A Cloud-based Video Conferencing Application

Naveen Reddy Dyava (nd2794)*, Abhilash Ganga (ag4797)*, Mooizz Abdul (ma4496)*
*Department of Computer Science, Columbia University, New York

***Abstract*—This project introduces a video conferencing application developed on AWS Cloud, utilizing OpenVidu—a technology stack facilitating seamless video call integration into applications. The application, built on top of OpenVidu, accommodates up to four participants, featuring login and registration, session recordings, screen sharing, chat functionality, and post-call meeting analysis.**

The post-call analysis includes user engagement tracking, attendance monitoring, and the host's capability to share meeting data. Notably, the application provides meeting transcripts in English and translations to Hindi, Chinese, Spanish, and German, fostering inclusivity. Leveraging ChatGPT API, the system conducts comprehensive analysis on transcripts, covering sentiment, topic modeling, speaker contributions, keyword extraction, named entity recognition, dialogue flow, diversity of opinions, key quotes identification, contextual analysis, and language complexity.

This project showcases the synergy of AWS Cloud, OpenVidu, and advanced analytics in creating an efficient video conferencing platform. By integrating advanced analytics, it not only improves the collaborative experience but also extracts valuable insights from meetings, contributing to the evolution of virtual collaboration technologies.

***Index Terms*—Video Conferencing, OpenVidu, AWS, Real-time Collaboration, Cloud Computing Applications, User Engagement Tracking, ChatGPT Analysis.**

I. Introduction

In emulation of successful platforms such as Zoom, our endeavor is to establish a cloud-based video conferencing application designed to seamlessly facilitate communication among multiple users. This project is driven by several key objectives:

1) **Feature-Rich Development**: Our primary goal is to craft a video conferencing application replete with advanced features to enhance the user experience.
2) **Cloud Computing Integration**: Leveraging cloud computing resources is integral to our strategy, aiming to achieve scalability and reliability in the application's performance.
3) **Real-Time Communication**: Ensuring seamless real-time communication stands as a fundamental aspect, emphasizing the importance of fluid interaction for users.

The imperative for a video conferencing service lies in its ability to facilitate effective remote communication over vast distances, catering to a diverse user base. This service holds significance for several reasons, as outlined below:

1) **Collaboration**: The incorporation of features such as screen sharing and text chat box functionality promotes collaboration among users.
2) **Versatility**: The application accommodates both one-to-one conversations and large meetings, demonstrating versatility for a myriad of applications.
3) **Global Reach**: Enabling people from across the globe to connect, collaborate, and conduct business without the constraints of extensive travel fosters a globalized approach to communication.
4) **Innovation**: Anticipating future advancements, our platform is poised for potential upgrades.

While established video conferencing platforms like Zoom and Teams have successfully addressed various communication challenges, this project aspires to introduce unique innovations. These innovations include meeting transcription, language translation, and an in-depth analysis of meeting effectiveness and participant engagement. By incorporating these enhancements, our video conferencing application seeks to distinguish itself in the landscape of collaborative platforms.

II. Comprehensive Overview of Application Capabilities

When a user visits the website, it will show a sign-up page. Using email and password, the users can sign up for an account. An OTP based email verification is required for completing the sign up process. Already registered users can directly sign in using their credentials. Successfully signed in users will be shown that landing page where there are two main components - The upper ribbon and the historic meeting details section (occupies majority of the screen). The upper ribbon on the top contains three buttons - Home, Sign Out and Meeting. The historic meeting details section will consists of details about historic meetings the user hosted, referenced by the Meeting Code (Session ID).

The Home button will redirect to the landing page at any time. The sign out button will sign out an user out of the application. The Meeting button will direct the user to page where they will be able to either create a new meeting by generating a new meeting code (creator of the meeting will be by default the host) or join an existing meeting using an existing meeting code which was generated by the hose (in this case the user won't be a host).

By pasting session code and clicking join button, the user will be able to join the call. In the call, the user can mute or unmute their audio and video streams. They also have a button to leave the call. Only host will be able to start recording the session and the users will be notified about it. There is also a chat-box where the users will be able to type text messages.

Note that when a meeting is created and there is no one in the session for more than 120s, the call will be terminated automatically. This waiting period can be configured at the back end. This helps in killing empty meeting and saving computational resources like memory and network bandwidth.

Whenever the call is over, the host of the meeting will get a very detailed email consisting link to the video recording, transcript of the meeting in English (if the meeting was recorded), translation of the transcript to four languages (Hindi, Spanish, German and Chinese), attendance details of the meeting participants and a text document containing comprehensive analysis on transcripts, covering sentiment, topic modeling, speaker contributions, keyword extraction, named entity recognition, dialogue flow, diversity of opinions, key quotes identification, contextual analysis, and language complexity.

The host will be able to view the completed meeting on their home page. The host of the meeting can optionally share the meeting recording, transcripts, etc. to other meeting participants just by the click of a button.

III. Architecture & Data Flow

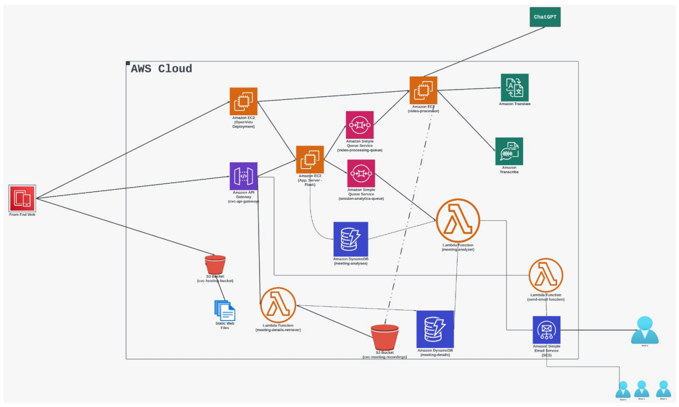

ChatGPT
AWS Cloud
Amazon EC2 (OpenVidu Deployment)
Amazon EC2 (video-processor)
Amazon Translate
Amazon Simple Queue Service (video-processing-queue)
Amazon Transcribe
Amazon API Gateway (cvc-api-gateway)
Amazon EC2 (App. Server - Flask)
Amazon Simple Queue Service (session-analytics-queue)
Front End Web
Amazon DynamoDB (meeting-analyses)
Lambda Function (meeting-analyzer)
S3 Bucket (cvc-hosting-bucket)
Lambda Function (send-email function)
Static Web Files
Lambda Function (meeting-details-retriever)
S3 Bucket (cvc-meeting-recordings)
Amazon DynamoDB (meeting-details)
Amazon Simple Email Service (SES)
Users

Fig. 1. Representation of the mechanism induced by traps on the average drain current.

The following is the list of all the Microservices and APIs that are used in the application:

1) cvc-api-gateway (AWS API Gateway)
2) application-server (AWS EC2 Instance)
3) openvidu-deployment-server (AWS EC2 Instance)
4) video-processor (AWS EC2 Instance)
5) session-analytics-queue (AWS Simple Queue Service)
6) video-processing-queue (AWS Simple Queue Service)
7) meeting-analytics-processor (AWS Lambda Function)
8) meeting-details-retriever (AWS Lambda Function)
9) meeting-emailer (AWS Lambda Function)
10) meeting-details (AWS DynamoDB)
11) meeting-analyses (AWS DynamoDB)
12) cvc-meeting-recordings (AWS S3)
13) cvc-hosting-bucket (AWS S3)
14) AWS Simple Email Service
15) AWS EventBridge Scheduler
16) AWS Transcribe
17) AWS Translate
18) ChatGPT API
19) AWS Cognito

Write about what happend during signup - Interaction with cognito.

Whenever the user creates a new meeting, the front end will make an API call which will touch the *cvc-api-gateway* and gets routed to the EC2 instance. The application server will make an API call to the *openvidu-deployment-server* to create a session. This session will run in the *openvidu-deployment-server* and the name of the session will be same as the session code generated by the user in the front end. The *openvidu-deployment-server* will get a response from the *openvidu-deployment-server* and that response will be sent as a response to the front end. On successful creation of the session, the application server will record it in the *meeting-details* table, thus identifying this user as the host of the created session.

Creating a session is just not enough. Immediately after successful creation of the session, the front end will make an API call to create a connection for that session. The data flow in this case will be same as before, except this time a connection object is returned this time. Using the Session ID and Connection ID, the front end will make an API call to the *openvidu-deployment-server* and create a WebSocket with it. Now, any other user who wants to join a running session will have to make an API call only to create a Connection ID and then create a WebSocket with *openvidu-deployment-server*. One instance of *openvidu-deployment-server* can run multiple sessions. All these sessions can be managed using the *application-server* by making REST API calls to *openvidu-deployment-server*, which can be controlled from the front end through REST API calls.

Activities performed by the users in the front end like, creating a session, joining a session, leave a session, clicking camera and mice mute and unmute buttons, screen share, clicking recording button, etc. will be notified to the *application-server* through REST API calls. The application server will keep recording these activities in the *meeting-details* DynamoDB table.

The *application-server* runs in a AWS EC2 instance, which is the heart of the whole application. It is a web server built using Flask. The *openvidu-deployment-server*, which runs in a different EC2 instance, is another web server that provides all the necessary infrastructure for streaming real-time audio and video. It can be treated as a black box whose state can be controlled through REST API calls provided in its documentation. The *cvc-api-gateway* is an AWS API Gateway that directs REST API calls from the frond end application to other services in the application, like *application-server, meeting-analytics-processor, meeting-details-retriever, meeting-emailer*.

Now that we have discussed about how meetings are created and managed, lets us see what happens when a meeting ends. Whenever a meeting is ended, in the next steps two different kinds of tasks have to be performed on the collected data - meeting click-streams and video recordings. The *openvidu-deployment-server* stores the recorded meeting in the EC2 instance it is running on.

Whenever a meeting ends, the application server will publish messages into two queues - *session-analytics-queue* & *video-processing-queue*. The message in each of these queues will be the Session ID. The *meeting-analytics-processor*, which is triggered by an Event Bridge Scheduler, will read a message from the *session-analytics-queue* and will perform the data analytics calculations on the click-stream data stored in the *meeting-details* table. This analyzed data will be stored in the *meeting-analyses* table. Similarly, the *video-processor* will periodically poll messages from the *video-processing-queue*. For each message, the *video-processor* will fetch the video recording from the *openvidu-deployment-server*. Here it will extract video from audio, transcribe the audio into text, translate text transcripts into other languages and will make an API call to ChatGPT to extract comprehensive analysis on transcripts, covering sentiment, topic modeling, speaker contributions, keyword extraction, named entity recognition, dialogue flow, diversity of opinions, key quotes identification, contextual analysis, and language complexity. All these newly generated files from videos will be stored in *cvc-meeting-recordings* S3 bucket.

There are 4 data stores in this application - two DynamoDB tables and two S3 buckets. The *meeting-details* table stores the click-stream data and also logs information about all the API calls made by users. Data will be written into *meeting-details* table by the *application-server*. The *meeting-analyses* table stores the meeting analytics data processed by the *meeting-analytics-processor*. The *cvc-meeting-recordings* bucket stores the video recordings, extracted audio, meeting transcripts (in English), translated meeting transcripts to other languages and the ChatGPT analysis of the meeting transcripts, all of which are processed by the *video-processor*.

After successful completion of processing by *meeting-analytics-processor*, it will send an email consisting meeting recording link, transcripts, attendance, all kinds of analyses we generated to the meeting host. AWS Simple Email Service is used to achieve this. Additionally, the meeting host can also send the meeting details to all the participants by simple click of a button in the front end. This button click will make an API call, which will hit the *cvc-api-gateway* and will trigger the *meeting-emailer* lambda function. This lambda function will retrieve all the participants for the meeting ID, fetch their email and will send them emails using AWS Simple Email Service.

Also, whenever the user logs in, they will be able to see details of all the meeting they hosted. To fetch details of all the meetings, upon login, the front end will make an API call that will trigger *meeting-details-retriever* upon touching the API gateway.

IV. Key Design Details

In the ARCHITECTURE & DATA FLOW section, role of each micro-service and API are clearly explained. It is easy to see the *application-server* and *openvidu-deployment-server* are very important components for functioning of the video call application and there is no other way of configuring them. Other than orchestrating the video calls and managing audio-streams, the application processes data of the meetings that ended and sends emails to hosts and participants. One can argue that many functionalities that are implemented using the *video-processor* and other lambda functions can be implemented on the application server itself. Although it sounds fine theoretically, through multiple iterations of architecture design, we decided that the existing configuration is the best.

Our emphasis is that the application will coordinate and manage multiple meetings at a time. Processing video and performing analytics on the meeting that ended require significant amount of database queries and compute power which will impact the performance and latency of the ongoing meetings. So, it is better to separate the post-meeting analysis compute from the application server.

Although its ideal to use a lambda function instead of EC2 for the *video-processor*, the choice of EC2 has been made for various reasons. First and the most significant one being that the python packages that are required to process the video, exceed the allowable size limit on lambda functions. Also, if we optimize package size by selectively building packages that are only necessary for this computation and ignore the others, lambda will not be an idea choice for meeting that are very long. Say a meeting goes for 2 hours and the host records it at highest resolution possible, the size the of file is going to be soo huge that 15 minutes compute time on lambda may not be enough. Even if we can do it under 15 minutes, the memory might not be sufficient.

Also, from our experiments, we found that lambda function *meeting-analytics-processor* is enough for analyzing the click-stream data stored in *meeting-details* and sending emails because it is computationally not as heavy as video processing and if we run EC2 for this, we would be wasting compute. The same reasoning goes for the choice of lambda function for *meeting-details-retriever* and *meeting-emailer*.

Regarding, the use of queues - *session-analytics-queue* and *video-processing-queue*, it was an ideal choice for many reasons. Although we can make an API call to the *video-processor* to tell it to process the video for each ended meeting, practically its not a good design. We have considered that EC2 instances can fail when a video processing is going on and when that happens we wouldn't compute analysis on video data. Using an SQS queue, we can poll messages from the queue and process them when needed. We will delete the message from the queue only when the video-processing is completely done. If the process fails in between, the newly launched EC2 instance will process it again till the end.

Similar logic goes for the use of the *session-analytics-queue*. We have placed a trigger on the *session-analytics-queue* to trigger *meeting-analytics-processor* lambda function. The message will be deleted from the queue only when the lambda is completely done executing. Also, It will launch as many number of lambda instances as there are messages and will perform automatic load-balancing.

Though we have used a single EC2 instance for *video-processor*, in deployment we will be using an elastic load balancer and launch multiple EC2 instances. Due to resource constraints and significant cost of testing the load balancer and multiple EC2 instances, we have used only one instance. The same explanation goes for the use of a single EC2 instance for *application-server* and *openvidu-deployment-server*.

Regarding data stores, we used DynamoDB tables - *meeting-details* and *meeting-analyses*, to store click-stream and meeting analyses because these are semi-structured data. The video and transcripts are stored in S3 because they are unstructured data.

V. Implementation Details & Code Structure

A. Application Server

Certainly, let's delve into a more detailed explanation of the two endpoints:

1) /api/sessions:-
   This pivotal endpoint orchestrates the creation of sessions within the application server. Upon invocation, it evaluates whether a session with the provided ID already exists. If not, a new session is initiated, designating the initiator as the host or moderator. The moderator, being the session orchestrator, is granted the authority to record the ongoing session. The process involves a meticulous verification of the session ID's existence. This verification mechanism determines whether to proceed with initializing a new session or to disregard the call, as the session is already in progress.
   Internally, this endpoint interfaces with the OpenVidu server through a POST request. The request encapsulates essential details such as authentication credentials and the session ID earmarked for creation. Subsequently, a new entry encapsulating the meeting ID, host details, and the initiation time of the call is meticulously stored in the meeting-details DynamoDB table. This record-keeping mechanism ensures a comprehensive log of each session, facilitating easy retrieval and management.
2) /api/sessions/<sessionId>/connections:-
   This endpoint is dedicated to the establishment of connections for participants within a specific session. It not only creates a connection but also furnishes a WebSocket to the frontend user, ensuring seamless communication. An additional layer of functionality involves the verification of a participant's role, particularly determining whether the participant holds the esteemed position of a moderator.
   In cases where the participant is indeed a moderator, their credentials are registered as the host of the meeting in the meeting-details DynamoDB table. This meticulous recording of the moderator's details enriches the system's understanding of the ongoing meeting dynamics. Similar to the previous endpoint, the process involves a POST call to the OpenVidu server. This call is structured to request a WebSocket connection, and concurrently, it communicates the participant's role, thereby bestowing exclusive permissions as befitting their moderator status.
   Collectively, these endpoints serve as the backbone of the application server, orchestrating the creation of sessions, managing connections, and seamlessly integrating with the OpenVidu server to facilitate robust real-time communication.
3) /api/sessions/<sessionId>/participants:- This API plays a crucial role in managing participant entries into a session. When participants join a session, this endpoint captures pertinent information, such as their join time, and stores it in the meeting-details DynamoDB. The utilization of mutex locks during write operations ensures the integrity of the data. This precaution becomes particularly significant in scenarios where a substantial number of participants may concurrently enter a video call, necessitating orderly updates to a single database element.
4) /api/sessions/<sessionId>/participants/<participantId> /events:- At the heart of participant interaction tracking within a session, this API is triggered by various actions, including toggling video/audio, screen sharing, stopping screen sharing, and participant exits. The detailed log of these interactions is stored in the meeting-details DynamoDB. Moderators, endowed with additional responsibilities, can initiate and terminate session recordings. When a moderator initiates recording, the API checks the moderator status of the participant in DynamoDB. If the participant holds moderator privileges, a POST call is made to the OpenVidu server to commence recording, using the corresponding sessionID. In the event of the moderator clicking the stop recording button, the function identifies active recordings associated with the sessionID through a POST call to the OpenVidu server. Subsequently, it orchestrates the cessation of the recording using the recordID, employing another POST call to the OpenVidu server.
   Upon a participant's departure from the session, a POST call to the OpenVidu server gauges the number of remaining participants. If no participants are left, the session concludes, and the details are meticulously updated in DynamoDB. The ensuing step involves dispatching the meetingID to two distinct SQS queues—session-analytics-queue and video-processing-queue. These queues are in turn consumed by two distinct processes: the meeting-analysis Lambda function conducts an analysis on the details present in the meeting-details DynamoDB, while the video-processing-queue oversees the video processing on the recorded meeting.
   These APIs collectively form the backbone of the application's participant management and interaction tracking, ensuring seamless and comprehensive handling of diverse user actions within a session.
5) Background Task:-
   This function operates as a crucial component, functioning as a daemon to ensure the integrity of sessions within the application. Its primary role is to identify and handle dangling sessions—those that have been created but not joined by participants, or sessions prematurely

closed due to issues such as internet connectivity problems. Executing at regular intervals of 120 seconds, this daemon proactively checks for the presence of active sessions without participants by initiating a POST call to the OpenVidu server.
Upon identifying such sessions, the daemon executes a series of well-coordinated actions. Firstly, it stops any ongoing recording associated with the session, ensuring a clean transition. Subsequently, the daemon proceeds to terminate the session. This involves making both POST and DELETE calls to the OpenVidu server, effectively closing the session and freeing up resources. As a final step, the meetingID linked to the concluded session is systematically pushed to SQS queues. This enqueued data is then made available for further processing, ensuring a streamlined and efficient handling of session lifecycle events.

6) /health:-
The health endpoint serves as a fundamental tool for monitoring the operational health of the application server. When a request is made to this endpoint, a rapid assessment is conducted to determine whether the server is actively running or facing any issues. This endpoint provides a straightforward and effective means of real-time health monitoring, allowing administrators and automated systems to quickly gauge the server's status. The simplicity and reliability of the '/health' endpoint make it an integral part of the server's overall health and performance management.

B. Video Processing Server

The video Processing server is an EC2 instance that will run a python code. This Python code consists of a while loop that goes on in infitie loop, with a sleep time of 15s for each loop. In each iteration of this loop, it will pull one message from the AWS SQS. We use AWS Boto3 Python SDK to create an SQS client using the url of the queue. Using this SQS client we will pull a message from the queue. If there exists no message in the SQS it will pass to the end of the iteration. If there is a message in the SQS, where each message corresponds to a meeting ID, it will process the the video corresponding that Session ID first. if we know the session ID we can get the video data from the *openvidu-deployment-server* using REST API calls.

After the video is downloaded to the EC2, it will be further processed to obtain audio from it. To obtain audio from video we use MoviePy library in python. Next we will create an s3 client using Boto3 in python. Using the S3 client we will create a folder with session Id as name in the *cvc-meeting-recordings* bucket. To this folder we will upload the video and audio files. Next we will create an AWS Transcribe client using Boto3. Using this transcribe client we will get a JSON output of transcriptions, which will be processed further using heuristics and we will generate transcriptions in English. This English transcriptions will be stored to a local text file. Next we will translate these english transcriptions into Japanese, Spanish, Hindi and Chinese languages. All of these transcriptions will be saved to local text files.

Now we will upload transcriptions in 5 languages to the folder we created in the S3 bucket for the session ID. Next, we will have to generate comprehensive analysis on transcripts, covering sentiment, topic modeling, speaker contributions, keyword extraction, named entity recognition, dialogue flow, diversity of opinions, key quotes identification, contextual analysis, and language complexity. For this we will have to use ChatGPT API, which will be called using the OpenAI library in python. Currently we are using a trial version of the API. A highly tested prompt and the transcripts will be passed as input to the ChatGPT API and we will get a report containing comprehensive analysis on transcripts, covering sentiment, topic modeling, speaker contributions, keyword extraction, named entity recognition, dialogue flow, diversity of opinions, key quotes identification, contextual analysis, and language complexity. There will he heading for each analysis in the ChatGPT output. This output will be saved to a local text will and will be uploaded to the S3 bucket.

At last we will delet the message from the SQS to prevent it from being processed again. Also, all the activities in this EC2 instance are neatly logged.

C. Databases - DB1, DB2 & S3

We employ two DynamoDB databases to comprehensively store information about all hosted meetings to date and the complete user base that has engaged with our video call application.

1) Meeting Details Database:-
The meeting-details DynamoDB employs the meetingID as the primary key to meticulously store information about meetings. For each meeting record, essential details such as the start time, end time, and the identity of the meeting host are stored. Furthermore, the database captures a comprehensive list of meeting participants, with individual participant entries containing specifics about their interactions during the meeting.
The interaction details are encapsulated as an array, where each element within the array comprises the type of interaction and the corresponding timestamp. This meticulous recording of interactions provides a granular and chronological account of the activities conducted by each participant throughout the meeting duration. In essence, the meeting-details DynamoDB acts as a structured repository, ensuring a detailed and organized record of all pertinent meeting information.
2) Meeting-analyses Database:-
The meeting-analyses DynamoDB is designed to store user analytics, leveraging the userID as the primary key. Within this database, the 'meetings' attribute contains comprehensive information about the meetings attended by the user, including details about their activity during each meeting. Additionally, the database maintains an 'available meetings' attribute, indicating which meetings

are accessible to the user. This attribute serves as a valuable indicator of meetings that are open and accessible to the respective user.

3) CVC Meeting Recordings S3 Bucket:- The S3 bucket named cvc-meeting-recordings serves as a repository for various meeting-related data. Within this bucket, information is stored in a structured manner, featuring transcripts in five languages, ChatGPT analyses, and video recordings. Each meeting is organized into a directory, and the directory is named after the meetingID, ensuring a systematic arrangement of data for easy retrieval and management.

D. Lambda Functions

There are three Lambda functions designed for distinct purposes: one for processing meeting details, another for interacting with the frontend to provide information on attended meetings, and the last for disseminating meeting details and granting access to all users.

1) Meeting Analyzer:-
The Meeting-Analyzer Lambda function is triggered when meeting details are placed in the ‘session-analytics-queue‘ SQS queue. Upon activation, it retrieves the meetingID from the SQS queue and proceeds to process the corresponding meeting details. This involves querying the ‘meeting-details‘ DynamoDB to obtain information about the meeting. For each participant in the meeting, the function meticulously processes their interactions, encompassing actions such as toggling video/audio/screensharing, joining, and leaving the meeting.
Utilizing these interaction logs, the function calculates various metrics, including audio activity, video activity, screen share activity, and the time spent by each participant in the call. The computed details are then stored in the ‘meeting-analysis‘ DynamoDB. For each user, a new entry specific to the meetingID is appended to their list of attended meetings.
Additionally, an email is dispatched to the host of the meeting (information sourced from ‘meeting-details‘ DynamoDB). This email contains a comprehensive analysis of the meeting details for all participants present. The analysis includes individual breakdowns of the audio, video, and screen sharing activities, providing a holistic view of each participant’s engagement during the meeting.

2) Meeting Details Retriever:-
The Meeting-Details-Retriever Lambda function is invoked through the API Gateway, serving the purpose of providing participants access to the details of meetings they have attended. Upon triggering, the function retrieves this information by querying the ‘meeting-analysis‘ DynamoDB, where the user serves as the key. For each user, the function compiles a list of attended meetings, drawing from the corresponding column in the DynamoDB.
There is an additional column in the meeting-analysis DynamoDB that specifies the meetings visible to the user (details of which are explained in the subsequent Lambda function). For each meeting, the function presents a comprehensive overview. This includes the user’s interactions during the meeting, a transcript available in five languages, ChatGPT’s analysis of the meeting, and the associated video recording. Meetings where the user holds the role of moderator feature an exclusive button allowing the moderator to share meeting details with participants.
Until the moderator utilizes this button, no other participant gains access to meeting details in the frontend, nor do they receive any information about the meeting via email. Additionally, it’s worth noting that the processing time required for meeting transcripts may cause a delay in the visibility of transcript details in the frontend. During this processing period, such details remain unavailable to participants.

3) Meeting Email Sender:-
The Meeting-Email Lambda function is activated when the host clicks the ”Share Details of the Meeting” button, intending to distribute meeting information to all participants. This involves retrieving the list of participants from the ‘meeting-details‘ DynamoDB, ensuring that each participant’s list of available meetings in the meeting-analysis DynamoDB is updated. This update ensures that all participants can view the meeting details in the frontend.
Subsequently, the Lambda function proceeds to send emails to all participants and the host of the meeting. Each email includes comprehensive details such as transcripts in five languages, ChatGPT analysis, and the video recording of the meeting. For each participant (excluding the host), the email additionally contains details of their individual interactions during the meeting. In contrast, the host receives a comprehensive overview, encompassing interactions of all participants, along with the complete transcript and video data of the meeting.
This Lambda function serves as a key facilitator in disseminating meeting information, fostering transparency, and ensuring that participants and hosts alike have access to a detailed record of their collaborative sessions.

VI. Results

See Figure 2, 3, 4 for results of outputs.

ANALYSIS OF THE CALL

SENTIMENT:

The sentiment in the call transcript appears to be primarily neutral. There are no strong positive or negative emotions expressed by the speakers. The conversation focuses on discussing various topics without significant emotional bias.

TOPIC MODELING:

Based on a topic modeling analysis of the call transcript, several key themes emerge:

1. Activism on college campuses: The speakers discuss the shift of activism from corporate boardrooms to college campuses. There is a concern about the current state of college campuses and the challenges faced by Jewish and Muslim students.

2. The role of universities in taking a stand on social issues: The conversation explores the question of whether universities should take a collective stand on social issues or allow individual voices to prevail. The University of Chicago's approach of promoting free speech is highlighted as a positive example.

3. Bill Ackman's investment strategies: The discussion briefly touches on Bill Ackman's investment strategies, particularly his recent focus on investing in treasury and making macro bets on interest rates.

4. Potential changes in response to current pressures: The speakers anticipate that there may be changes in response to the current pressures faced by universities. They suggest that university boards may become more sensitive to these issues and there may be increased security for certain students.

SPEAKER CONTRIBUTION:

Speaker 1 has a significantly higher word count compared to Speaker 0. Speaker 1 provides more detailed responses and elaborates on various topics discussed in the call transcript. Speaker 0 mainly asks questions and prompts Speaker 1 to share their insights.

KEYWORD EXTRACTION:

Frequently mentioned keywords or phrases in the call transcript include:

1. Activism
2. College campuses
3. Business people 4. Jewish and Muslim students 5. Social issues 6. University of Chicago 7. Free speech 8. Bill Ackman 9. Investing 10. Treasury 11. Interest rates 12. University boards

NAMED ENTITY RECOGNITION:

Named entities identified in the call transcript are:

1. Bill Ackman - Business person 2. Mark Owen - Individual associated with Apollo 3. Claudine Gay - Individual associated with Harvard 4. Mark Rowan - Individual associated with Penn 5. Congress - Legislative body 6. University of Chicago - Educational institution

DIALOGUE FLOW ANALYSIS:

The conversation follows a typical turn-taking pattern, with Speaker 0 initiating most exchanges by asking questions, and Speaker 1 providing detailed responses. There are no notable shifts in topic or interruptions.

DIVERSITY OF OPINIONS:

Both speakers express agreement on certain points, such as the need for universities to address the concerns on college campuses and the importance of free speech. However, they also recognize the complexity of the issues discussed and do not provide definitive answers or solutions, indicating a diversity of opinions.

KEY QUOTES IDENTIFICATION:

1. ”There's no doubt that there's a lot of concern in the business but other communities about what's going on on college campuses.” 2. ”Should corporations be taking positions on these kind of issues?” 3. ”But there's been a lot of free speech at the University of Chicago. And I think that's a great tradition there.” 4. ”Some campuses are far left, some are maybe more conservative and people who disagree with the conventional or the majority view don't get the kind of support that they might want to receive from the college presences or universities in some cases.” 5. ”I think the university boards are probably going to be more sensitive to these issues.”

CONTEXTUAL ANALYSIS:

The conversation provides insights into the broader context surrounding activism on college campuses, the role of universities in addressing social issues, and the investment strategies of Bill Ackman. The speakers draw from their personal experiences and observations to discuss these topics.

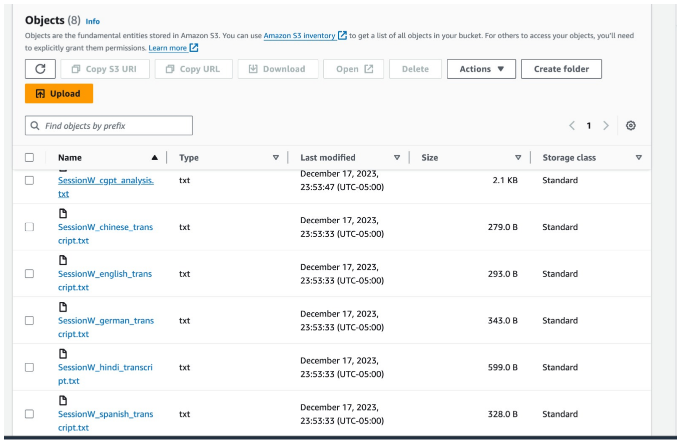
LANGUAGE COMPLEXITY:

The language used in the call transcript is relatively formal and professional. There are no instances of technical or highly complex language. The speakers convey their ideas clearly and effectively without excessive use of jargon.

Overall, the call transcript analysis highlights the sentiment, key themes, speaker contributions, frequently mentioned keywords, named entities, dialogue flow, diversity of opinions, key quotes, contextual analysis, and language complexity present in the conversation.

VII. Opportunities for Improvement

Enhancing the user interface for video calls will be a priority. Additionally, we aim to leverage Amazon EKS for

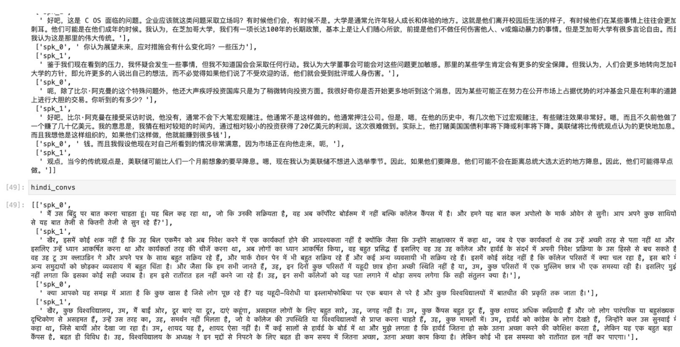


Objects (8)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
SessionW_cgpt_analysis.txt	txt	December 17, 2023, 23:53:47 (UTC-05:00)	2.1 KB	Standard
SessionW_chinese_transcript.txt	txt	December 17, 2023, 23:53:33 (UTC-05:00)	279.0 B	Standard
SessionW_english_transcript.txt	txt	December 17, 2023, 23:53:33 (UTC-05:00)	293.0 B	Standard
SessionW_german_transcript.txt	txt	December 17, 2023, 23:53:33 (UTC-05:00)	343.0 B	Standard
SessionW_hindi_transcript.txt	txt	December 17, 2023, 23:53:33 (UTC-05:00)	599.0 B	Standard
SessionW_spanish_transcript.txt	txt	December 17, 2023, 23:53:33 (UTC-05:00)	328.0 B	Standard

Fig. 2. S3 Image of outpute

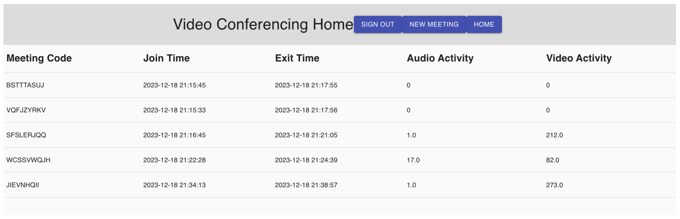


```
  ' 好吧，这是 C OS 面临的问题。企业应该就这类问题采取立场吗？有时候他们会，有时候不是。大学是通常允许年轻人成长和体验的地方。这就是他们离开校园后生活的样子，有时候他们在某些事情上往往会更加
刺耳。他们可能是在他们成年的时候。我认为，在芝加哥大学，我们有一项长达100年的长期政策，基本上是让人们随心所欲，前提是他们不做任何伤害他人、v或煽动暴力的事情。但是芝加哥大学有很多言论自由。而
我认为这是那里的伟大传统。'],
 ['spk_0', ' 你认为展望未来，应对措施会有什么变化吗？一些压力'],
 ['spk_1',
  ' 鉴于我们现在看到的压力，我怀疑会发生一些事情，但我不知道国会会采取任何行动。我认为大学董事会可能会对这些问题更加敏感。那里的某些学生肯定会有更多的安全保障。但我认为，人们会更多地转向芝加
大学的方针，即允许更多的人说出自己的想法，而不必觉得如果他们说了不受欢迎的话，他们就会受到批评或人身伤害。'],
 ['spk_0',
  ' 呃，除了比尔·阿克曼的这个特殊问题外，他还大声疾呼投资国库只是为了稍微转向投资方面。我很好奇你是否开始更多地听到这个消息，因为某些可能正在努力在公开市场上占据优势的对冲基金只是在利率的道路
上进行大胆的交易。你听到的有多少？'],
 ['spk_1',
  ' 好吧，比尔·阿克曼在接受采访时说，他没有，通常不会下大笔宏观赌注。他通常不是这样做的。他通常押注公司。但是，嗯，在他的历史中，有几次他下过宏观赌注，有些赌注效果非常好。嗯，而且不久前他做了
一个赚了几十亿美元。我的意思是，我猜在相对较短的时间内，通过相对较小的投资获得了20亿美元的利润。这次很难做到。实际上，他打赌美国国债利率将下降或利率将下降。美联储将比传统观点认为的更快地加息。
而且我想他是这样组织的，如果他们这样做，他就能赚到很多钱'],
 ['spk_0', ' 钱。而且我假设他现在对自己所看到的情况非常满意，因为市场正在向他走来，呃，'],
 ['spk_1',
  ' 观点，当今的传统观点是，美联储可能比人们一个月前想象的要早降息。嗯，现在我认为美联储不想进入选举季节。因此，如果他们要降息，他们可能不会在距离总统大选太近的地方降息。因此，他们可能得早点
做。']]

[49]: hindi_convs

[49]: [['spk_0',
  ' मैं उस बिंदु पर बात करना चाहता हूं। यह बिल कह रहा था, जो कि उनकी सक्रियता है, वह अब कॉर्पोरेट बोर्डरूम में नहीं बल्कि कॉलेज कैंपस में है। और हमने यह बात कल अपोलो के मार्क ओवेन से सुनी। आप अपने कुछ साथियों
से यह बात तेजी से कितनी तेजी से सुन रहे हैं?'],
 ['spk_1',
  ' खैर, इसमें कोई शक नहीं है कि उह बिल एकमैन को अब निवेश करने में एक कार्यकर्ता होने की आवश्यकता नहीं है क्योंकि जैसा कि उन्होंने साक्षात्कार में कहा था, जब वे एक कार्यकर्ता थे तब उन्हें अच्छी तरह से पता नहीं था और
इसलिए उन्हें ध्यान आकर्षित करना था और कार्यकर्ता तरह की चीजें करना था, अब लोगों का ध्यान आकर्षित किया, वह बहुत प्रसिद्ध हैं इसलिए वह उह उह कॉलेज और हार्वर्ड के संदर्भ में अपनी निवेश प्रक्रिया के उस हिस्से से बच सकते है
वह उह टू उम क्लाउडिन गे और अपने पत्र के साथ बहुत सक्रिय रहे हैं, और मार्क रोवन पेन में भी बहुत सक्रिय रहे हैं और कई अन्य व्यवसायी भी सक्रिय रहे हैं। इसमें कोई संदेह नहीं है कि कॉलेज परिसरों में क्या चल रहा है, इस बारे में
अन्य समुदायों को छोड़कर व्यवसाय में बहुत चिंता है। और जैसा कि हम सभी जानते हैं, उह, इन दिनों कुछ परिसरों में यहूदी छात्र होना अच्छी स्थिति नहीं है या, उम, कुछ परिसरों में एक मुस्लिम छात्र भी एक समस्या रही है। इसलिए मुझे
नहीं लगता कि इसका कोई सही जवाब है। हम इसे रातोंरात हल नहीं करने जा रहे हैं। उह, इन सभी कॉलेजों को यह पता लगाने में थोड़ा समय लगेगा कि सही संतुलन क्या है।'],
 ['spk_0',
  ' क्या आपको यह समझ में आता है कि कुछ खास है जिसे लोग पूछ रहे हैं? यह यहूदी-विरोधी या इस्लामोफोबिया पर एक बयान से परे है और कुछ विश्वविद्यालयों में बातचीत की प्रकृति तक जाता है।'],
 ['spk_1',
  ' खैर, कुछ विश्वविद्यालय, उम, मैं बाईं ओर, दूर बाएं या दूर, दाएं कहूंगा, असहमत लोगों के लिए बहुत सारे, उह, जगह नहीं है। उम, कुछ कैंपस बहुत दूर हैं, कुछ शायद अधिक रूढ़िवादी हैं और जो लोग पारंपरिक या बहुसंख्यक
दृष्टिकोण से असहमत हैं, उन्हें उस तरह का, उह, समर्थन नहीं मिलता है, जो वे कॉलेज की उपस्थिति या विश्वविद्यालयों से प्राप्त करना चाहते हैं, उह, कुछ मामलों में। उम, हार्वर्ड को कांग्रेस के लोग देखते हैं, जिन्होंने कल उस सुनवाई में
कहा था, जिसे बायीं ओर देखा जा रहा है। उम, शायद यह है, शायद ऐसा नहीं है। मैं कई सालों से हार्वर्ड के बोर्ड में था और मुझे लगता है कि हार्वर्ड जितना हो सके उतना अच्छा करने की कोशिश करता है, लेकिन यह एक बहुत बड़ा
कैंपस है, बहुत ही विविध है। उह, विश्वविद्यालय के अध्यक्ष ने इन मुद्दों से निपटने के लिए बहुत ही कम समय में जितना अच्छा, उतना अच्छा काम किया है। लेकिन कोई भी इस समस्या को रातोंरात हल नहीं कर पाएगा।'],
```

Fig. 3. Multiple Language Translated



Meeting Code	Join Time	Exit Time	Audio Activity	Video Activity
BSTTTASUJ	2023-12-18 21:15:45	2023-12-18 21:17:55	0	0
VQFJZYRKV	2023-12-18 21:15:33	2023-12-18 21:17:56	0	0
SFSLERJQQ	2023-12-18 21:16:45	2023-12-18 21:21:05	1.0	212.0
WCSSVWQJH	2023-12-18 21:22:28	2023-12-18 21:24:39	17.0	82.0
JIEVNHQII	2023-12-18 21:34:13	2023-12-18 21:38:57	1.0	273.0

Fig. 4. Meetings History

scaling both the Flask Application (Application Server) and the OpenVidu Server (WebRTC communication facilitator). As part of our improvement efforts, we are introducing a Language Model-based (LLM) interface for the generated transcripts. This interface enables us to pose questions based on the meeting transcript, further enriching the interactive and analytical capabilities of our application.

VIII. Conclusion

In conclusion, this project introduces a robust video conferencing application developed on the AWS Cloud, seamlessly integrating OpenVidu technology to enhance real-time collaboration experiences. The application, accommodating up to four participants, encompasses essential features such as login and registration, session recordings, screen sharing, and chat functionality. Post-call meeting analysis, a distinctive aspect of this application, includes user engagement tracking, attendance monitoring, and the host's ability to share meeting data.

Noteworthy is the application's commitment to inclusivity by providing meeting transcripts in English and translations into Hindi, Chinese, Spanish, and German. Leveraging the ChatGPT API, the system performs a comprehensive analysis on meeting transcripts, covering sentiment, topic modeling, speaker contributions, keyword extraction, named entity recognition, dialogue flow, diversity of opinions, key quotes identification, contextual analysis, and language complexity.

This project underscores the synergy achieved through the integration of AWS Cloud, OpenVidu, and advanced analytics, resulting in an efficient and feature-rich video conferencing platform. Beyond enhancing the collaborative experience, the application extracts valuable insights from meetings, contributing to the ongoing evolution of virtual collaboration technologies.

Index Terms—Video Conferencing, OpenVidu, AWS, Real-time Collaboration, Cloud Computing Applications, User Engagement Tracking, ChatGPT Analysis.

References

Open Vidu Docs - https://docs.openvidu.io/en/stable/