A Framework for Automated Digital Media Asset Acquisition using Cloud

Adrian VINTILĂ, Constanța-Nicoleta BODEA
Bucharest University of Economic Studies, Romania
adi.vint@gmail.com, bodea@ase.ro

This paper presents a cloud-based automated media asset acquisition framework designed to enhance the efficiency and speed of media ingest workflows in broadcast environments. Traditional media ingestion, particularly in news production, often involves manual processes such as physical file transfers, operator intervention, and transcoding delays, which can slow down content availability. Our proposed framework leverages cloud storage, mobile journalism technology, and automation scripts to eliminate these inefficiencies. By developing a dedicated mobile application, integrating cloud storage with automated monitoring and downloading mechanisms, and employing a local watchfolder-based transcoding system, the workflow minimizes human intervention and significantly reduces media ingest time. We conducted a series of comparative real-world experiments evaluating the new framework against conventional workflows in a news television station, measuring ingest time and resource utilization. The results demonstrate that our automated solution outperforms traditional and alternative cloud-based methods, reducing ingest time by up to fifteen times while eliminating the need for additional personnel. These findings highlight the potential of automation and cloud computing to optimize media workflows, ultimately improving production speed and operational efficiency which can also lead to potential economic benefits.

Keywords: *Cloud, Automation, Media asset acquisition, Ingest, Broadcast*
DOI: 10.24818/issn14531305/29.1.2025.03

1 Introduction

This paper proposes an efficient framework for automated media asset acquisition using cloud technology and provides an experimental analysis demonstrating its advantages over existing models. Media asset acquisition, commonly referred to as ingest in the broadcast industry, is the process of bringing new media content into a broadcast facility's media ecosystem. It serves as the fundamental entry point for all content in broadcast operations. Whether managing a television station, running a streaming platform, operating a production company, or creating video content, media ingest represents the critical stage where content first enters the media workflow system. Over time, this process has evolved significantly, transitioning from physical tape delivery to modern digital workflows. This paper focuses on broadcast environments, where ingest involves capturing, transcoding, and storing media content within a facility's Media Asset Management (MAM) system. This includes handling multiple formats and sources, such as satellite feeds, live camera or mobile phone feeds, file-based content delivered through digital means, live streaming, and IP streams from news agencies, among others. Several operations typically occur during the ingest process. Incoming content must be transcoded into the facility's preferred video format. Once transcoded, the media is stored in the MAM's primary storage for further processing or live broadcasting. In most cases, one or more ingest operators manage this process, handling tasks such as recording, transcoding, and storage. Given the variety of content sources available to broadcasters, different workflows are tailored to the technical capabilities of each facility. One universally present ingest workflow involves filming with a video camera or a mobile phone, transferring the video footage via a memory card or USB connection, and then copying it to a temporary location where it is transcoded before being imported to the MAM. Since this paper aims to present and test an automated framework focused on speed and efficiency, the primary focus will be on mobile phone-based filming. Mobile

phone camera technology has advanced dramatically in recent years, featuring enhancements such as image stabilization, high dynamic range (HDR), multiple lenses, and AI-driven video processing algorithms [1]. These algorithms improve various aspects of video quality, such as noise reduction, color enhancement, and image focus optimization using object recognition. Additionally, mobile phone cameras are significantly more affordable than traditional professional cameras, offering an excellent price-performance ratio and greater portability. These advancements have led to the rapid adoption of mobile journalism (MoJo) in the media industry [2]. MoJo refers to a growing trend where journalists, particularly reporters, use smartphones equipped with lightweight accessories, such as portable microphones, compact tripods, and small lighting gear, to capture high-quality video footage. Unlike traditional workflows, MoJo eliminates the need for a dedicated camera operator or bulky production equipment, enabling a more agile and efficient filming process. This paper will examine the current ingest workflows of television stations utilizing the MoJo technique and propose an automated media ingest framework that leverages cloud technology. In this framework, media footage filmed on a mobile phone is uploaded to the cloud, automatically downloaded to a local temporary storage location, transcoded, and imported to the MAM, entirely without human intervention. Once the framework is established, we will design an ingest workflow for reporters, enabling them to film events and seamlessly send video footage to the MAM using a mobile application. To validate our approach, we will conduct real-world experiments comparing existing workflows of a modern news television station with our proposed framework and analyze the results. The goal is to enhance an already efficient filming model by further accelerating the ingest process. Speed is a critical factor in broadcasting, particularly in news environments where rapid turnaround times are essential. Increased automation, improved efficiency, and reduced resource requirements could also yield economic benefits for broadcasters.

This paper is structured as follows: Following the introduction, we present a review of existing literature on the topic. The third section outlines the Methodology, detailing the steps necessary to build the framework and to conduct the experiments. Next, we describe the technical architecture of the framework in depth, and we'll design the workflow. In the fifth section, we describe the experiments. Next, we'll present and discuss the results of the experiments conducted. Finally, the paper concludes with the Conclusions and References sections.

2 Literature review

Media Asset Management (MAM) is an important technology that is rapidly changing the media, entertainment, and content industries. It enables organizations to manage and distribute digital assets more effectively. As companies expand into digital platforms, MAM is becoming an essential tool for survival and growth in a competitive digital landscape [3]. It is used in various industries: film, internet & IT, broadcasting or corporate and government sectors. Regardless of the industry in which it is implemented, every MAM system must be capable of acquiring media assets. Also called media ingest, it's the first step of every MAM system. The ingest process ensures that media assets are acquired, validated, and structured efficiently enabling downstream workflows such as content management and distribution [4]. Our goal in this paper is to introduce a more efficient ingest approach by automating manual tasks in traditional workflows and leveraging cloud technology. Automation continues to evolve in all fields to optimize efficiency, speed, flexibility, and sustainability of production systems leading to increased productivity and cost reduction [5],[6],[7],[8],[9]. In this paper we are focusing on cloud-based workflow automation, which has use cases in the media and broadcasting industries such as automated video encoding, metadata tagging, and content distribution across streaming platforms [10]. Our proposed framework uses cloud and on-premises resources. Both have their merits,

and the best choice depends on organizational needs, cost considerations, and IT capabilities [11]. Public clouds (e.g., AWS, Azure, Google Cloud) provide cost efficiency and scalability but raise security and data ownership concerns [12]. We opted for a public cloud solution as its cost-effectiveness and scalability advantages outweighed potential security considerations in our analysis. Cloud computing will facilitate mobile, remote, and collaborative journalism, minimizing reliance on physical newsrooms. As journalism rapidly evolves with advancements in artificial intelligence (AI), cloud computing, and automation, media organizations must make strategic investments in these technologies to remain competitive [13]. Our framework uses a hybrid cloud computing approach. Hybrid cloud computing is basically a combination of cloud computing with on-premises resources to provide work portability, load distribution, and security [14]. Hybrid cloud computing is transforming the broadcasting industry by enabling media companies to balance the need for high-performance on-premises infrastructure with the scalability and flexibility of cloud services. Traditionally, broadcasters relied on fully on-premises setups for production, asset management, and distribution. However, with the increasing demand for remote collaboration, cloud-based workflows and hybrid cloud architectures have become essential. One emerging trend in the field of journalism is Mobile Journalism (MoJo), a new form of multimedia newsgathering and storytelling that enables journalists to document, edit and share news using small, network connected devices like smartphones [15]. MoJo benefits greatly from cloud integration, as it provides a scalable, flexible, and efficient solution for mobile journalists working in remote or dynamic environments [16]. This is why we selected this method of news gathering when designing our experiments as digital immediacy has transformed journalistic workflows, emphasizing the pressure to update stories continuously. Media outlets such as television and digital media compete for breaking news dominance, leading to rapid, incremental updates that often prioritize speed over depth [17]. News strategies in digital journalism are driven by competition, audience retention, and economic survival. Therefore, in the upcoming chapters, we introduce and evaluate a framework aimed at speeding up the essential process of media asset acquisition while optimizing resource utilization.

3 Methodology

This section describes our research methodology. First, we develop the framework's foundational processes, documenting the workflow, then conduct experimental testing, and finally analyze and discuss the obtained results. Figure 1 shows the research methodology.

The development of the automated ingest framework involves work across cloud and on-premises environments, including the creation and deployment of a mobile application. Once the framework is established, we will design an automated workflow tailored to suit this framework. Our real-world experimental testing will compare existing workflows of a news television station with the proposed automated cloud-based approach. Following the experiments, we will generate a performance report and analyze the findings.

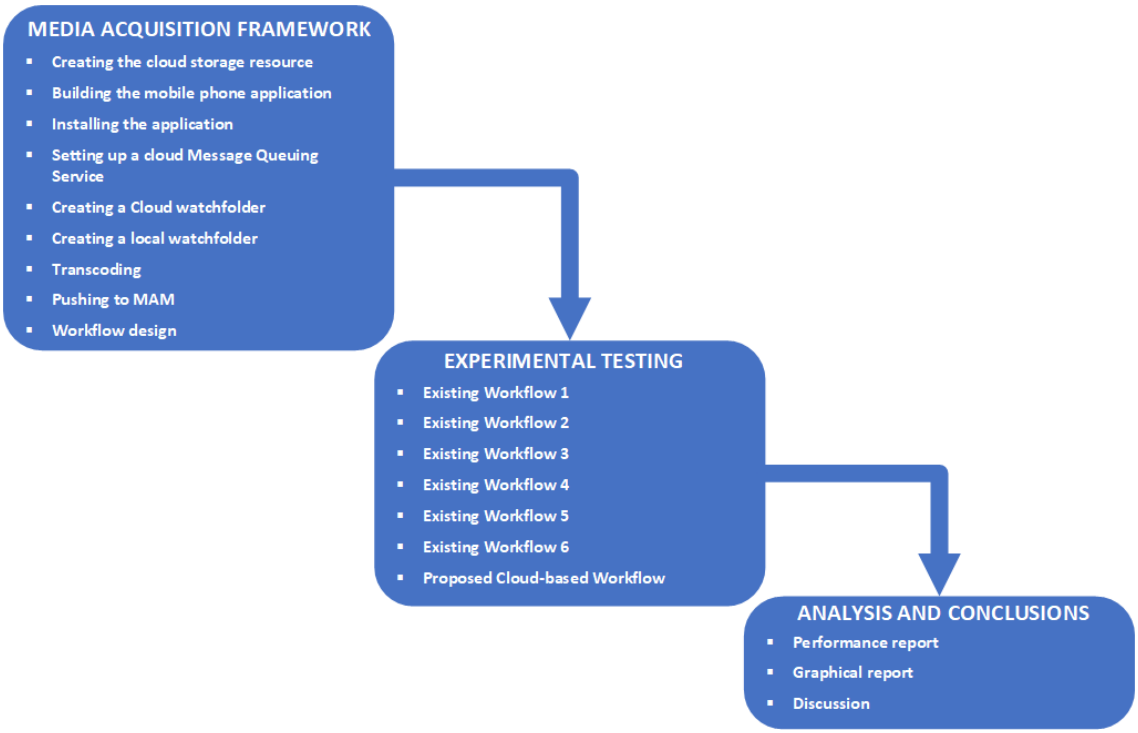

Fig. 1. Research methodology

Subsequent chapters will provide detailed description of each development step.

4 Cloud-based automated digital media acquisition framework

This section provides a comprehensive overview of the cloud-based automated media acquisition framework architecture, detailing each development stage. Our implementation process began with establishing a cloud environment and configuring cloud storage resources. We then developed a mobile application and installed it on the MoJo mobile phone. Using a cloud Message Queuing Service, we created a monitoring script to track files uploaded to the cloud and download them to temporary local storage. Subsequently, we designed a script to continuously monitor the local folder, automatically transcoding and transferring new files to the Media Asset Management system (MAM) as they arrive.

4.1 Setting up cloud infrastructure and the cloud storage resource

We chose Amazon Web Services (AWS) as the cloud computing platform because it is widely adopted in the media industry, provides an extensive suite of services, and benefits from a robust ecosystem, strong support, and thorough documentation. Within the AWS infrastructure, we set up a dedicated S3 bucket, as the cloud storage resource. We then proceeded to configure its access permissions, creating a custom bucket policy that provides access to the objects stored in the bucket to only one user. The user, called "android-s3-uploader", was created using the Identity and Access Management (IAM), a core service of AWS. After the user was created, we defined the permissions for this user required to access the S3 bucket so that we can have the credentials needed for the files upload. The S3 bucket is the cloud location where the reporters are uploading the filmed media assets (videos).

4.2 Developing the mobile phone application

We developed an Android app, called "MediaUpload", using Android Studio. We used the Android operating system as it is a more familiar development environment for us, an iPhone app would work the same way. It is a simple application with just an upload button, as we did not pay too much attention to the Graphical User Interface (GUI) design as it is not relevant for the scope of this paper. The

simple it is, the better. The app allows the reporter to just push a button, select a video from the mobile phone internal gallery and the selected video file is uploaded to the cloud automatically, more specifically, to our S3 bucket. In the Android Studio project, we included the AWS SDK for Android's S3 (Simple Storage Service) library that allows our application to interact with AWS S3 buckets and the AWS Mobile Client library which provides authentication and authorization mechanisms for AWS services.

```
implementation (libs.amazonaws.aws.android.sdk.s3)
implementation (libs.amazonaws.aws.android.sdk.mobile.client)
```

In the MainActivity section of the Android Studio project we defined the credentials, using the access key and secret key generated by the IAM service, and we implemented the upload function and button.

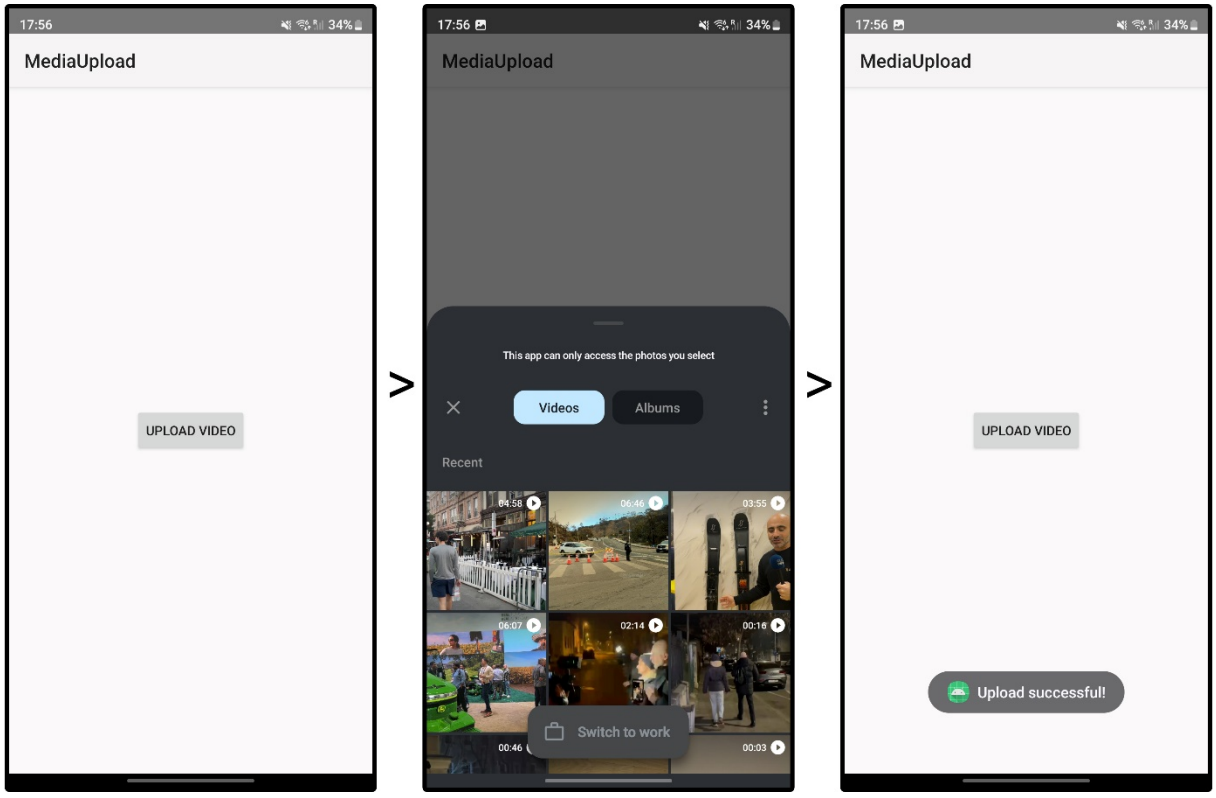

Fig. 2. Android Mobile Phone Application

Figure 2 shows the Graphical User Interface of our application. After pressing the “Upload video" button, the user will be prompted to choose a video from the mobile phone’s video gallery. The selected video is uploaded into the S3 bucket. In “AndroidManifest.xml”, we granted the MediaUpload app permissions to access the internet and to access video files from the local storage. After the file is successfully uploaded, the application shows a confirmation pop-up message.

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_MEDIA_VIDEO"/>
```

This is the first stage of the automated media ingest process, when a reporter captures video using a mobile phone that is part of the MoJo kit. Afterwards, the reporter launches the MediaUpload application to transfer the recorded footage to the cloud. The sole requirement for this upload is that the mobile phone has an active internet connection, which can be established either through cellular data (GSM) or a wireless network (Wi-Fi).

4.3 App installation for the MoJo workflow

As this mobile application is not intended for commercial use and not meant to be public, we have opted not to register it with the Google Play Store, the official digital distribution platform for Android devices. Thus, using Android Studio, we manually compiled the "MediaUpload" project into an Android Package Kit (APK). We then transferred this APK file to the mobile device via USB connection and installed it through the device's pre-installed File Manager application, bypassing the traditional app store distribution method. After the successful installation, we ran a few upload tests to verify that everything works correctly, and to confirm that the selected files are being uploaded to the AWS S3 bucket.

4.4 Automated cloud monitoring and file download script

Once the MediaUpload mobile phone application has been developed and installed, and the media was being uploaded to the cloud, we proceeded to work on a system that can locally download, on premise, every file uploaded to the cloud, immediately after the upload was completed. This meant that we had to configure the S3 bucket to send a notification each time a file is received and a cloud queuing message service, that would store messages sent by the S3 service when a file is uploaded to the cloud. In our case, we used Amazon Simple Queue Service (SQS) to receive an event notification when a new object is created in the bucket, like a file upload would do. Next, we created a Python script that automates the process of downloading files from the Amazon S3 bucket whenever new files are uploaded. It achieves this by polling the SQS queue to retrieve the event notifications created as the files are being uploaded.

The configuration of SQS was straightforward:

- We created a queue called, very generic, "MyQueue";
- Set the "Delivery delay" to "0", so that messages would be added to the queue immediately;
- Using the "Access policy" we added a policy to ensure that the S3 bucket has permission to send messages to our SQS queue.

We also had to configure the S3 bucket to send event notifications to our SQS queue whenever new files are uploaded:

- In the "Properties" section of our S3 bucket we created an event notification;
- We configured the type of event as "All object create events";
- In the "Destination" section, we pointed to our SQS queue by specifying its Amazon Resource Name (ARN).

At this stage the cloud configuration is complete. We can accept video files uploaded from the mobile phone, using our application, and after the file is uploaded into the S3 bucket, an event notification is created.

Next, we designed a Python script, that would run on-premises, on a dedicated server. This script listens to event notifications from the SQS queue and downloads locally the files as soon as they are uploaded to the cloud. When a file is uploaded to an S3 bucket, the S3 service is configured to generate an event notification. This notification contains metadata about the upload, such as the name of the bucket and the object key (file name) of the uploaded file. S3 sends this notification to an Amazon SQS queue, where it is stored. The script begins by importing several libraries needed for its functionality. We used the "boto3" library, the official Python SDK for AWS, to interact with both the SQS and S3 services. "boto3" abstracts the complexities of AWS API calls, allowing us to work with AWS services using Python code. In addition to "boto3", the script imports other libraries for handling JSON data (json), file operations (os), time-based delays (time), and URL

decoding (unquote_plus). The script defines two AWS clients, one for interacting with SQS and the other for S3. These clients are created using the "boto3.client" function, specifying the AWS region where the resources, queue and bucket, are located. The script also includes the URL of the SQS queue, "queue_url", which will be polled for incoming messages. Next, the "poll_sqs_queue" function is defined to handle the polling process. It is an infinite loop that continuously checks the SQS queue for new messages. The function begins by calling the "receive_message" method of the SQS client. This method retrieves from the queue up to 10 messages at a time, waiting for up to 20 seconds, the maximum allowable value for new messages to arrive if the queue is empty. Having this 20 seconds delay, unnecessary requests to the queue when no messages are available are minimized, thus also reducing costs. If no messages are received during the 20 seconds window, the script waits for 5 seconds before polling again. If messages are received, the script iterates through them. Each message body, formatted as JSON, contains the S3 event notification. The notification includes the Records field, which holds details about the uploaded file, such as the name of the bucket and the object key.

```
body = json.loads(message['Body'])
for record in body.get('Records', []):
	s3_bucket = record['s3']['bucket']['name']
	s3_key = unquote_plus(record['s3']['object']['key'])
```

For each record in the notification, the script extracts the bucket name and object key. The object key is URL-encoded, so it is decoded using "unquote_plus" to ensure the file name is in the correct format. These details are then passed to the" download_s3_object" function.

```
def download_s3_object(bucket_name, object_key):
	os.makedirs(LOCAL_FOLDER, exist_ok=True)
	local_file_path = os.path.join(LOCAL_FOLDER, os.path.basename(object_key))
	s3.download_file(bucket_name, object_key, local_file_path)
	print(f"Downloaded '{object_key}' from bucket '{bucket_name}' to '{lo-
	cal_file_path}'.")
```

The "download_s3_object" function ensures that the local folder (specified in the variable "LOCAL_FOLDER") exists by creating it if necessary using "os.makedirs". It constructs the full local path for the downloaded file by combining the folder path with the base name of the object key. The function then uses the "download_file" method of the S3 client to get the file from the bucket and save it to the specified local path. Messages are shown to the console to indicate the progress and completion of the download. After a message is successfully processed and the corresponding file is downloaded, the script deletes the message from the queue using the "delete_message" method. This step ensures that the same message is not reprocessed in subsequent polling iterations.

```
sqs.delete_message(
	QueueUrl=queue_url,
	ReceiptHandle=message['ReceiptHandle']    )
```

The script is executed by calling the "poll_sqs_queue" function within a conditional block that verifies if the script is being run directly. This ensures that the polling process starts only when the script is executed as a standalone program.

```
if __name__ == '__main__':
	poll_sqs_queue()
```

The script is designed to run indefinitely, continuously monitoring the SQS queue for new notifications about file uploads to S3. By doing so, it automates the process of downloading files from S3 to a local temporary folder,

ensuring that uploaded files are promptly available locally for further processing.

4.5 Automated local watchfolder and transcoding script

After establishing the automated file download process from the cloud to a temporary local directory, we implemented an automated file conversion script, written in Python, on the same on-premises server as before, to transform the incoming files into the specific video format required by the Media Asset Management system (MAM) of the news television station involved in our experimental research. The script will monitor the temporary local folder where the files are being downloaded from the cloud. After the conversion, this script will save the video files to a network shared location from where the MAM can import them.

First, we import the necessary Python libraries. The "time" library is used to introduce delays in the folder monitoring process, ensuring the script does not overburden system resources by checking too frequently. The "os" library provides functionality to interact with the file system, enabling the script to list files, verify their properties, and build paths. Finally, the "subprocess" library facilitates the execution of external commands, specifically "Ffmpeg", for the transcoding process. The folder monitoring logic is implemented in the "monitor_folder" function, which continuously scans the specified source folder for new files. It maintains a record of processed files using a set, "processed_files", to prevent multiple processing of the same file. Within an infinite loop, the script retrieves the current list of files in the source folder using the "os.listdir()" function. It filters this list to include only files with an .mp4 extension, regardless of the case, by applying the "str.endswith()" method, as this is the file format used for the mobile phone videos in our experiments. Additionally, it verifies that the listed items are files, not folders, using "os.path.isfile()" function. The filtered list of files is then converted into a set, "current_files".

```
def monitor_folder(source_folder, mxf_folder):
    processed_files = set()
    while True:
        try:
            current_files = {
                f for f in os.listdir(source_folder)
                if f.lower().endswith('.mp4') and
os.path.isfile(os.path.join(source_folder, f))   }
```

By computing the difference between the sets "current_files" and "processed_files", the script identifies new media files that have not yet been processed. For each new file, the script builds its full path using "os.path.join()" and passes this path, along with the destination folder path, to the "process_file" function for further handling. After successfully processing a file, its name is added to the "processed_files" set to avoid reprocessing in subsequent iterations. Between scans, the script introduces a five-second delay using "time.sleep()", to balance efficiency with system resource usage.

```
new_files = current_files - processed_files
            for filename in new_files:
                    filepath = os.path.join(source_folder, filename)
                    process_file(filepath, mxf_folder)
                    processed_files.add(filename)
            time.sleep(5)
```

Next, the "process_file" function is responsible for handling each newly detected file. Upon receiving a file, it first ensures that the file is fully copied to the source folder before beginning the transcoding process. This is necessary because files arriving from an external source, as a cloud service in our case, may not be immediately complete. To verify

this, the script monitors the file size using "os.path.getsize()" in a loop. It compares the current size to the previous size recorded during the previous iteration of the loop. If the file size remains unchanged over consecutive checks, the script concludes that the file is fully written and ready for processing.

```
previous_size = -1
while True:
    try:
        current_size = os.path.getsize(filepath)
        if current_size == previous_size:
            break
        previous_size = current_size
        time.sleep(1)
```

Once the file is deemed ready, we can transcode it. For the transcoding process we used "Ffmpeg", a powerful, open-source multimedia software project used for handling multimedia data such as audio, video, and other related files and streams. It is widely regarded as one of the most versatile and comprehensive tools for media processing [18]. We installed "Ffmpeg" on the local server that is running the script and used it via the "subprocess" library to invoke the "Ffmpeg" command-line tool directly from Python. To perform the transcoding, the script prepares an "Ffmpeg" command as a list of arguments. The input file is specified with the -i option, while the video is encoded using the H.264 codec (libx264) with a bit rate of 50 Mbps, a frame rate of 50 frames per second, and a color space of Rec. 709. The output video format uses 4:2:2 chroma subsampling in a planar layout (yuv422p). The audio is encoded as 24-bit PCM (pcm_s24le) with a sample rate of 48 kHz. The final argument specifies the path for the transcoded file. The FFmpeg command is executed using the "subprocess.run()" function.

```
ffmpeg_cmd = [
        'ffmpeg',
        '-i', filepath,
        '-c:v', 'libx264',
        '-b:v', '50M',
        '-r', '50',
        '-pix_fmt', 'yuv422p',
        '-colorspace', 'bt709',
        '-c:a', 'pcm_s24le',
        '-ar', '48000',
        output_filepath
    ]

    try:
        subprocess.run(ffmpeg_cmd, check=True)
```

The source and destination folder paths are specified as UNC paths for network shares. The script logs a success message, indicating that the file has been transcoded and saved to the destination folder. Both scripts were compiled to executable files using "pyinstaller".

4.6 MAM importing and workflow design

The final step involves instructing the Media Asset Management (MAM) to import the transcoded file. Once imported, the file is ready for on-air playback as it is or can be further edited as part of a news segment.

We utilized the MAM's built-in configuration manager to define a remote storage location by specifying the UNC path where the automated file conversion script saves the transcoded files. Once the remote storage was defined, we configured a Media Migration Policy to automatically import each new file from the remote storage into the MAM system. Figure 3 illustrates the digital journey of the media file, tracing its path from the mobile phone to the MAM system.

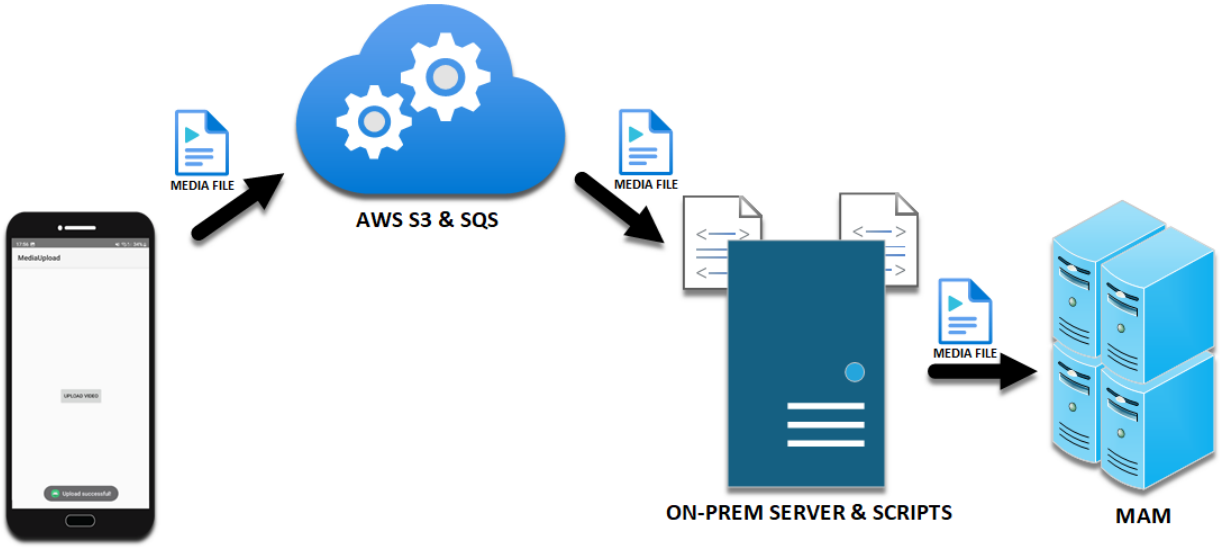

Fig. 3. The progression of the media file through each stage

The workflow is designed to be straightforward, reporters that are dispatched at events to capture interviews or supplementary visual content (b-roll), upload the media using the designated Android application. The uploaded media is expected to appear in the television station's Media Asset Management system (MAM) automatically and faster than through current workflows.

5 Experimental testing

With the framework components now established, this study aims to evaluate the performance of the automated media acquisition framework through comparative analysis against conventional media acquisition workflows. The primary metric of the comparison is on speed, specifically the time elapsed from the moment the footage is completely filmed to its availability in the Media Asset Management system (MAM). This duration is the sum of all steps involved in the media acquisition process. Additionally, the analysis will include a resource perspective, examining the number of personnel involved in the ingest process for each workflow.

Given that both file transfer and transcoding times are highly influenced by video length, all experiments were standardized using one-minute video footage to ensure a fair comparison. The timer begins immediately after the filming is completed, with the one-minute video serving as the point at which the media asset is considered ready for ingestion. Since most of the footage is filmed outside the television station, the filming location also plays an important role in the experiments. In current workflows, reporters often return to the television station with the recording device to ingest the footage, contributing to significant delays between filming and MAM import. For this study, two remote filming locations were selected, one in central Bucharest and another just outside the television station's building. The first remote location is the University's Square, in Bucharest, 4.4 km away from the television station. The second location was chosen to reflect common practices, such as reporters filming updates near the station to provide timely information for evolving news segments. The timer stops once the media asset is successfully imported into the MAM, providing the ingestion speed for each workflow analyzed. Additionally, we will record the number of personnel, apart from the reporter, required for the media asset to be imported. With the video length, remote locations, and defined metrics in place, the experiments are designed to assess and compare the workflows.

5.1 Experiment 1

The first scenario examines one of the most common workflows employed by

broadcasters for media ingestion. In this experiment the location where our standardized one-minute video was filmed is in the center of Bucharest and the reporter returns to the television station with the recording device. The device is then handed over to an ingest operator, who imports the footage into the Media Asset Management system (MAM). For this experiment, the measured time is the sum of several steps: driving the company car back from the remote filming location to the station, handing over the device to the ingest operator and making the ingest request, and completing the ingest process at the ingest workstation. The ingest process consisted of copying the file from the phone to a temporary folder via USB, transcoding it, and importing it to the Media Asset Management (MAM) system. This workflow involves an additional person, the ingest operator, to successfully ingest the filmed media. For this experiment, we assumed optimal conditions where the operator was immediately available and not occupied with other ingest tasks, which would significantly impact the calculated time. This allowed us to measure the best-case scenario completion time for this workflow.

5.2 Experiment 2

The second experiment is like the first one. We chose the same remote location, the city center, but instead of involving the ingest operator, the reporter returns to the newsroom and downloads the file locally from the phone to a temporary location on his work provided laptop, and then moves it to a network shared location set up as a watchfolder for the Media Asset Management (MAM) system own transcoder server. By moving the file to this location, the MAM will transcode and import the file automatically using a transcode and import server. The MAM transcoder server has four channels meaning that it can only handle four files at a time. In most cases there is a queue of files waiting to be transcoded and imported. For this second experiment, we assumed optimal conditions where no files were being queued, because it would significantly increase the measured time. For this experiment, the measured time is the sum of the time it took to drive the company car back from the remote filming location to the television stations newsroom, the time it took the reporter to locally copy the media file and then transfer it to the network share, and finally the time it took the MAM transcoder server to process and import the media asset. This workflow does not involve additional human resources, the reporter being in control of the ingest process.

5.3 Experiment 3

For this experiment we considered the filming location to be right outside the television station, a common place to film last minute updates to a news segment. In this scenario, we measured the time it took the reporter to go to the ingest workstation, hand over the mobile phone to the ingest operator and make the ingest request. We added the time it took for the ingest process to occur and stopped the timer once the media file was successfully imported in the Media Asset Management system. We assumed optimal conditions where the operator was immediately available and not occupied with other ingest tasks. This workflow requires the involvement of the ingest operator as an additional resource to ensure the ingest process is completed successfully.

5.4 Experiment 4

In this experiment the filming location is the same as the one in the previous experiment, right outside the television station, but instead of going to the ingest workstation, the reporter is ingesting the file by transferring it to the MAM transcoder server from his desk in the newsroom. The total measured time is the time it took the reporter to get back to his desk from the filming location, the time it took to transfer the file from the phone to the shared network location and the time it took the MAM transcoder server to process and import the file. This workflow does not involve additional resources besides the reporter.

5.5 Experiment 5

At the television station where we conducted our experiments, each employee has one terabyte of dedicated cloud storage. We tested a

workflow where the reporter uploads the recorded footage to their cloud storage, generates a sharing link, and emails this link and an ingest request to the ingest operator. The ingest operator then downloads the file locally and imports it into the media asset management system. To simulate remote filming conditions, we used the cellular data (GSM) internet connection of the mobile phone rather than Wi-Fi for uploading the files. Our tests showed no significant upload time differences across different city locations, so we decided to use the city center as the remote location for uploading files in this experiment. The total measured time for this experiment consisted of several sequential steps: uploading the file, generating a sharing link, sending the email to the ingest operator, the time the operator spent downloading the media file locally, and finally processing and importing it into the Media Asset Management system. This experiment involves an additional resource, the ingest operator, and we considered the ideal scenario where the operator was available to take on the ingest request immediately.

5.6 Experiment 6

In this experiment, we tested a workflow where a reporter in the city center uploads media files from their remote location to the dedicated cloud storage. They then share the files and email a colleague in the newsroom the link of the shared file. The newsroom reporter reads the email, downloads the files locally and transfers them to the MAM transcoder for import into the Media Asset Management (MAM) system. The measured time includes the duration of uploading the file to the cloud, the time taken by the remote reporter to share the file and email the link to the newsroom reporter, the time required for the newsroom reporter to read the email and download the file locally and transfer it to the MAM transcoder, and finally, the time needed for the transcoder to import the file into the Media Asset Management system. In this workflow, the newsroom reporter is utilized as an additional resource. For this experiment, we considered the ideal scenario where the reporter is free to download the media file immediately and the MAM transcoder server has no queue.

5.7 Experiment 7

The final experiment employs the framework and workflow we developed and presented in this paper. We uploaded the same file using the same device and cellular data connection. The remote location chosen is also the city center. Using our custom-built mobile application, MediaUpload, the reporter uploads media files to the S3 cloud storage. We measured the total process duration from the moment the reporter opens the application and selects the file, through the media's upload to the cloud, automated local download by the cloud monitoring script, transcoding by the local watchfolder script, and ending once the MAM import was complete. The workflow is fully automated after the reporter selects the file in MediaUpload, requiring no additional human resources or manual steps.

5.8 Experiments overview

In each experiment, all uploads were performed using the same cellular data connection and the same mobile device, while all downloads used the company's wired internet connection. Transportation to the television station was provided via company car and driver, which is the standard procedure for reporters covering events. The midday timing was chosen to ensure smoother traffic conditions. The experiments were conducted using the infrastructure of a news television station headquartered in the city of Bucharest, district 6.

6 Results and discussion

Table 1 shows the measured metrics for all the experiments conducted. The key metric is time, but we've also considered the additional human resources necessary, besides the reporter, to successfully import the media asset to the Media Asset Management (MAM) system of the television station as these two factors together reveal each workflow's efficiency. To distinguish the cloud-based workflow experiments, their corresponding time measurements appear highlighted in blue.

Table 1. Experimental results

Experiment	Measured time	Time in seconds	Additional resources
Experiment 1	26 minutes and 17 seconds	1577	1
Experiment 2	27 minutes and 3 seconds	1623	0
Experiment 3	9 minutes and 17 seconds	557	1
Experiment 4	10 minutes and 3 seconds	603	0
Experiment 5	5 minutes and 29 seconds	329	1
Experiment 6	7 minutes and 5 seconds	425	0
Experiment 7	1 minute and 47 seconds	107	0

Figure 4 presents the experimental results sorted by duration, using the fastest time as a baseline and showing the percentage difference between each experiment. Having the fastest time, Experiment 7 is considered the reference.

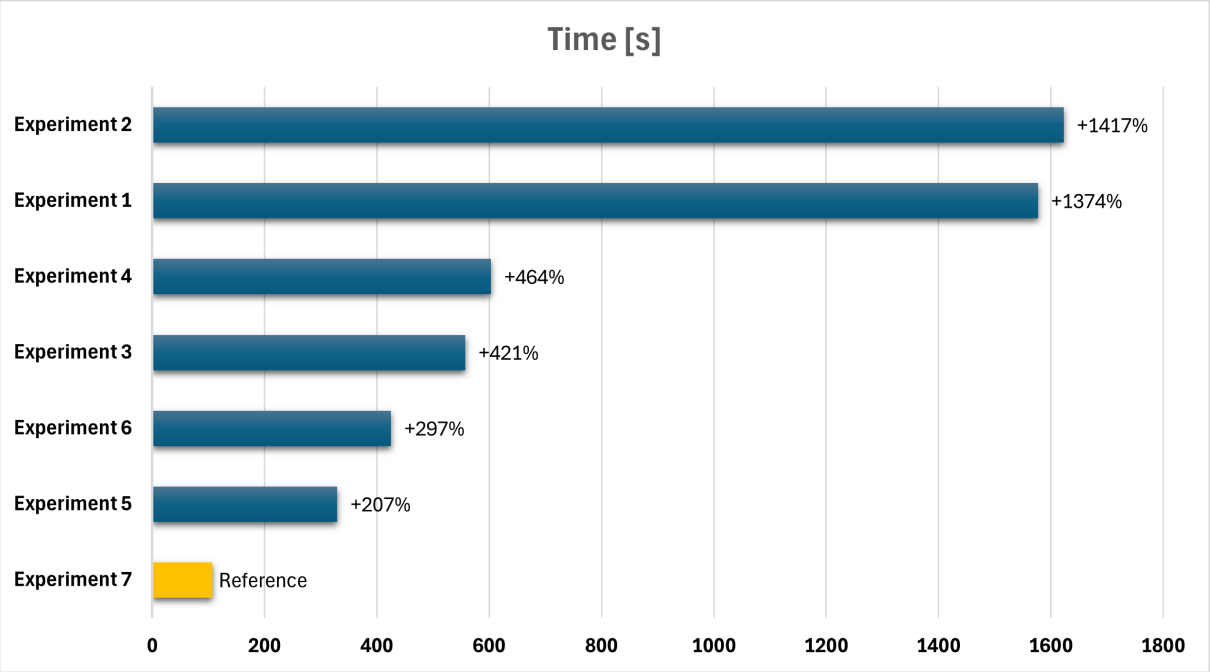

Fig. 4 Experiments performance chart

The performance data reveals a clear advantage for cloud-based workflows, with our automated framework demonstrating particular efficiency. Our solution outperforms other cloud-based approaches by more than three times, primarily due to automating previously manual processes such as file transfers, link sharing, email communication, and transcoding.

While "Experiment 5" achieves the second-best time, it requires help from the ingest operator, an additional human resource that our framework is not requiring. The efficiency gap becomes even more pronounced when comparing against non-cloud workflows. For instance, "Experiment 2," which requires physical transportation of devices to the processing location, takes more than fifteen times longer than our framework.

Our framework achieves two key advantages. First and most important, it's delivering the fastest media asset import times to the Media Asset Management (MAM) system. Second, it eliminates the need for additional human resources during the ingest process. These combined benefits demonstrate that our proposed solution represents the most efficient workflow overall, even when all other experiments are conducted under ideal conditions.

7 Conclusions

This paper presented a framework for

automating the media asset acquisition process through a cloud-based workflow. The solution we developed delivers both accelerated ingest times and reduced resource requirements, which allows operational staff to focus on other priorities. We validated this solution in real-world experiments covering every established workflow of a modern news television station and documented the results. Following the experiments, we presented data demonstrating the advantages of automation and cloud-based workflows over traditional approaches in two key metrics, time and human resources. Substantially reduced media ingest times greatly benefit any content producer, especially news television stations where rapid information delivery is crucial. Faster news delivery often translates to higher viewership, which directly impacts advertising revenue. Stations known for quick, reliable breaking news coverage can command premium advertising rates. Furthermore, eliminating the need for additional personnel, such as ingest operators, potentially provides additional economic benefits.

References

[1] Morikawa, Chamin, et al. "Image and video processing on mobile devices: a survey." The Visual Computer 37.12 (2021): 2931-2949.

[2] Westlund, Oscar. "Mobile news: A review and model of journalism in an age of mobile media." Digital journalism 1.1 (2013): 6-26.

[3] Van Tassel, Joan. "Media Asset Management." NTQ-AUSTIN TEXAS- 6 (1998): 21-28.

[4] Wager, Skiff. "Ingest, manage and distribute." Journal of Digital Asset Management 1 (2005): 157-163.

[5] Jämsä-Jounela, Sirkka-Liisa. "Future trends in process automation." Annual Reviews in Control 31.2 (2007): 211-220.

[6] Sheth, Amit. "Workflow automation: applications, technology and research." ACM SIGMOD Record 24.2 (1995): 469.

[7] Stohr, Edward A., and J. Leon Zhao. "Workflow automation: Overview and research issues." Information Systems Frontiers 3 (2001): 281-296.

[8] Schultz, Fred. "Automation and live television news: Enhanced support for a complex workflow." SMPTE motion imaging journal 112.1 (2003): 29-35.

[9] Restrepo, Pascual. "Automation: Theory, Evidence, and Outlook." Annual Review of Economics 16 (2023).

[10] Georgakopoulos, Diimitrios, Mark Hornick, and Amit Sheth. "An overview of workflow management: From process modeling to workflow automation infrastructure." Distributed and parallel Databases 3 (1995): 119-153.

[11] Fisher, Cameron. "Cloud versus on-premise computing." American Journal of Industrial and Business Management 8.9 (2018): 1991-2006.

[12] Alzakholi, Omar, et al. "Comparison among cloud technologies and cloud performance." *Journal of Applied Science and Technology Trends* 1.1 (2020): 40-47.

[13] Tessem, Bjørnar, Are Tverberg, and Njål Borch. "The future technologies of journalism." Procedia Computer Science 239 (2024): 96-104.

[14] Deb, Moumita, and Abantika Choudhury. "Hybrid cloud: A new paradigm in cloud computing." Machine learning techniques and analytics for cloud security (2021): 1-23.

[15] Rodrigues, Luis Pedro Ribeiro, Vania Baldi, and Adelino de Castro Oliveira Simões Gala. "MOBILE JOURNALISM: the emergence of a new field of journalism." Brazilian journalism research 17.2 (2021): 280-305.

[16] Mahon, James, and PgCert TLHE Dip. "The mojo revolution: A critical evaluation of mobile journalism practice and its impact on journalistic identity." (2021).

[17] Usher, Nikki. "Breaking news production processes in US metropolitan newspapers: Immediacy and journalistic authority." Journalism 19.1 (2018): 21-36.

[18] V. Subhash. " Quick Start Guide to FFmpeg: Learn to Use the Open Source Multimedia-Processing Tool like a Pro." Apress; 1st ed. edition (2023)

Adrian VINTILĂ has graduated the National University of Science and Technology Politehnica Bucharest. He is currently a PhD student at the Bucharest University of Economic Studies. With a strong background in the field of information technology and broadcast engineering, his research focuses on intelligent systems for innovation in media and television production within educational centers.

Constanta-Nicoleta BODEA is Professor Emerita at Bucharest University of Economic Studies, Romania, holding a Ph.D. in Economic Cybernetics and Statistics. She is Senior Researcher at the Centre for Industrial and Services Economics, Romanian Academy of Sciences, and contributes to the Doctoral Study in Project Management at Alma Mater Europaea University, Slovenia. Her research focuses on project management, service innovation, and technology-enhanced teaching and learning. She published more than 50 books and book chapters, and 370 papers in journals and conferences.