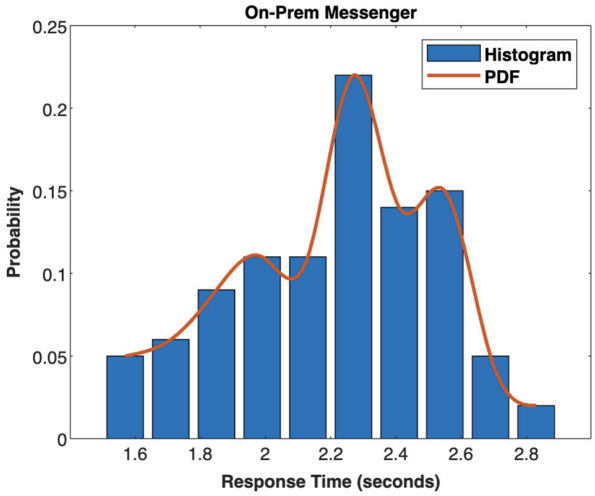
(a)



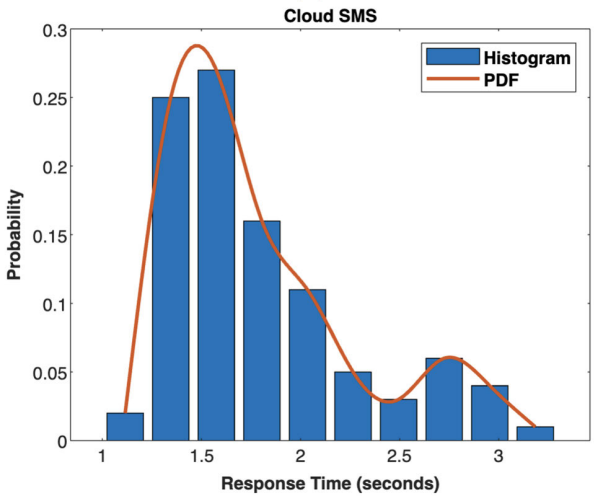
(b)



(c)



(d)



(e)

FIGURE 8. Histogram and probability distribution function of 100 user messages response times.

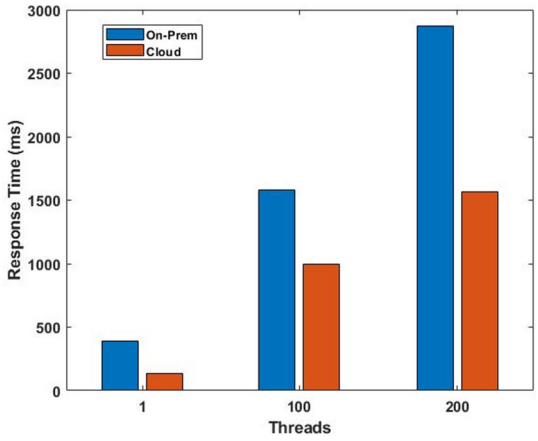


FIGURE 9. Response time of different threads.

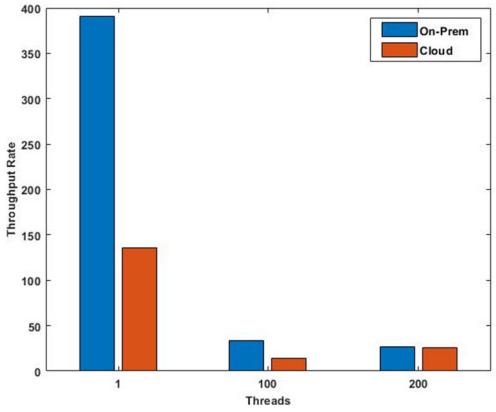


FIGURE 10. Throughput rate of cloud and on-premises chatbot for 1000 requests.

server took an average of 1566 milliseconds to process each request and 7830 milliseconds to process all 1000 requests with 100 threads, as illustrated in Figure 9.

We then calculated the throughput for cloud and on-premises server deployments, measured as the time to complete 1000 requests. Initially, we conducted experiments with one thread and gradually increased it to 100 threads and then to 200 threads for both server types. With only one thread, the on-premises server took 390 seconds to serve 1000 requests, while the cloud server completed the same setup in 136 seconds. When we repeated the experiment with 100 threads, the on-premises server took approximately 33 seconds, whereas the cloud server only required 14 seconds. Finally, with 200 threads, both servers completed the task in roughly the same time, approximately 26 seconds for the on-premises server and 25 seconds for the cloud chatbot deployments, as illustrated in Figure 10.

The above experiment repeated for 250 threads, 500 threads, and 1000 threads on cloud deployments. Figure 11 shows these results, and the throughput rate gradually decreases with the increase of threads.

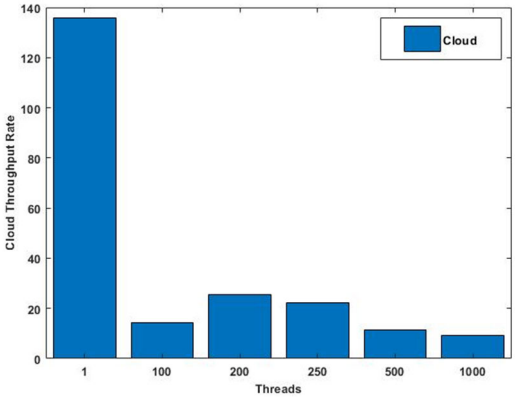


FIGURE 11. Throughput rate of cloud chatbot deployment for 1000 requests.

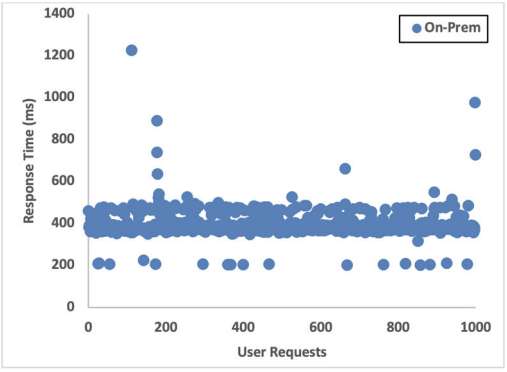


FIGURE 12. The scatter of response times for On-Prem.

This study utilized scatter plots to examine the response times of both on-premises server and cloud chatbot deployments, considering one simulated user at a time and a total of 1000 users. Figure 12 illustrates the scatter of response times for all 1000 users on the on-premises server, and Figure 13 displays the distribution of response times for all 1000 users on the cloud server, both processed individually.

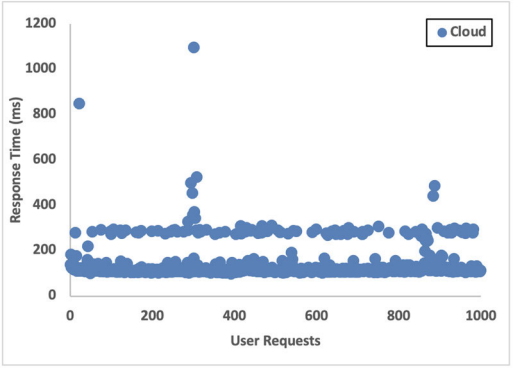


FIGURE 13. The scatter of response times for cloud.

In our analysis, Figure 12 provides a visualization of the response times exhibited by the on-premises server,

illustrating four distinct and concentrated lines. These lines correspond to the four Virtual Central Processing Units (vCPUs) deployed in the on-premises server environment. Each vCPU independently handles user requests, contributing to the observed pattern of four concentrated response timelines. This insight sheds light on the parallel processing capability of the on-premises server, highlighting its ability to efficiently distribute workloads across multiple vCPUs. Similarly, Figure 13 shows a comparable pattern for the AWS cloud server, which has two virtual CPUs. The two lines of concentrated response times in this figure align with the dual vCPU configuration of the AWS cloud server, highlighting the parallel processing nature of the cloud-based infrastructure. These observations provide a nuanced understanding of the impact of server architecture on response times, underscoring the role of virtual CPU configurations in shaping the performance characteristics of chatbot systems in diverse computing environments.

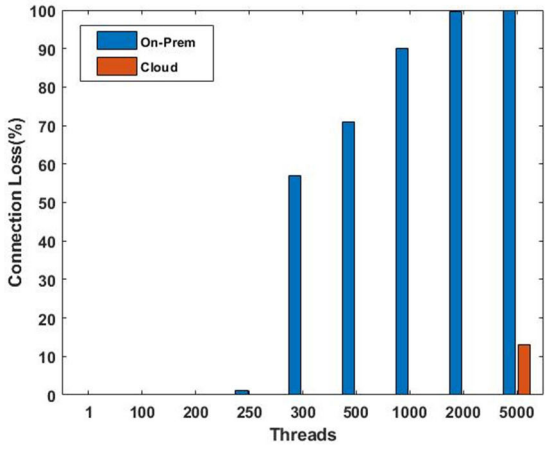


FIGURE 14. Connection loss percentage comparison.

Lastly, we stressed on-prem and cloud servers with many simulated users and loads (threads). The on-premises server successfully processed all 1000 users with 200 threads. However, with 1000 users tested with 250 threads, it processed 989 users but lost 11 user requests. As simulated users and threads increased, the on-premises server experienced a linear decrease in successful user requests. For instance, it only processed seven user requests out of 5000 users with 5000 threads, resulting in a 99.86% loss. Conversely, the cloud server processed 2000 simulated users with 2000 threads with a 100% success rate and maintained an 87% connection rate for 5000 simulated users with 5000 threads. Figure 14 depicts the connection loss percentage of both on-premises and cloud servers with various requests. These successful connections for on-premises and cloud are shown in Table 6.

C. AUTOMATED TESTING SMS WITHOUT CHANNEL DELAY

In this section, the DM cloud chatbot underwent testing with SMS, this time utilizing real user messages.

TABLE 6. Successful user connections for multi-threading.

Total Simulated Users	Threads	On Prem	Cloud
1000	1	1000	1000
1000	100	1000	1000
1000	200	1000	1000
1000	250	989	1000
1000	300	430	1000
1000	500	29	1000
1000	1000	10	1000
2000	2000	7	2000
5000	5000	7	4352

Initially, 100 messages were sent to the Cloud chatbot from actual users' cellphone numbers. Their one-way response times were recorded upon message delivery to the chatbot, averaging approximately 585 milliseconds. Subsequently, 100 messages were sent from the chatbot to the actual users via another automated script, yielding an average response time of about 345 milliseconds. Consequently, the average total response time for the roundtrip amounted to 1660 milliseconds. The Histogram and Probability Distribution Function of 100 User Messages Response Times are shown in Figure 15.

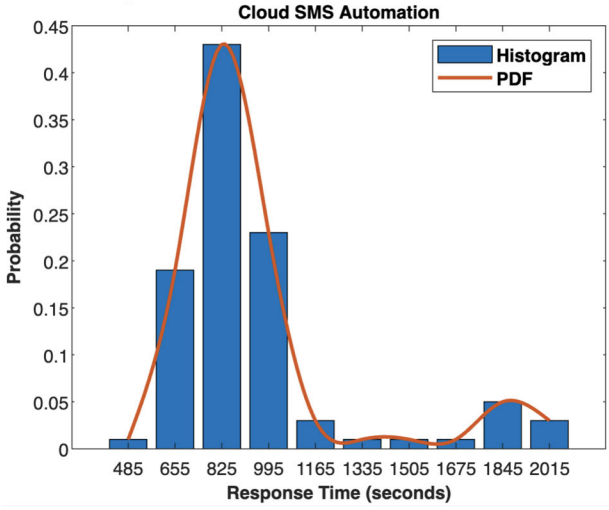


FIGURE 15. Histogram and probability distribution function of 100 user messages response times.

VII. DISCUSSION

This section discusses the three sets of results from the previous section. In human testing, around 1.6 to 2.2 seconds of total response time was observed, including channel and chatbot delays. All these experiments were conducted with real users. Analyzing the probability distribution plot lets us identify the typical response times for the chatbot and any outliers or patterns in the data. This can help to optimize the chatbot's performance and improve the user experience by identifying areas for improvement, such as optimizing the

chatbot's natural language processing algorithms or increasing computing resources to reduce response times.

In the PDF plots of the five scenarios discussed in section VI, we observe a single peak, indicating a narrow range of response times in the chatbot system. This suggests efficiency and effectiveness in processing user input and generating responses. In the automated testing, simulated users without any channels were used. We observed 390ms of average response time for the on-premises server chatbot and 136ms of average response time for the cloud chatbot. Both chatbots underwent testing with different thread combinations to calculate throughput, revealing a decrease in throughput with the increasing number of threads. And experimented with stress tests; an observed chatbot can handle around 4300 parallel connections, and an on-premises server chatbot can handle 250 parallel connections.

In the semi-automated testing, we used script instead of real users by eliminating channel delay. The analysis conducted in this paper unveiled an average roundtrip response time of 1.6 seconds. It is imperative to underscore those chatbots tailored for specific tasks, such as information retrieval or customer support, that must meet a designated response time benchmark to ensure effectiveness. As suggested by Nursetyo et al. [40], an ideal response time for such chatbots falls in the range of 2-5 seconds. This timeframe holds significance as it not only aligns with user expectations for prompt interactions but also guarantees the chatbot's efficient processing of user requests and delivery of accurate responses. Maintaining response times within this optimal range serves as a key performance indicator, directly impacting user satisfaction and the overall usability of the chatbot in fulfilling its designated tasks.

Our analysis revealed significant metrics such as response time, where we observed an average roundtrip time of 1.6 seconds, aligning with the recommended 2-5 seconds for effective task-specific chatbots [40]. Additionally, metrics like connection loss and throughput were assessed, offering quantifiable insights into the system's reliability and scalability.

Responses from real users highlighted aspects of the chatbot's interaction such as clarity, relevance, and overall user satisfaction. This qualitative feedback gathered through real-time testing, provided fine insights into the user experience, shedding light on subjective aspects that metrics alone might not capture. This thorough understanding enables us to identify strengths and areas for improvement, ensuring that the chatbot meets quantitative benchmarks and delivers a positive and engaging user experience.

Scatter plots, bar charts, and histogram representations were employed to visually represent the findings. Scatter plots illustrate the relationship between various communication metrics, offering insights into patterns and trends. Bar charts were employed to compare performance across several aspects of the chatbot system, highlighting strengths and areas for improvement. Additionally, histogram representations allowed for exploring the distribution of key performance indicators, offering a granular perspective on the system's behavior. Ranci et al. [42] listed descriptive statistics as representation methods that visually integrate multiple datasets to contextualize the data and improve reader understanding. The approach aimed to quantify the system's communication proficiency and present a visually informative analysis, enhancing the interpretability and applicability of this study.

We used DashMessaging [38] as a representative environment for a heterogeneous cloud-based chatbot platform, but any chatbot can be assessed through the performance assessment methodology presented in this paper. This chatbot uses AWS cloud, but one can use any cloud provider. Identifying potential biases and limitations is crucial for understanding the scope and generalizability of research findings. In the context of a study evaluating the performance of cloud chatbot systems in a heterogeneous environment, we have observed some biases during this assessment.

The study deliberately avoids reliance on a specific dataset or exclusive user group to handle sample bias. Also, this study deliberately incorporates diversity in operating systems, encompassing both Windows and Linux. Additionally, the performance assessment extends to different deployment environments, covering both on-premises and cloud settings. This multifaceted approach ensures assessing the chatbot's performance, reducing the risk of biases associated with specific operating systems or deployment platforms. This assessment methodology applies to Rule-based and Intent-based chatbots. It ensures insights that transcend specific functionalities and contributes to a universal understanding of cloud chatbot performance in heterogeneous environments to observe task-specific bias.

This study also incorporates real-user testing alongside simulated users. Real users offer authentic insights into natural interactions, while simulated users offer controlled scenarios. This ensures a balanced and thorough assessment of the chatbot's performance in a heterogeneous environment that observes simulation bias.

VIII. CONCLUSION

Navigating the intricacies of heterogeneous environments is paramount in assessing chatbot performance across diverse methodologies. This study scrutinizes chatbots deployed on both on-premises and cloud server platforms, shedding light on their effectiveness through various assessment techniques. The performance assessment methodology involved statistical analysis, allowing for a thorough examination of the system's effectiveness and limitations.

Since webhooks are a widely used integration method to connect chatbots to messaging platforms such as Facebook Messenger and WhatsApp, we have tested our chatbot with this webhook integration, and our assessment encompasses human and automated testing methodologies, providing the understanding of chatbot performance.

During human testing, the response times of messaging platforms like Messenger and WhatsApp were analyzed, revealing insights into both on-premises and cloud-based chatbot systems. Messenger demonstrated average response times of 2.2 and 2.1 seconds for on-premises servers and cloud deployments, respectively, while WhatsApp exhibited 1.6 and 1.7 seconds for the same configurations. These metrics, inclusive of channel delays and human reading errors, offer a fine perspective on chatbot performance in real-world scenarios.

Automated testing further delved into the efficiency of chatbots, distinguishing between scenarios with and without channel delays. The cloud server outperforms the on-premises server by processing each user request in an average response time of 139ms. The on-premises server took almost three times more than the cloud server. Stress-testing revealed the robustness of cloud-based deployments, handling approximately 4300 parallel connections compared to the on-premises server's capacity of 250 users. Additionally, the absence of channel delays in automated testing yielded insights into the cloud server's average response time of 1.66 seconds, with scatter plot analysis indicating a correlation between hardware specifications and response times.

REFERENCES

[1] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: Yesterday, today, and tomorrow," in *Present and Ulterior Software Engineering*. Berlin, Germany: Springer, 2017, pp. 195–216, doi: 10.1007/978-3-319-67425-4_12.

[2] N. Rosruen and T. Samanchuen, "Chatbot utilization for medical consultant system," in *Proc. 3rd Technol. Innov. Manag. Eng. Sci. Int. Conf. (TIMES-iCON)*, Dec. 2018, pp. 1–5, doi: 10.1109/TIMES-iCON.2018.8621678.

[3] D. Inupakutika, M. Nadim, G. R. Gunnam, S. Kaghyan, D. Akopian, P. Chalela, and A. G. Ramirez, "Integration of NLP and speech-to-text applications with chatbots," in *Proc. IST Int. Symp. Electron. Imag. Sci. Technol.*, Jun. 2021, vol. 33, no. 3, pp. 6–35, doi: 10.2352/issn.2470-1173.2021.3.mobmu-035.

[4] (2024). *IBM Watson*. Accessed: Feb. 4, 2024. [Online]. Available: https://www.ibm.com/

[5] *DialogFlow | Google Cloud*. Accessed: Feb. 4, 2024. [Online]. Available: https://cloud.google.com/dialogflow

[6] *Facebook*. Accessed: Feb. 4, 2024. [Online]. Available: https://wit.ai/

[7] M. Yan, P. Castro, P. Cheng, and V. Ishakian, "Building a chatbot with serverless computing," in *Proc. 1st Int. Workshop Mashups Things APIs*, Dec. 2016, pp. 1–4, doi: 10.1145/3007203.3007217.

[8] A. Gajbhiye and K. M. P. Shrivastva, "Cloud computing: Need, enabling technology, architecture, advantages and challenges," in *Proc. 5th Int. Conf. Confluence Next Gener. Inf. Technol. Summit*, Sep. 2014, pp. 1–7, doi: 10.1109/CONFLUENCE.2014.6949224.

[9] L. Malhotra, D. Agarwal, and A. Jaiswal, "Virtualization in cloud computing," *J. Inf. Technol. Softw. Eng.*, vol. 4, no. 2, pp. 1–3, Jan. 2014, doi: 10.4172/2165-7866.1000136.

[10] S. M. Jang, W. H. Choi, and W. Y. Kim, "Client rendering method for desktop virtualization services," *ETRI J.*, vol. 35, no. 2, pp. 348–351, Apr. 2013, doi: 10.4218/etrij.13.0212.0213.

[11] M. Ben-Yehuda, M. D. Day, Z. Dubitzky, M. Factor, N. Har'El, A. Gordon, A. Liguori, O. Wasserman, and B. A. Yassour, "The turtles project: Design and implementation of nested virtualization," in *Proc. 9th USENIX Symp. Operating Syst. Design Implement.*, 2010, pp. 1–14.

[12] C. Pahl, "Containerization and the PaaS cloud," *IEEE Cloud Comput.*, vol. 2, no. 3, pp. 24–31, May 2015, doi: 10.1109/MCC.2015.51.

[13] G. Pék, L. Buttyán, and B. Bencsáth, "A survey of security issues in hardware virtualization," *ACM Comput. Surv.*, vol. 45, no. 3, pp. 1–34, Jun. 2013, doi: 10.1145/2480741.2480757.

[14] A. R. Shabaitah. (Jan. 2014). *Server-Based Desktop Virtualization*. [Online]. Available: https://scholarworks.rit.edu/cgi/viewcontent.cgi?article=8844&context=theses

[15] J. Cito, G. Schermann, J. E. Wittern, P. Leitner, S. Zumberi, and H. C. Gall, "An empirical analysis of the Docker container ecosystem on GitHub," in *Proc. IEEE/ACM 14th Int. Conf. Mining Software Repositories (MSR)*, May 2017, pp. 323–333, doi: 10.1109/MSR.2017.67.

[16] (2024). *Overview of the Get Started Guide*. Accessed: Feb. 6, 2024. [Online]. Available: https://docs.docker.com/get-started/

[17] D. Jaramillo, D. V. Nguyen, and R. Smart, "Leveraging microservices architecture by using Docker technology," in *Proc. SoutheastCon*, Mar. 2016, pp. 1–5, doi: 10.1109/secon.2016.7506647.

[18] P. Mell and T. Grance, "The NIST definition of cloud computing," U.S. Dept. Commerce, Nat. Inst. Standards Technol., Gaithersburg, MD, USA, Tech. Rep. Special Publication 800-145, Jan. 2011, doi: 10.6028/nist.sp.800-145.

[19] A. V. Papadopoulos, L. Versluis, A. Bauer, N. Herbst, J. V. Kistowski, A. Ali-Eldin, C. L. Abad, J. N. Amaral, P. Tuma, and A. Iosup, "Methodological principles for reproducible performance evaluation in cloud computing," *IEEE Trans. Softw. Eng.*, vol. 47, no. 8, pp. 1528–1543, Aug. 2021, doi: 10.1109/TSE.2019.2927908.

[20] G. Atas and V. C. Gungor, "Performance evaluation of cloud computing platforms using statistical methods," *Comput. Electr. Eng.*, vol. 40, no. 5, pp. 1636–1649, Jul. 2014, doi: 10.1016/j.compeleceng.2014.03.017.

[21] N. Khanghahi and R. Ravanmehr, "Cloud computing performance evaluation: Issues and challenges," *Int. J. Cloud Comput., Services Archit.*, vol. 3, no. 5, pp. 29–41, Oct. 2013, doi: 10.5121/ijccsa.2013.3503.

[22] M. Rak, A. Cuomo, and U. Villano, "Cost/performance evaluation for cloud applications using simulation," in *Proc. Workshops Enabling Technol., Infrastructure Collaborative Enterprises*, Jun. 2013, pp. 152–157, doi: 10.1109/WETICE.2013.36.

[23] V. Stantchev, "Performance evaluation of cloud computing offerings," in *Proc. 3rd Int. Conf. Adv. Eng. Comput. Appl. Sci.*, Oct. 2009, pp. 187–192, doi: 10.1109/ADVCOMP.2009.36.

[24] A. Bahga and V. K. Madisetti, "Performance evaluation approach for multi-tier cloud applications," *J. Softw. Eng. Appl.*, vol. 6, no. 2, pp. 74–83, Jan. 2013, doi: 10.4236/jsea.2013.62012.

[25] M. S. Aslanpour, S. S. Gill, and A. N. Toosi, "Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research," *Internet Things*, vol. 12, Dec. 2020, Art. no. 100273, doi: 10.1016/j.iot.2020.100273.

[26] M. Malawski, K. Figiela, A. Gajek, and A. Zima, "Benchmarking heterogeneous cloud functions," in *Proc. Eur. Conf. Parallel Process.* (Lecture Notes in Computer Science), 2018, pp. 415–426, doi: 10.1007/978-3-319-75178-8_34.

[27] K. Figiela, A. Gajek, A. Zima, B. Obrok, and M. Malawski, "Performance evaluation of heterogeneous cloud functions," *Concurrency Comput., Pract. Exper.*, vol. 30, no. 23, p. e4792, Aug. 2018, doi: 10.1002/cpe.4792.

[28] X. Tang, X. Li, and Z. Fu, "Budget-constraint stochastic task scheduling on heterogeneous cloud systems," *Concurrency Comput., Pract. Exper.*, vol. 29, no. 19, p. e4210, Jun. 2017, doi: 10.1002/cpe.4210.

[29] J. Scheuner and P. Leitner, "Function-as-a-service performance evaluation: A multivocal literature review," *J. Syst. Softw.*, vol. 170, Dec. 2020, Art. no. 110708, doi: 10.1016/j.jss.2020.110708.

[30] X. Li, M. A. Salehi, M. Bayoumi, N.-F. Tzeng, and R. Buyya, "Cost-efficient and robust on-demand video transcoding using heterogeneous cloud services," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 3, pp. 556–571, Mar. 2018, doi: 10.1109/TPDS.2017.2766069.

[31] H. Singh, S. Tyagi, P. Kumar, S. S. Gill, and R. Buyya, "Metaheuristics for scheduling of heterogeneous tasks in cloud computing environments: Analysis, performance evaluation, and future directions," *Simul. Model. Pract. Theory*, vol. 111, Sep. 2021, Art. no. 102353, doi: 10.1016/j.simpat.2021.102353.

[32] D. Akopian, R. D. E. Palacios, and S. Kaghyan, "Interactive mobile service for deploying automated protocols," U.S. Patent 10 819 663, Jun. 13, 2010, doi: 10.1145/1837274.1837461.

[33] *Webhook*. Accessed: Feb. 6, 2024. [Online]. Available: https://en.wikipedia.org-/wiki/Webhook

[34] R. Mundlamuri, D. Inupakutika, G. R. Gunnam, S. Kaghyan, and D. Akopian, "Chatbot integration with Google Dialogflow environment for conversational intervention," in *Proc. IST Int. Symp. Electron. Imag. Sci. Technol.*, Jan. 2022, vol. 34, no. 3, pp. 5–206, doi: 10.2352/ei.2022.34.3.mobmu-206.

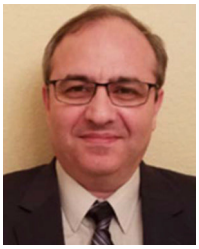
[35] (2024). *Exact Time Clock Now (With Seconds, Milliseconds)*. Accessed: Feb. 6, 2024. [Online]. Available: https://clock.zone/
[36] *SMS Pricing in United States for Text Messaging | Twilio*. Accessed: Feb. 6, 2024. [Online]. Available: https://www.twilio.com/sms/pricing/us
[37] R. B. Miller, "Response time in man-computer conversational transactions," in *Proc. Manag. Requirements Knowl., Int. Workshop*, vol. 1, Dec. 1899, p. 267, doi: 10.1109/afips.1968.149.
[38] H. Koziolek, S. Grüner, and J. Rückert, "A comparison of MQTT brokers for distributed IoT edge computing," in *Proc. Eur. Conf. Softw. Architecture* (Lecture Notes in Computer Science), 2020, pp. 352–368, doi: 10.1007/978-3-030-58923-3_23.
[39] W. Reese, "NGINX: The high-performance web server and reverse proxy," *Linux J.*, vol. 2008, p. 2, Sep. 2008. [Online]. Available: https://dl.acm.org/citation.cfm?id=1412204
[40] A. Nursetyo, D. R. I. M. Setiadi, and E. R. Subhiyakto, "Smart chatbot system for e-commerce assitance based on AIML," in *Proc. Int. Seminar Res. Inf. Technol. Intell. Syst. (ISRITI)*, Nov. 2018, pp. 641–645, doi: 10.1109/ISRITI.2018.8864349.
[41] M. M. Eyada, W. Saber, M. M. El Genidy, and F. Amer, "Performance evaluation of IoT data management using MongoDB versus MySQL databases in different cloud environments," *IEEE Access*, vol. 8, pp. 110656–110668, 2020, doi: 10.1109/ACCESS.2020.3002164.
[42] R. Ren, M. Zapata, J. W. Castro, O. Dieste, and S. T. Acuña, "Experimentation for chatbot usability evaluation: A secondary study," *IEEE Access*, vol. 10, pp. 12430–12464, 2022, doi: 10.1109/ACCESS.2022.3145323.
[43] A. R. Mohammad Forkan, P. Prakash Jayaraman, Y.-B. Kang, and A. Morshed, "ECHO: A tool for empirical evaluation cloud chatbots," in *Proc. 20th IEEE/ACM Int. Symp. Cluster, Cloud Internet Comput. (CCGRID)*, May 2020, pp. 669–672, doi: 10.1109/CCGrid49817.2020.00-26.
[44] M. Zubani, L. Sigalini, I. Serina, L. Putelli, A. E. Gerevini, and M. Chiari, "A performance comparison of different cloud-based natural language understanding services for an Italian e-learning platform," *Future Internet*, vol. 14, no. 2, p. 62, Feb. 2022, doi: 10.3390/fi14020062.
[45] C. F. Salazar, "Using cloud-based chatbot builder in developing pedagogical conversational agent," *Int. J. Eng. Trends Technol.*, vol. 71, no. 7, pp. 301–314, Jul. 2023, doi: 10.14445/22315381/ijett-v71i7p229.
[46] Z. Jiang, M. Rashik, K. Panchal, M. Jasim, A. Sarvghad, P. Riahi, E. DeWitt, F. Thurber, and N. Mahyar, "CommunityBots: Creating and evaluating a multi-agent chatbot platform for public input elicitation," *Proc. ACM Hum.-Comput. Interact.*, vol. 7, pp. 1–32, Apr. 2023, doi: 10.1145/3579469.
[47] H. P. Grice, *Logic and Conversation*. Leiden, The Netherlands: BRILL, 1975, pp. 41–58, doi: 10.1163/9789004368811_003.
[48] Z. Xiao, M. X. Zhou, Q. V. Liao, G. Mark, C. Chi, W. Chen, and H. Yang, "Tell me about yourself: Using an AI-powered chatbot to conduct conversational surveys with open-ended questions," *ACM Trans. Comput.-Hum. Interact.*, vol. 27, no. 3, pp. 1–37, Jun. 2020.

GANESH REDDY GUNNAM received the M.Sc. degree in electrical engineering from The University of Texas Rio Grande Valley (UTRGV), in 2017, and the Ph.D. degree from the Department of Electrical Engineering, The University of Texas at San Antonio (UTSA). His research interests include smart deep-logic chatbot design, development, performance assessment methodology, chatbot cloud deployment, virtualization, and cloud computing.

DEVASENA INUPAKUTIKA received the Ph.D. degree from the Department of Electrical Engineering, The University of Texas at San Antonio (UTSA). She is currently with Samsung Semiconductor Inc. Her research is in the development and performance analysis of systems and methods for enhancing mobility. Her research interests include web and mobile application development, cloud-IoT integration, and deep learning-based wireless LAN indoor positioning systems.

RAHUL MUNDLAMURI received the bachelor's degree in electronics and communications engineering from JNT University, India, in 2014, the master's degree from the University of Houston, Houston, TX, USA, in 2016, and the Ph.D. degree in electrical engineering from The University of Texas at San Antonio, San Antonio, TX, USA. He is currently a Data Engineer at Autodesk, while actively pursuing research in ML-aided customer data platform development. His research interests include neural network-based localization services and data processing.

SAHAK KAGHYAN received the Ph.D. degree in computer science from Russian-Armenian University, in 2014. He is currently a Postdoctoral Research Scientist with The University of Texas at San Antonio (UTSA). His research interests include full-stack web development, mobile application development, conversational AI design and development, machine learning, and software engineering.



DAVID AKOPIAN (Senior Member, IEEE) received the Ph.D. degree from Tampere University of Technology, Finland. He is currently a Professor with The University of Texas at San Antonio (UTSA). Before joining UTSA, he was a Senior Research Engineer and a Specialist with Nokia Corporation. His current research interests include digital signal processing algorithms for communication and navigation receivers, positioning, dedicated hardware architectures, and platforms for software-defined radio and communication technologies for healthcare applications. He is a fellow of U.S. National Academy of Inventors.