IrisBench: An Open-Source Benchmark Suite for Video Processing Systems in Cloud

Zhiqi Li
Carleton University
Ottawa, Canada
zhiqili3@cmail.carleton.ca

Ruiqi Yu
Independent
Beijing, China
yuruiqi422@gmail.com

Jianshu Liu
Boise State University
Boise, USA
jianshuliu@boisestate.edu

Abstract

Recent advances in generative text-to-video AI models (e.g., VideoPoet and Sora) have spurred a surge in video production, leading to an increased demand for video processing pipelines among various video service providers such as YouTube and TikTok. With the improvement of cloud computing, video processing systems are frequently updated and present both opportunities and challenges while optimizing the quality of service (QoS) and cloud resource utilization. However, research on evaluating the performance of video processing systems is limited. Besides the availability of video datasets and realistic workloads, the lack of an open-source benchmark system reflecting the characteristics of industrial video processing systems is a significant gap. To fill this gap, we develop IrisBench, an open-source benchmark suite for cloud video processing systems to facilitate research on performance analysis. Our benchmark suite includes three video services: video transcoding, video partitioning, and video object detection services. Our future work relies on using IrisBench to study the architectural implications of various cloud video processing systems in the cloud.

CCS Concepts

• **General and reference** → *Performance*; **Experimentation**; • **Computer systems organization** → **Cloud computing**.

Keywords

Cloud Computing; Serverless; Stream Processing; Benchmark

ACM Reference Format:
Zhiqi Li, Ruiqi Yu, and Jianshu Liu. 2025. IrisBench: An Open-Source Benchmark Suite for Video Processing Systems in Cloud. In *Companion of the 16th ACM/SPEC International Conference on Performance Engineering (ICPE Companion '25), May 5–9, 2025, Toronto, ON, Canada.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3680256.3721317

1 Introduction

The rapid advancement of generative text-to-video AI models (e.g., VideoPoet [25] and Sora [33]) has significantly transformed the landscape of video production. The increased demand for designing robust and scalable video processing pipelines to guarantee fast and reliable delivery of videos has expanded among video service providers such as YouTube and TikTok. Figure 1 describes a popular

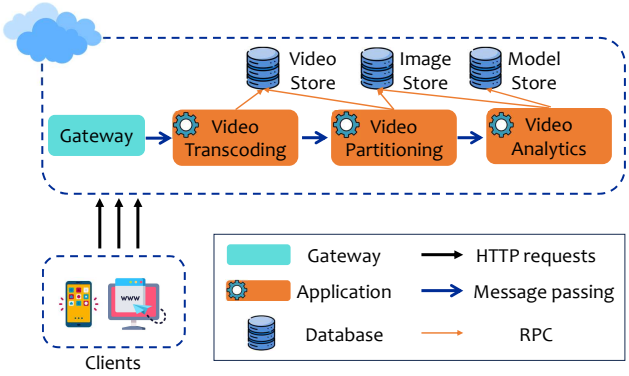


Figure 1: Cloud video processing pipeline.

pipeline for cloud-based video processing systems, which consists of several specialized modules, each dedicated to handling a specific task of video data processing. Based on a scalable cloud infrastructure, video processing systems can handle large volumes of video data efficiently, supporting applications ranging from streaming services to real-time video analytics.

Besides the increasing attention of the industry, video processing systems present numerous opportunities for performance evaluation research in cloud computing. First, video processing workloads are representative of cloud applications with strict Quality of Service (QoS) requirements, making them critical case studies for achieving both good performance and high resource efficiency [12, 18, 28, 32, 37, 42]. Second, video processing is often stateful, relying on robust state management and intensive memory and disk usage, which expands the horizon of existing benchmarks of cloud applications beyond CPU-intensive workloads [10, 23, 27]. Furthermore, video processing systems intersect with interdisciplinary fields, such as computer vision and AI/ML, driving researchers from diverse domains to collaborate for evolving cloud workflow design and the delivery of new products of AI/ML algorithms.

However, systematic performance evaluation research of video processing systems remains limited. Besides the availability of video datasets and realistic workloads, the lack of an open-source benchmark suite reflecting the characteristics of industrial video processing systems is a significant gap. Existing benchmarks only adopt a single architecture and focus on specific tasks in the video processing pipeline. For example, vbench [34] provides a video transcoding framework designed to replicate the wide-ranging demands of platforms like YouTube by using a diverse set of video formats and parameters. However, it lacks considerations for concurrent, multi-user workloads, limiting its scalability insights for

ICPE Companion '25, Toronto, ON, Canada

ACM ISBN 979-8-4007-1130-5/2025/05
https://doi.org/10.1145/3680256.3721317

high-demand environments. Video2Flink [22] offers a distributed video partitioning framework based on a streaming system, but it does not provide publicly available code, limiting its reproducibility. To bridge this gap, we have developed IrisBench, an open-source benchmark suite for cloud-based video processing systems to facilitate research on performance evaluation. Our benchmark currently includes three video services: video transcoding based on Function-as-a-Service (FaaS) architecture, video partitioning using streaming architecture, and FaaS-based video object detection with pre-trained models. We have made IrisBench available at https://github.com/rei-smz/CloudVideoAppBench to encourage further research and collaboration in this domain.

2 The IrisBench Suite

We first describe the suite's design principles and then introduce the overall architecture of our IrisBench. Finally, we present the architecture and functionality of each end-to-end video service in our IrisBench.

2.1 Design Principles

IrisBench adheres to the following design principles:

- **Representativeness:** IrisBench is built using popular open-source applications deployed by cloud providers, such as Nginx [2], Flask [36], MINIO [1], Apache Kafka [14] and Apache PyFlink [15]. We employed Kubernetes [16], an open-source container orchestrator engine, and OpenFaaS [9], a platform for deploying event-driven functions and microservices. Additionally, we utilized Rook [3], specifically its Ceph and Block storage capabilities, as a cloud-native storage orchestrator. Most new code corresponds to benchmark module and video service design, using Golang [4], Python [17], gRPC [21] and HTTP requests.
- **End-to-end operation:** IrisBench implements the full functionality of a service from the moment the requests are sent from users until they reach the service's backend and/or return to the client. Our benchmark client further implements a configurable realistic workload generator to simulate user requests for cloud video services.
- **Heterogeneity:** IrisBench implements applications and benchmark modules using Python, Golang, Java, HTML, and scripting to provide software heterogeneity, which is in alignment with modern software development paradigms, such as CI/CD and DevOps [5].
- **Reconfigurability:** All three applications are easily updated and reconfigured. The video transcoding and object detection are built atop serverless functions, and video partitioning provides convenient plugins to modify the business logic. Besides, the benchmark module provides APIs for loading property files to enhance the reconfigurability of the system and workloads.

2.2 Benchmark Modules

Our IrisBench was developed mainly in Golang and consists of three components: benchmark client, benchmark server, and benchmark data management. Figure 2 describes the overall architecture of IrisBench.

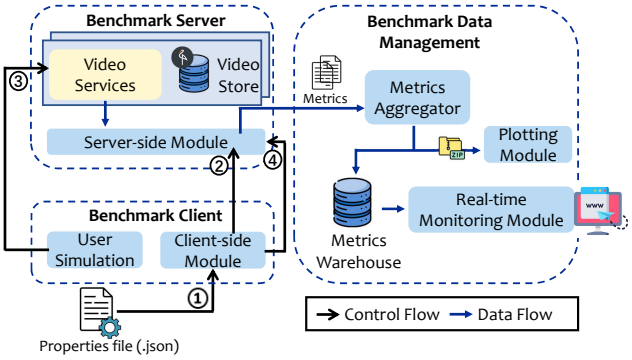


Figure 2: The architecture of IrisBench.

Table 1: Field Descriptions for Benchmark Configuration

Field	Type	Description
n_user	integer	Number of concurrent users
duration	integer	Benchmark duration (seconds)
id_range	integer	Maximum user ID
wait	integer	Waiting time between requests (seconds)
url	string	Service URL
test_name	string	Test name
path_prefix	string	Prefix of the paths to video objects
req_args	JSON Object	Arguments to be sent to the app

Benchmark Client includes a client-side module for workflow control and a user simulation module that emulates the behaviors of users accessing video services by sending concurrent requests to cloud applications. The client-side module manages the configuration of benchmarks and controls the benchmark workflow. First, the client-side module loads the configuration of workloads via JSON files. Table 1 presents the parameters for defining the realistic workloads for video services. Such a design enhances the flexibility of the client-side module, enabling it to support various user requests without requiring code modifications. Second, the client-side module sends a message to the server-side module via gRPC to start monitoring server-side metrics and initiates a timer. Third, the User Simulation module creates a goroutine for each simulated user. Synchronization between the main goroutine and the goroutines handling simulated users is managed via a channel observed in the simulated user goroutines. During the experiment, the simulated user continuously sends user requests to the corresponding video services at intervals defined in the configuration file. Once the timer expires, the synchronization channel is destroyed in the main goroutine, signaling the simulated users that the experiment has ended. Fourth, after all simulated users have completed their tasks after receiving the timer expiration signal, the client-side module sends a message to the server-side module to stop monitoring and save collected metrics. In the meantime, the performance results from the client side are saved to files. These performance results include the timestamps for sending requests, status code, and timestamps for receiving responses, as well as the error rate and the average response time of all user requests. Finally, both

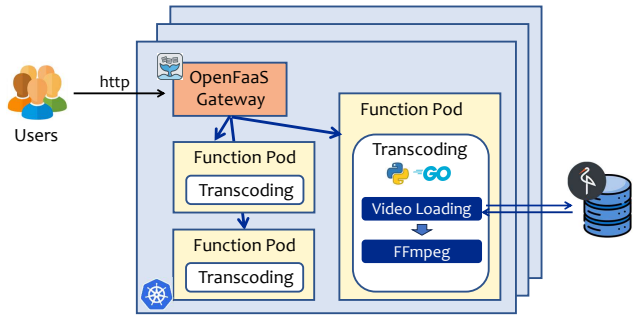


Figure 3: The architecture of video transcoding service.

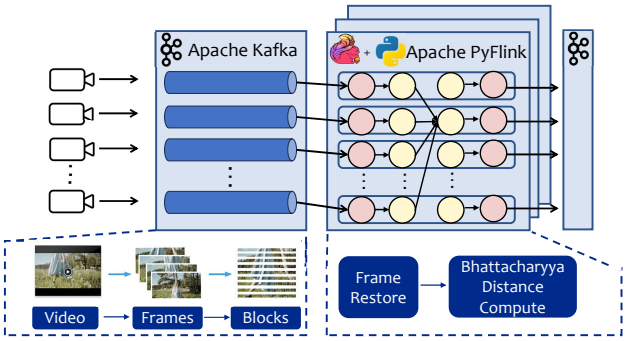


Figure 4: The architecture of video partitioning service.

client-side and server-side metrics will be forwarded to Benchmark Data Management for visualization.

Benchmark Server refers to a server-side module deployed on the cloud cluster. For a Kubernetes cluster, it is deployed explicitly on the master node. The server-side module remains active but on standby when no experiments are running. We implement a submodule serving as an RPC server to listen to the control message from the client-side module. It controls the state of the server-side module via a channel. Upon receiving a "start monitoring" message, the server-side module transitions to a running state. It continuously monitors and collects metrics until the RPC server receives a 'stop monitoring' message, at which point the state reverts to standby. We design an interface with methods for controlling the runtime of the server-side module and a method where the functionality for collecting metrics is to be implemented. Additionally, we provide a base class that implements the runtime control methods, supporting metric collection from different cloud platforms.

Benchmark Data Management is designed to visualize the metrics monitored at runtime. It includes a metrics aggregator module, a plotting module, and a real-time monitoring module. The metrics aggregator module periodically pulls the metrics generated from both client and server-side modules and flushes them to a metrics warehouse. On the one hand, we implement a real-time monitoring module with a web portal to provide support for quickly detecting performance anomalies through a timeline graph of monitored metrics, such as CPU usage, memory usage, and latency of services. It extracts the required metrics from the metrics warehouse asynchronously. On the other hand, the plotting module can automatically generate statistical graphs for performance anomaly analysis by aggregating multiple metrics. For example, we observed the latency degradation and transient CPU bottlenecks from the generated timeline graph from the real-time monitoring module, then the plotting module provides the correlation analysis to verify the relationship between performance and system anomalies, which can help researchers to identify interesting phenomena and inspire in-depth research.

2.3 Video Transcoding

Scope: This application implements an end-to-end service for transcoding videos to the target format and resolution specified by the users (clients).

Design: Figure 3 shows the architecture of the end-to-end service. Since the Function as a Service (FaaS) paradigm is gaining popularity and has the potential to represent the next generation of video processing systems, we designed the video transcoding application atop OpenFaaS and Kubernetes. We deployed Kubernetes and OpenFaaS Community Edition on a cloud cluster. As the FaaS paradigm deals with stateless functions, we integrated an Amazon S3-compatible object storage, MINIO, to manage and store clients' video files. User requests are received and dispatched to Function Pods by OpenFaaS Gateway. We observed that the FaaS paradigm is typically employed for short-duration scenarios due to the ephemeral nature of functions, while video transcoding is inherently time-intensive [24]. Hence, we also deployed a MongoDB database to track the status of requests while preserving the statelessness of FaaS functions. Each status record contains a field indicating the current state (e.g., running, success, or error) and the corresponding start and end timestamps. Upon receiving a user request, the function within a Function Pod creates an entry in the request status management database and returns a request ID to the user instead of providing the status once the request is completed. In the meantime, the invoked function retrieves the corresponding videos from the object storage using the MINIO API and performs the required transcoding operation with user-provided arguments. We rely on FFmpeg [11] to perform video transcoding tasks like converting video codecs (e.g., from H.265 to H.264), audio codecs (e.g., from MP3 to AAC), resolution (e.g., from 1080p to 720p), and video file formats (e.g., from MP4 to MKV). FFmpeg leverages its `libavcodec` library to handle video and audio codec conversions, supporting a wide range of compression and encoding algorithms, such as H.264 and H.265 for video and AAC and MP3 for audio. Besides, FFmpeg utilizes the `libswscale` library to execute the Bicubic interpolation algorithm by default. In contrast to simpler techniques like nearest-neighbor or bilinear interpolation, bicubic interpolation is more effective in reducing artifacts such as blockiness, resulting in smoother and more visually appealing images. It provides better edge transitions, making it a preferred choice for resizing images in many computer vision applications [19]. Once the process is complete, the transcoded video is uploaded back to the object storage using the MINIO API, and the corresponding status in the management database is updated. Users are allowed to query the status of requests using the request IDs from the application.

2.4 Video Partitioning.

Scope: The service implements an end-to-end service that can partition video into meaningful sequences of frames based on detected scene changes.

Design: To process in-stream video from various video sources in real-time, we employ Apache Flink and Apache Kafka to design the video partitioning application, as Apache Flink with Apache Kafka is still a popular solution to facilitate the communication between video production and video processing systems [22]. To improve Apache Flink's compatibility with Python libraries for image processing, we leverage Apache PyFlink, which enables the development of scalable batch and streaming workloads through its Python API. In this paper, we use the terms "Flink" and "PyFlink" interchangeably. Figure 4 describes the architecture of the end-to-end service. Each Kafka producer can independently read multiple distinct video files and extract frames at regular intervals. These frames are then divided into blocks in byte streams and forwarded to a Kafka input topic. Flink, integrated with a Kafka consumer, subsequently consumes the data for processing. Our video partitioning approach is based on comparing color histograms between consecutive video frames. We compute the color histogram to get a 3D color histogram over the RGB channels with 8 bins per channel, resulting in an $8 \times 8 \times 8$ histogram. The histogram captures the distribution of colors in the frame and is then normalized to ensure that the histogram values remain comparable across frames. We track the histogram of the previous frame and compare it with the current frame's histogram using the Bhattacharyya distance. The Bhattacharyya distance is an established metric used to measure the similarity between two probability distributions [6]. It is particularly effective for comparing histograms because it quantifies the overlap between two distributions, providing a value between 0 and 1, where 0 indicates identical distributions and 1 indicates maximum dissimilarity. The Bhattacharyya distance D_B between two normalized histograms P and Q is given by the following formula:

$$D_B(P, Q) = -\ln\left(\sum_{i=1}^{n} \sqrt{P(i)Q(i)}\right) \tag{1}$$

$P(i)$ and $Q(i)$ represent the values of corresponding histogram bins, and n is the total number of bins. The summation is the Bhattacharyya coefficient, which measures the overlap between the two histograms. Then, we set a threshold to filter out frames with low Bhattacharyya distances, retaining only those with higher distances, which are more likely to correspond to significant scene changes.

In the upstream section of the data pipeline, the client extracts video frames at configurable intervals (e.g., every 30 or 60 frames, with smaller intervals representing higher data throughput). Each frame is divided into multiple blocks and converted into byte streams. Each record's key consists of the corresponding video name, the frame's position within the video, and the block's index, while the value holds the byte data for that block. The client then sends this record to the Kafka input topic.

In the downstream section, Flink processes the data, offering operators that facilitate the retrieval of records from Kafka for further processing. First, the blocks associated with each frame are aggregated and reassembled to restore the original frame. Flink reorders the incoming out-of-order data streams based on frame numbers to ensure sequential processing. Subsequently, the Bhattacharyya distance between consecutive frames is computed. Frames with higher Bhattacharyya distances are filtered and selected as keyframes, marking potential video partition points. Finally, the results are sent to the Kafka output topic on the designated node. We have implemented stateless operators, Map and Filter, and stateful operators, Window, KeyBy, and Reduce, to process each record using different logic.

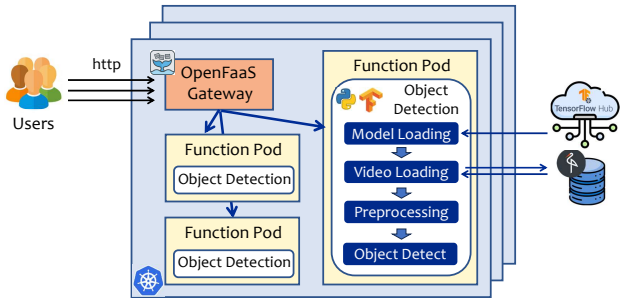


Figure 5: The architecture of video object detection service.

2.5 Video Object Detection.

Scope: The service implements an application that can identify the objects occurring in the provided videos.

Design: Figure 5 depicts the architecture of the end-to-end service. As with the video transcoding application, we designed the video object detection application atop OpenFaaS and Kubernetes with Amazon S3-compatible object storage, MINIO, for video management. The application was built using the official OpenFaaS Python3-Flask Debian template, with dependencies on OpenCV and TensorFlow. We implement a user handler for receiving and parsing user requests. The user request may include video path and object detection model parameters. Then, our object detection service has four phases: *(1) Model Loading Phase*: we can load pre-trained models from external model providers during pod initialization, which enhances the flexibility of our application. For example, we load two deep learning models: SSD-MobileNetV2 [39] for lightweight benchmarks and Inception ResNetV2 [38] for high-performance benchmarks. Both models are sourced from TensorFlow Hub. *(2) Video Loading Phase*: we load the video object from the MINIO database according to the video path inside the user request. *(3) Preprocessing Phase*: the video preprocessing utility first processes the video retrieved from the object storage, which splits the video into key frames (e.g., we can select the first frame of each one-second time window as a keyframe). *(4) Object Detection Phase*: the function within a Function Pod performs the object detection task with pre-trained models. The object detection model takes the preprocessed keyframes as input, performs detection on each frame, and returns the detection results along with the video's width and height to the user request handler. Finally, the user request handler responds to the user with the detection data in JSON format.

3 Example Experiments

In this section, we show example results of our IrisBench implementations to investigate the characteristics of our benchmark under different realistic workloads. Extensive experiments on a larger cluster require substantial efforts and need to be addressed in future work.

3.1 Experimental Setup

Deployment. We run experiments on a cloud cluster hosted on CloudLab [8] to evaluate the functionality of our benchmark system. Our cluster comprises five servers equipped with 2 x Intel Xeon E5-2650v2 processors (8 cores each, 2.6GHz) and 64GB of RAM (8 x 8GB DDR-3 RDIMMs, 1.86GHz). We reserve one node to serve as the client, and the remaining four nodes are designated as servers for running video processing applications.

Video Transcoding. We set up a Kubernetes v1.31.0 cluster for container orchestration and management, with one node acting as the master and the remaining three as worker nodes. We utilize the Rook Ceph and Rook Block Storage class on the Kubernetes cluster. In particular, we deploy MINIO to provide 500GB of storage space for object storage. We deployed OpenFaaS Community Edition on the Kubernetes cluster for the designed serverless functions and installed the OpenFaaS CLI on the master node. We also uploaded transcoding functions with the codebases to the master node. For the benchmark server design, we upload the server-side module to the master node and make it run in the background.

Video Partitioning. We set up a streaming architecture to process in-stream video. We use Apache Kafka-v2.11-1.1.1 as a persistent message queue for the input/output video streams. We use Apache Flink-1.17.1 as the stateful stream processing engine, and the Flink TaskManager memory size was set to 40GB. We configure RocksDB as the state backend in the Flink TaskManager node and employ HDFS as reliable backup storage for runtime states [41].

Video Object Detection. Similar to the setup of video transcoding with serverless functions, we also prepare the codebases of video object detection and upload them to the master node. In particular, object detection supports two modes: lightweight mode (i.e., LW) and high-performance mode (HP).

Video Dataset. Currently, we selected 25 distinct videos from the MOT Challenge dataset [29, 35], a widely used dataset for object detection and tracking in videos, and duplicated them to create a total of 50 videos to simulate 50 users in our system. The videos were in MP4 format with resolutions of 960x540, 640x480, or 768x576, with an average length of 39.48 seconds. The FPS of these videos ranges from 5 to 30, and the codec is MPEG-4 Visual. We have continuous efforts in input video selection by considering the video features, privacy and availability.

Workload and Parameters. Table 2 lists the input parameters used in the experiments. In the example experiments, for the video object detection and video transcoding benchmarks, we specified requesting periods of 300 seconds and 30 seconds, respectively. During the requesting period, each simulated user continuously sends requests, with a 5-second interval between consecutive requests. We increase the number of concurrent users from 5 to 50 in step 5 to evaluate the effect of the concurrency level in our benchmark system.

Service	Pattern	# of simulated user/videos	Input Parameters	Output
Video Transcoding	client-server	[5-50]	video format mp4	Video in mkv format and 1280x720 resolution
Video Partitioning	streaming	[5-50]	video length avg. 39.48s	Time when shot change occurs
Video Object Detection	client-server	[5-50]	video resolution 960x540	Identified objects

Table 2: Workload Parameters of IrisBench

The general outputs of the three video services are also listed in Table 2. We configure the video transcoding service to transcode videos to a resolution of 1280x720 in MKV format and change the video codec to H.264. The video partitioning service detects shot changes inside the video and outputs the timestamp when the shot changes occurred. Video object detection is to detect all identifiable objects based on its training dataset (i.e., COCO dataset).

3.2 Experimental Results

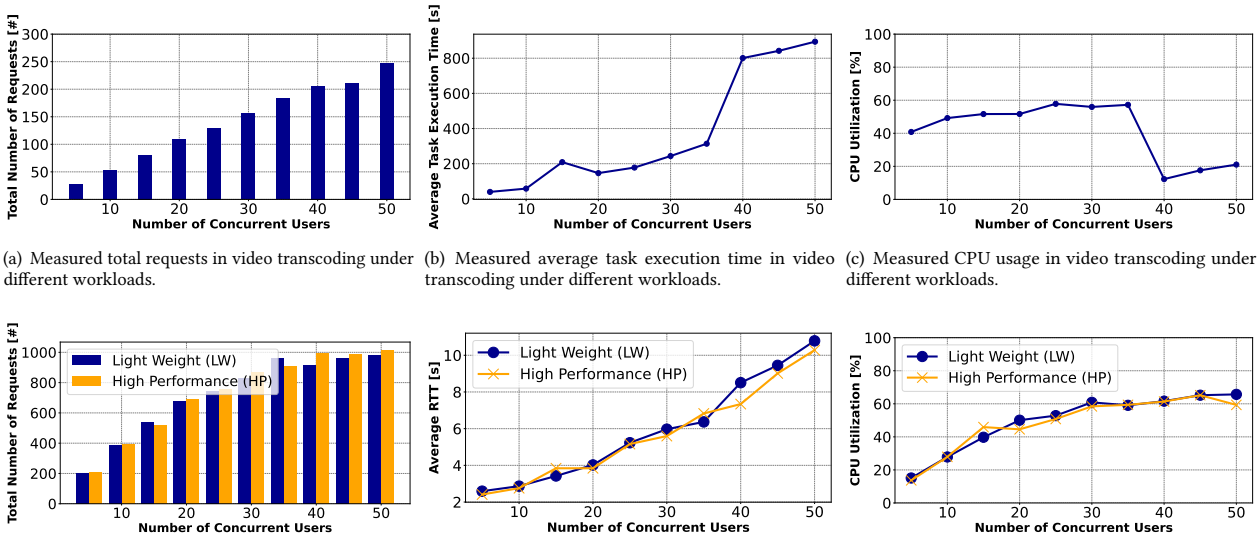
We evaluate our benchmark system's functionality and performance by monitoring practical metrics under realistic workloads with different levels of concurrency.

Figure 6 summarizes the monitored metrics of video transcoding and object detection experiments under different workloads. As the workload concurrency increases, Figure 6(a) and Figure 6(d) record the total number of requests processed in the application. Figure 6(b) and Figure 6(e) show that the average task execution time increases linearly as the number of users increases. In particular, a sharp rise is observed when the number of users increases from 35 to 40 in Figure 6(b). Meanwhile, Figure 6(c) and Figure 6(f) capture the average CPU usage of the cluster. The drop in CPU usage after the number of users reached 40 occurred because, in these cases, the system spent several periods executing retries for a few requests while many other requests had already been completed successfully, leaving some pods idle. These results show the functionality of our benchmark. The comprehensive performance evaluation analysis lies in our future work.

4 Related Work

Cloud applications have attracted a lot of attention over the past decade, with various benchmark suites emerging from both academia and industry. These benchmarks can broadly be categorized into traditional cloud benchmarks, microservices benchmarks, and specialized benchmarks for video processing. While many frameworks perform well within single or simplified multi-tier environments, they fall short in addressing the complexities of large-scale, concurrent multi-user cloud applications. Additionally, most prior work does not utilize FaaS methodologies [7, 30, 40], which we argue are essential for flexible scalability in modern cloud applications. To meet these evolving requirements, our work introduces a method that accommodates multi-user demands through a scalable, FaaS-enabled architecture, capturing the interdependencies and real-world complexities that prior approaches do not fully address.

Cloud Benchmarks. With the rise of microservices, recent work [12, 18, 28, 32, 37, 42] has developed benchmarks that specifically address their unique characteristics. μSuite [37], for example,



(a) Measured total requests in video transcoding under different workloads.

(b) Measured average task execution time in video transcoding under different workloads.

(c) Measured CPU usage in video transcoding under different workloads.

(d) Measured total requests in video object detection under different workloads.

(e) Measured average response time in video object detection under different workloads.

(f) Measured CPU usage in video object detection under different workloads.

Figure 6: Performance evaluation on the effect of concurrency level (# of concurrent users).

targets sub-millisecond latency measurements and analyzes the effects of OS scheduling, network protocols, and RPCs on microservices' response times. Liu et al. [32] focus on the evaluation of containerized edge-cloud computing stacks for industrial applications from a client's perspective, examining the impact of factors like message-sending intervals, payload size, network bandwidth, and concurrent devices on system latency and processing capability. DeathStarBench [18] diverges by targeting large-scale applications with numerous unique microservices, thus revealing phenomena like network contention and cascading QoS issues caused by inter-tier dependencies. Our IrisBench targets the video processing pipelines and emulates real-world deployment in large-scale cloud environments.

Video Processing Benchmarks. In response to the growing demands of video processing in cloud environments, specialized benchmarks have been developed [7, 22, 26, 30, 31, 40]. vbench [34] provides a video transcoding framework designed to replicate the wide-ranging demands of platforms like YouTube by using a diverse set of video formats and parameters. Fouladi et al. [13] developed a low-latency transcoding system using AWS Lambda, adhering to a FaaS paradigm, which is particularly useful for scenarios requiring rapid scaling. SVE [20] is a scalable, distributed system designed for efficient multi-task video processing that meets the extensive computational requirements of Facebook's video services infrastructure. Our IrisBench supports diverse tasks in video processing pipelines and adopts popular FaaS and streaming architectures for performance evaluation.

5 Limitations and Future Work

IrisBench has several limitations that we plan to address in future work. First, the scalability of IrisBench is currently constrained by the OpenFaaS Community Edition, which prevents comprehensive evaluations of function scaling in a large-scale production environment under high-concurrency workloads. Future research will explore alternative FaaS platforms with more flexible scalability options, such as Knative, to extend IrisBench 's applicability to large-scale serverless video workloads. Meanwhile, a large-scale performance evaluation will be conducted for studying the architectural implications. For example, our experimental results of video object detection services using lightweight and high-performance models have shown minimal performance variation, necessitating further investigation into underlying architectural factors.

Second, IrisBench relies on the Kubernetes metrics server for performance monitoring, which introduces limitations in accuracy and granularity. Currently, metrics are indirectly obtained via the Kubernetes metrics server, which may affect the precision of benchmarking results. Future iterations of IrisBench will incorporate direct metric extraction from Prometheus and the Kubelet metrics API, ensuring higher accuracy in performance measurement and system profiling.

Finally, a visualization framework for real-time performance monitoring of IrisBench is still in progress. To enhance observability, future versions will deploy an InfluxDB-backed storage system for historical performance data and a Grafana-based web interface for real-time visualization. This will provide researchers with a more comprehensive view of workload execution, supporting in-depth performance analysis and system tuning.

6 Conclusions

In this paper, we present the design of IrisBench, a benchmark suite for cloud-based video processing, addressing the critical need for open-source tools for performance analysis research. Our benchmark supports multi-task processing with components for video transcoding, partitioning, and object detection, allowing for detailed performance evaluations across diverse video workloads.

Our future work is to use IrisBench to explore the architectural implications of modern cloud video processing systems, supporting research into cloud infrastructure and performance optimization of video-centric applications.

References

[1] [n. d.]. MinIO. https://min.io/. Accessed: 2024-11-09.
[2] [n. d.]. NGINX. https://www.nginx.com/. Accessed: 2024-11-09.
[3] Rook Authors. 2018. Rook (Ceph and Block). https://rook.io/. Accessed: 2024-11-09.
[4] The Go Authors. 2009. The Go Programming Language. https://go.dev/. Accessed: 2024-11-09.
[5] Shashikant Bangera. 2018. *DevOps for Serverless Applications: Design, deploy, and monitor your serverless applications using DevOps practices.* Packt Publishing Ltd.
[6] Anil Kumar Bhattacharyya. 1943. On a measure of divergence between two statistical populations defined by their probability distributions. *Bulletin of the Calcutta Mathematical Society* 35 (1943), 99–109.
[7] Chavit Denninnart and Mohsen Amini Salehi. 2024. SMSE: A serverless platform for multimedia cloud systems. *Concurrency and Computation: Practice and Experience* 36, 4 (2024), e7922.
[8] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. 2019. The Design and Operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC)*. 1–14. https://www.flux.utah.edu/paper/duplyakin-atc19
[9] Alex Ellis. 2017. OpenFaaS. https://www.openfaas.com/. Accessed: 2024-11-09.
[10] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. 2012. Clearing the clouds: a study of emerging scale-out workloads on modern hardware. *Acm sigplan notices* 47, 4 (2012), 37–48.
[11] FFmpeg Team. 2024. FFmpeg. https://ffmpeg.org/.
[12] Stefan Fischer, Pirmin Urbanke, Rudolf Ramler, Monika Steidl, and Michael Felderer. 2024. An Overview of Microservice-Based Systems Used for Evaluation in Testing and Monitoring: A Systematic Mapping Study. In *Proceedings of the 5th ACM/IEEE International Conference on Automation of Software Test (AST 2024)*. 182–192.
[13] Sadjad Fouladi, Riad S. Wahby, Brennan Shacklett, Karthikeyan Vasuki Balasubramaniam, William Zeng, Rahul Bhalerao, Anirudh Sivaraman, George Porter, and Keith Winstein. 2017. Encoding, Fast and Slow: Low-Latency Video Processing Using Thousands of Tiny Threads. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 363–376. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/fouladi
[14] Apache Software Foundation. 2011. Apache Kafka. https://kafka.apache.org/. Accessed: 2024-11-09.
[15] Apache Software Foundation. 2019. Apache PyFlink. https://flink.apache.org/. Accessed: 2024-11-09.
[16] Cloud Native Computing Foundation. 2014. Kubernetes. https://kubernetes.io/. Accessed: 2024-11-09.
[17] Python Software Foundation. 1991. Python. https://www.python.org/. Accessed: 2024-11-09.
[18] Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayan Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, et al. 2019. An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 3–18.
[19] Rafael C. Gonzalez and Richard E. Woods. 2008. *Digital Image Processing* (3rd ed.). Pearson Prentice Hall.
[20] Qi Huang, Petchean Ang, Peter Knowles, Tomasz Nykiel, Iaroslav Tverdokhlib, Amit Yajurvedi, Paul Dapolito, Xifan Yan, Maxim Bykov, Chuen Liang, Mohit Talwar, Abhishek Mathur, Sachin Kulkarni, Matthew Burke, and Wyatt Lloyd. 2017. SVE: Distributed Video Processing at Facebook Scale. In *Proceedings of the 26th Symposium on Operating Systems Principles* (Shanghai, China) *(SOSP '17)*. Association for Computing Machinery, New York, NY, USA, 87–103. doi:10.1145/3132747.3132775
[21] Google Inc. 2015. gRPC. https://grpc.io/. Accessed: 2024-11-09.
[22] Dimitrios Kastrinakis and Euripides GM Petrakis. 2023. Video2Flink: real-time video partitioning in Apache Flink and the cloud. *Machine Vision and Applications* 34 (2023), 42.
[23] Harshad Kasture and Daniel Sanchez. 2016. Tailbench: a benchmark suite and evaluation methodology for latency-critical applications. In *2016 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 1–10.
[24] Jeongchul Kim and Kyungyong Lee. 2019. Practical cloud workloads for serverless faas. In *Proceedings of the ACM Symposium on Cloud Computing*. 477–477.
[25] Dan Kondratyuk, Lijun Yu, Xiuye Gu, José Lezama, Jonathan Huang, Grant Schindler, Rachel Hornung, Vighnesh Birodkar, Jimmy Yan, Ming-Chang Chiu, et al. 2023. Videopoet: A large language model for zero-shot video generation. *arXiv preprint arXiv:2312.14125* (2023).
[26] Tajinder Kumar, Purushottam Sharma, Jaswinder Tanwar, Hisham Alsghier, Shashi Bhushan, Hesham Alhumyani, Vivek Sharma, and Ahmed I Alutaibi. 2024. Cloud-based video streaming services: Trends, challenges, and opportunities. *CAAI Transactions on Intelligence Technology* 9, 2 (2024), 265–285.
[27] Supreeth Kurpad, BT Smruthi, Swarupa Vijaykumar, Suvigya Jain, and Subramaniam Kalambur. 2023. Microarchitectural Analysis and Characterization of Performance Overheads in Service Meshes with Kubernetes. In *2023 3rd Asian Conference on Innovation in Technology (ASIANCON)*. IEEE, 1–6.
[28] Rodrigo Laigner, Zhexiang Zhang, Yijian Liu, Leonardo Freitas Gomes, and Yongluan Zhou. 2024. A benchmark for data management in microservices. (2024).
[29] L. Leal-Taixé, A. Milan, I. Reid, S. Roth, and K. Schindler. 2015. MOTChallenge 2015: Towards a Benchmark for Multi-Target Tracking. *arXiv:1504.01942 [cs]* (April 2015). http://arxiv.org/abs/1504.01942 arXiv: 1504.01942.
[30] Pawissanutt Lertpongrujikorn and Mohsen Amini Salehi. 2023. Object as a service (oaas): Enabling object abstraction in serverless clouds. In *2023 IEEE 16th International Conference on Cloud Computing (CLOUD)*. IEEE, 238–248.
[31] Xiangbo Li, Mohsen Amini Salehi, Yamini Joshi, Mahmoud K Darwich, Brad Landreneau, and Magdy Bayoumi. 2018. Performance analysis and modeling of video transcoding using heterogeneous cloud services. *IEEE Transactions on Parallel and Distributed Systems* 30, 4 (2018), 910–922.
[32] Yu Liu, Dapeng Lan, Zhibo Pang, Magnus Karlsson, and Shaofang Gong. 2021. Performance evaluation of containerization in edge-cloud computing stacks for industrial applications: A client perspective. *IEEE Open Journal of the Industrial Electronics Society* 2 (2021), 153–168.
[33] Yixin Liu, Kai Zhang, Yuan Li, Zhiling Yan, Chujie Gao, Ruoxi Chen, Zhengqing Yuan, Yue Huang, Hanchi Sun, Jianfeng Gao, et al. 2024. Sora: A review on background, technology, limitations, and opportunities of large vision models. *arXiv preprint arXiv:2402.17177* (2024).
[34] Andrea Lottarini, Alex Ramirez, Joel Coburn, Martha A Kim, Parthasarathy Ranganathan, Daniel Stodolsky, and Mark Wachsler. 2018. vbench: Benchmarking video transcoding in the cloud. *ACM SIGPLAN Notices* 53 (2018), 797–809.
[35] A. Milan, L. Leal-Taixé, I. Reid, S. Roth, and K. Schindler. 2016. MOT16: A Benchmark for Multi-Object Tracking. *arXiv:1603.00831 [cs]* (March 2016). http://arxiv.org/abs/1603.00831 arXiv: 1603.00831.
[36] Armin Ronacher. 2010. Flask. https://flask.palletsprojects.com/. Accessed: 2024-11-09.
[37] Akshitha Sriraman and Thomas F Wenisch. 2018. μ suite: a benchmark suite for microservices. In *2018 ieee international symposium on workload characterization (iiswc)*. IEEE, 1–12.
[38] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander Alemi. 2017. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 31.
[39] TensorFlow. [n. d.]. SSD MobileNetV2 (TensorFlow2). https://www.kaggle.com/models/tensorflow/ssd-mobilenet-v2/TensorFlow2/ssd-mobilenet-v2/1.
[40] Huaizheng Zhang, Meng Shen, Yizheng Huang, Yonggang Wen, Yong Luo, Guanyu Gao, and Kyle Guan. 2021. A serverless cloud-fog platform for dnn-based video analytics with incremental learning. *arXiv preprint arXiv:2102.03012* (2021).
[41] Shungeng Zhang, Qingyang Wang, Yasuhiko Kanemasa, Julius Michaelis, Jianshu Liu, and Calton Pu. 2022. ShadowSync: latency long tail caused by hidden synchronization in real-time LSM-tree based stream processing systems. In *Proceedings of the 23rd ACM/IFIP International Middleware Conference*. 281–294.
[42] Yanqi Zhang, Zhuangzhuang Zhou, Sameh Elnikety, and Christina Delimitrou. 2024. Ursa: Lightweight Resource Management for Cloud-Native Microservices. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 954–969.