






# ChimeraPy: A Scientific Distributed Streaming Framework for Real-time Multimodal Data Retrieval and Processing


Eduardo Davalos   
Department of Computer Science  
Vanderbilt University  
Nashville, TN, USA  
eduardo.davalos.anaya@vanderbilt.edu

Umesh Timalsina   
Institute for Software Integrated Systems  
Vanderbilt University  
Nashville, TN, USA  
umesh.timalsina@vanderbilt.edu

Yike Zhang   
Department of Computer Science  
Vanderbilt University  
Nashville, TN, USA  
yike.zhang@vanderbilt.edu

Jiayi Wu   
Department of Computer Science  
Vanderbilt University  
Nashville, TN, USA  
jiayi.wu@vanderbilt.edu

Joyce Horn Fonteles   
Department of Computer Science  
Vanderbilt University  
Nashville, TN, USA  
joyce.h.fonteles@vanderbilt.edu

Gautam Biswas   
Department of Computer Science  
Vanderbilt University  
Nashville, TN, USA  
gautam.biswas@vanderbilt.edu

**Abstract**—Multimodal data analysis provides profound insights into behaviors and interactions within various settings. However, the collection and analysis of this data in real-world scenarios are intricate and resource-intensive. To streamline these processes, we introduce ChimeraPy: an open-source, distributed streaming platform optimized for high-throughput data transfer across processing nodes within a computer cluster. The utility and performance of ChimeraPy are showcased through two benchmark applications, highlighting its capability to handle complex data environments.

**Index Terms**—multimodal, multimedia, distributed, streaming, artificial intelligence

## I. INTRODUCTION

In today’s AI-driven era, reliable data is a key asset, invaluable for providing insights into our world and lives. The deployment of big data systems and advanced sensors has enabled the collection of extensive telemetry data from various sources, including the environment, operational systems, and internet users. Enriching AI models with diverse media types enhances our ability to make predictions and inferences. However, managing heterogeneous sensor data for multimodal analysis is complex [1].

Multimodal data analysis, crucial for understanding complex behaviors and interactions, poses technical challenges in collection and processing. It requires significant expertise and careful engineering to maintain data integrity [1].

While distributed frameworks like Apache Flink [2] excel in big data analysis, their complexity and deployment demands limit their use in rapid scientific application development. Simpler systems like Dask [3] do not support multimodal data (MMD) collection and alignment [3]. Moreover, existing MMD systems like PSI are not easily integrated with state-of-the-art Python-based AI and machine learning methods [4].

These gaps in software infrastructure for multimodal analysis and the challenges of implementing distributed MMD pipelines hinder data science practitioners and AI researchers from leveraging advanced multimodal AI algorithms in online applications. Addressing this, we propose a distributed framework specifically designed for data and learning scientists to deploy cutting-edge MMD pipelines, enabling efficient online collection and processing.

### A. Framing Our Approach

The increasing complexity of deep learning models processing MMD necessitates more sophisticated pipelines for MMD collection, streaming, and processing. Traditional MMD collection systems often entail complex engineering costs, impeding MMD research. **ChimeraPy**, our minimal setup distributed streaming platform, aims to address these challenges. It is optimized for high throughput MMD transfer, time-aligned MMD collection, and scalable Big MMD analytics. Our development was guided by four key architecturally defining requirements (ADRs).

*a) Minimal Setup Costs:* Many distributed computing frameworks are intricate and time-consuming to set up, making rapid deployment in research settings challenging. ChimeraPy emphasizes quick deployment and prototyping.

*b) Time Alignment:* For effective multimodal analytics, MMD collected from different sources must be time-aligned. This alignment is crucial for accurate inference and prediction in various environments.

*c) Data Availability:* Generic computing frameworks often overlook the distributed and heterogeneous nature of MMD sources. ChimeraPy addresses this by ensuring the availability and local extraction of data streams.

d) *Ethical Concerns*: Uploading all collected multimedia data to the cloud raises significant privacy concerns. ChimeraPy advocates for in-network de-identification of sensitive data like video or GPS before internet transmission.

## B. Contributions

We summarize the key contributions of this work:

- *Minimal Setup Solution*: ChimeraPy offers a streamlined, Python-based setup for distributed streaming and processing of scientific MDD, ensuring ease of use.
- *Generality of Pipeline Design*: Demonstrating its versatile and efficient architecture, ChimeraPy excels in processing MMD in applications such as classrooms and museums.

All code for this project is open-source under GPL-3, accessible at <https://github.com/ChimeraPy>.

## II. RELATED WORK

This section reviews pre-existing MMD collection tools and general-purpose distributed (GPD) frameworks that support complex computation tasks. We study the architectural features and compare five currently available MMD collection tools in terms of pipeline complexity, effective utilization of multiple computational resources, and deployment speed. We discuss how pre-existing GPD frameworks are not well-equipped to be MMD collection and processing tools.

### A. Multimodal Data Collection Systems

The demand for MMD processing, storage, and analysis has led to the development of various solutions. Among these, iMotions [5] offers a comprehensive suite for MMD research, but its proprietary nature and licensing costs restrict scalability and adoption. The platform also limits the integration of new analytical methods. In contrast, open-source projects like EZ-MMLA Toolkit [6], MediaPipes [7], SSI [8], and PSI [4] have emerged, providing some advances over closed-source software but still falling short of a streamlined architecture for MMD research. EZ-MMLA and MediaPipes lack time alignment and support only unimodal data processing.

SSI and PSI, both designed for aligning temporal signals in multimodal data (MMD) collection, represent advancements in real-time data gathering. SSI offers a C++ API complemented by Python plugin support, while PSI is developed on the .NET framework. They use directed acyclic graphs (DAG) for data flows but have limitations for distributed AI systems. SSI is limited to single-machine use and lacks native distributed support. PSI, though allowing distributed pipelines, is less compatible with Python AI models and has a cumbersome setup process. In contrast, our approach with ChimeraPy utilizes a top-down deployment with mutable workers, offering a more efficient pipeline setup. Next, we explore how GPD frameworks might address or adapt to MMD collection challenges.

### B. Distributed Frameworks

GPD frameworks, optimized for varied use cases, encompass messaging, processing, and streaming frameworks, often as free and open software (FOSS) with diverse architectures.

a) *Messaging*: Frameworks like Kafka [9] enable high-volume data transfer across devices but are time-consuming to set up and modify due to Java dependencies. They lack built-in multimedia processing, necessitating additional tools, and are less suitable for applications requiring quick deployment.

b) *Computing*: Computing frameworks, mostly following the MapReduce programming model [10], excel in processing large datasets via worker nodes, achieving data parallelism through distributed data processing. However, these frameworks, such as Dask [3], focus on task parallelism and aren't designed for streaming workloads.

c) *Streaming*: Apache Spark Streaming [11] and Apache Flink [2], integrated with Apache Kafka for data input, are adept at structured data processing but fall short in handling multimedia formats. They depend on external tools for data extraction and are not ideal for rapid, low-cost setup environments.

Apache Storm [12] supports data processing and collection but has limited Python integration, requiring manual dependency management and suffering from reduced throughput in larger data streams due to JSON subprocess communication.

In summary, our review of GPD frameworks for MMD collection revealed gaps in current tools regarding our four ADRs. While some frameworks show promise in specific areas, they face challenges in multimedia support, streaming adequacy, and comprehensive data collection. The need for an efficient, Python-friendly MMD collection and processing framework led to the development of ChimeraPy.

## III. CHIMERAPY FRAMEWORK COMPONENTS

We developed ChimeraPy to address the limitations of existing data collection and processing frameworks for educational and research applications. Our design process involved consultations with end-users, power users, and researchers/engineers to understand their needs and concerns, leading to several initial prototypes. These efforts culminated in the ChimeraPy design, a *managed peer-to-peer* (P2P) local cluster with *human-in-the-loop* controls for distributed MMD pipeline management.

ChimeraPy operates as a DAG-modeled distributed MMD pipeline within a local cluster. A central coordinator computer facilitates updates and requests, while each worker in the cluster hosts multiple graph nodes for specific operations. The cluster's web application dashboard allows deployment, monitoring, and visualization of pipeline execution, serving as the control panel.

ChimeraPy's programming model is a DAG of user-defined nodes, allowing complex operations for versatile data pipeline functionality. As illustrated in Fig. 1, the task network includes source nodes for data extraction, step nodes for data operations with return values, and sink nodes for outputs. MMD streams, forming the DAG's edges, are transmitted between P2P nodes via socket programming.

Our design integrates network connection typologies from the outset, considering the diverse MMD entry points in scientific research. The framework merges server-client and

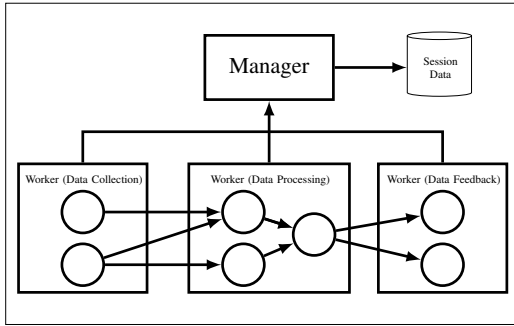


Fig. 1. Selected Network Typologies and Architectures: Server-client typology supports the development of a centralized controller for the distributed cluster. P2P typology can be leveraged to physically model the data pipelines’ DAG

P2P architectures, balancing centralized coordination and distributed processing. This combination addresses computational bottlenecks and ensures data availability and accessibility.

a) *System Design*: ChimeraPy consists of two main subsystems: (1) the Application Interface (Web controller via REST API and CLI) and (2) the Framework Interface (a low-level P2P networking engine), detailed in Fig. 2. The Application Interface, built with FastAPI and Svelte, functions as the cluster’s controller for pipeline design, deployment, and monitoring. The Framework Interface, extending ZeroMQ, enables real-time data streaming. This subsystem division ensures safe human and programmatic interactions, simplifying the framework’s complexity.

The *Application Interface* features a finite state machine (FSM), a dashboard, REST APIs, and a plugin registry. The FSM ensures cluster operation safety, the FastAPI server offers a REST API for dashboard and CLI-based system administration, and the plugin registry facilitates external node integration. The dashboard, serving as the central control panel, includes a pipeline designer with plugins and tools for managing the pipeline and cluster.

The *Framework Interface* comprises three actors: the Manager, Worker, and Node. The Manager orchestrates cluster operations and manages the cluster’s state, including deploying the pipeline in a top-down manner. The Workers, hosting the Nodes, follow the Manager’s instructions and operate their HTTP servers for bidirectional communication. Nodes, embodying DAG nodes, have three functions: `setup`, `step`, and `teardown`, executed during the cluster’s lifecycle for service management and data processing. Nodes, categorized as source, step, or sink, communicate through WebSocket and ZeroMQ. Pipeline outputs are stored locally for offline analysis, with data eventually transferred to the Manager’s file system post-session.

#### IV. APPLICATIONS

In this section, we provide an overview of two sample applications (face detection and sentiment analysis), representative of the use cases we envision for the framework. While ChimeraPy

has been used to develop multiple applications<sup>1</sup> of varying complexity, the two applications illustrated here showcase two of the modalities currently supported in ChimeraPy and address three primary concerns for measuring framework performance<sup>2</sup> (a) **Computational Uniformity**: this ensures that nodes can be randomly assigned to any Worker; (b) **Artifacts Generation**: to ensure that the pipelines process and generate meaningful artifacts for collection and archiving; and (c) **Computational Complexity**: the pipelines should be comparable in complexity with the ones presented in DSPBench [13]. We use the DSPBench benchmarks to demonstrate the properties and effectiveness of the ChimeraPy framework.

##### A. Face Detection

The face detection pipeline, illustrated in Fig. 3(a), is composed of a `Camera` node that captures input from a computer’s webcam. The input frames from the `Camera` node are then directed to a `GrayScaler` node, which transforms the BGR images into grayscale images. Following this, the frames are passed to a `FaceDetector` node that identifies faces using OpenCV’s cascade classifier [14], crops them, and subsequently stores these cropped faces within a frame. Ultimately, the processed frames, containing the detected faces, are presented through a `Display` node in the form of videos.

##### B. Sentiment Analysis

The sentiment analysis pipeline presented in Fig. 3(b), comprises a `TwitterSource` node responsible for emulating real-time tweets (we filter out non-English-language tweets) generated at random and configurable intervals by reading from a CSV file. This is followed by a `Tokenizer` node, which conducts sentence tokenization using `nltk` [15]. The outputs from the tokenizer node are then directed to a `SentimentAnalyzer`, which employs `VADER` [16] to compute polarity scores for the tokenized sentences. Subsequently, the polarity scores, in conjunction with the sentences, are fed into an `Aggregator` node. This node aggregates the polarity scores for the tweets and stores a CSV file containing the tweets and the predicted aggregate sentiment.

#### V. BENCHMARKS

This benchmark study implements ChimeraPy on a local area network, primarily running on low-resource computers. Establishing resource needs and overall cluster performance are key factors for demonstrating the feasibility of this framework. We adopt the methodology presented in [13] to illustrate the benchmark results for the two aforementioned applications.

##### A. Experimental Setup and Metrics

The experiments conducted in a local setup included four Windows processors (the Workers) and a Linux processor (the Manager). All processors used the same LAN via a TPLink Archer AX73 router. The Worker processors had the following specifications: (a) **Manufacturer**: Dell; (b) **CPU**: 11<sup>th</sup> Gen

<sup>1</sup><https://github.com/ChimeraPy/Pipelines>

<sup>2</sup><https://github.com/ChimeraPy/Benchmarks>

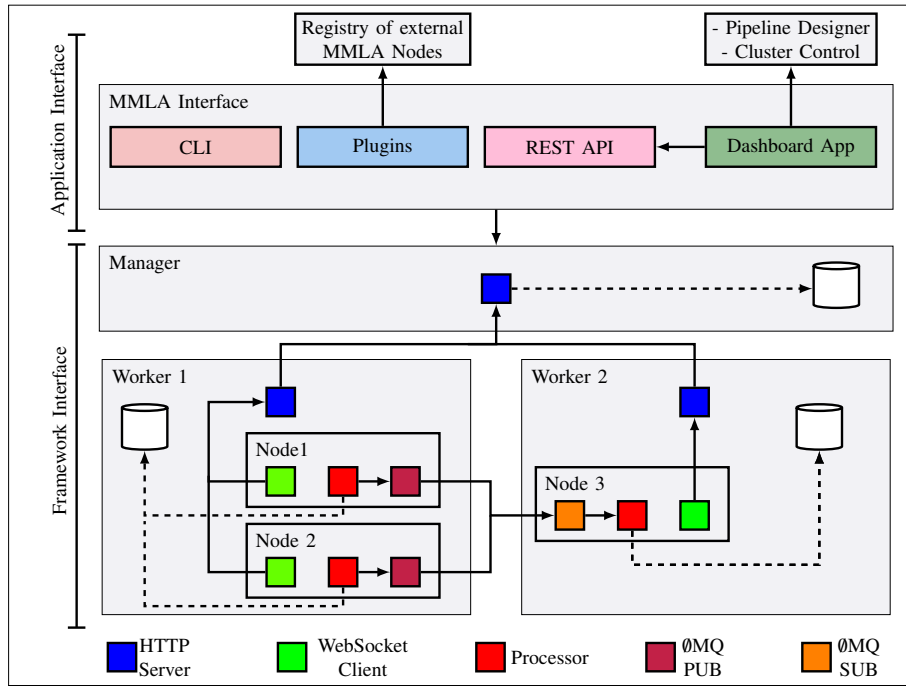


Fig. 2. Detailed System Diagram: Broken into 2 sections: (1) Framework Interface, composed of the Manager, Worker, and Node, performs the cluster setup, configuration, execution, and data aggregation. (2) Application Interface, made up of a CLI, plugin registry, REST API, and a Web app, provides the tooling and scaffolding for pipeline design, deployment, and orchestration.

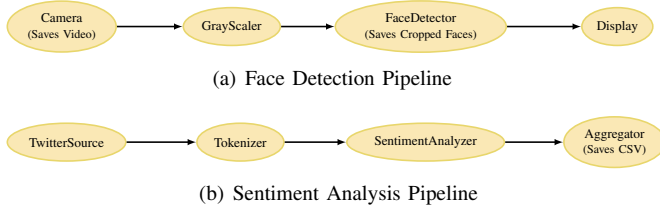


Fig. 3. Pipelines for face detection and sentiment analysis, represented as ChimeraPy DAGs.

Intel(R) Core(TM) i7-11390H @ 3.40GHz; (c) **RAM**: 16GB; and (d) **Operating System**: Windows 11.

The Manager instance had the following specifications: (a) **Manufacturer**: Dell; (b) **CPU**: Intel(R) Core(TM) i9-10900X CPU @ 3.70GHz; (c) **RAM**: 66GB; and (d) **Operating System**: Ubuntu 22.04.

For both applications, we ran experiments with 1, 2, and 4 workers. Depending on the number of available workers, we randomly assigned nodes to the workers in a round-robin fashion. Each pipeline was run for 200 seconds and the runs were repeated 5 times to balance the results. For example, if face detection was run with two workers the nodes were assigned randomly to each of the 2 workers (round-robin). After a successful commit, the pipeline was run( $r$ ) 5 times representing the experiment configuration as  $w2-r5-t200$ .

For each run of the pipelines, we recorded the **Step Time**, **CPU Usage**, **Memory Usage**, and **Payload Size** per node and **Setup Time**, **Teardown Time**, and **Collection Time** per run. These are defined as:

- **Setup Time**: Time to set up a pipeline;
- **Teardown Time**: Time to release resources;
- **Collection Time**: Time to gather artifacts to Manager.
- **Step Time**: Average time to complete a single step;
- **CPU Usage**: Average CPU usage by node;
- **Memory Usage**: Average memory consumption; and
- **Payload Size**: Size of the Node's payload.

Additionally, to get an estimate of latency, we used the publishers and subscribers within the framework to establish a bi-directional communication channel between machines in the network and orchestrated an experiment where payloads of varying sizes were sent and collected from publishers. This provided us with an estimate of the round-trip time and hence the latency compared with payload sizes within the framework.

## B. Results and Discussion

Our performance evaluation of two applications, shown in Fig. 5, focused on four benchmarks: per-node, per-worker, transmission latency, and setup times. These benchmarks reveal how CPU usage correlates with workload intensity. In the face detection pipeline, an expanded worker pool notably reduces CPU usage and step time per node, typical of parallelism. However, this trend reverses with lighter workloads, as observed in the sentiment analysis pipeline. Here, despite increased CPU usage due to communication overhead, the impact on scalability is limited. Importantly, memory usage and data sizes are consistent across nodes, reducing the risk of data loss.

Per-worker results in Fig. 6 show a marked decrease in CPU and memory usage with more workers, across all workloads.

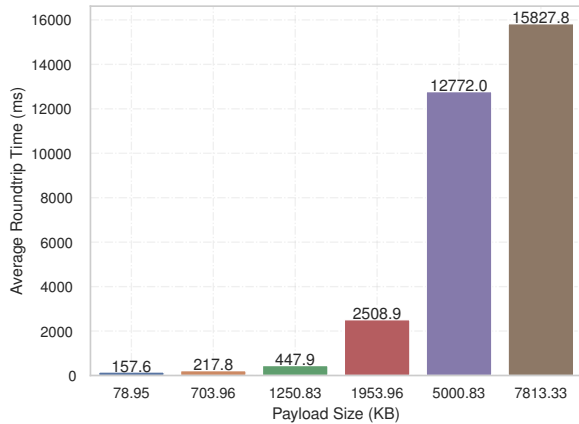


Fig. 4. Average roundtrip time vs payload size for ChimeraPy’s SUB/PUB.

TABLE I  
AVERAGE SETUP (ST), TEARDOWN (TT) AND COLLECTION (CT) TIMES

Pipeline	Configuration	ST (ms)	TT (ms)	CT (ms)
Face Detection	w1-r5-t200	8744.16	8667.78	5017.09
	w2-r5-t200	7651.24	8776.66	4072.13
	w4-r5-t200	10528.17	9402.94	7832.72
Sentiment Analysis	w1-r5-t200	7343.76	9135.10	3609.54
	w2-r5-t200	9399.47	10784.67	4810.96
	w4-r5-t200	8734.76	9243.51	4844.79

Fig. 4 indicates increased transmission times with larger payloads, highlighting a limitation in streaming large payloads. To mitigate this, ChimeraPy offers compression utilities.

For the *Minimal Setup Costs* ACR, we measured setup, collection, and teardown times, shown in Tab. I. Depending on configuration, these times range from 5–12 seconds, suitable for environments needing quick setup. Collection time may increase with longer pipeline run-time.

ChimeraPy excels in managing intensive tasks like video processing, balancing resource demands with additional workers, and maintaining data availability. The framework ensures real-time processing and transmission, with a synchronized clock for time alignment. Our tests show that ChimeraPy can be rapidly deployed for various pipelines, effectively using CPU resources and addressing ethical concerns through capabilities like face-blurring. This confirms that ChimeraPy meets our design specifications and performance criteria.

## VI. FUTURE WORK AND CONCLUSION

This paper introduced the ChimeraPy framework, designed to address gaps in MMD and GPD frameworks, focusing on minimal setup costs, time alignment, data availability, and ethical concerns. Our open-source code facilitates further MMD research and community development. Benchmarking results validate the framework’s effectiveness in data collection and analysis. ChimeraPy represents a step towards democratizing MMD streaming and processing in various research fields.

Future enhancements for ChimeraPy include advancing fault tolerance, particularly at the Worker level, and enriching the developer experience with features like hot-reloading and detailed error tracebacks.

In conclusion, ChimeraPy stands as a significant development in the realm of MMD collection and processing. Its ability to streamline development and enhance system analytics positions it to transform sectors like education and healthcare. Acknowledging the ever-evolving landscape of AI, MMD, and distributed systems, we emphasize the necessity for continual research and innovation in these areas.

## VII. ACKNOWLEDGEMENTS

This work has been supported by the National Science Foundation AI Institute Grant No. DRL-2112635. We thank the reviewers’ useful comments that helped us improve this paper.

## REFERENCES

- [1] K. Sharma and M. Giannakos, “Multimodal data capabilities for learning: What can multimodal data tell us about learning?” *British Journal of Educational Technology*, vol. 51, no. 5, pp. 1450–1484, Sep. 2020, ISSN: 0007-1013, 1467-8535. DOI: 10.1111/bjet.12993.
- [2] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, “Apache Flink™: Stream and Batch Processing in a Single Engine,” p. 11,
- [3] M. Rocklin, “Dask: Parallel Computation with Blocked algorithms and Task Scheduling,” 2015, pp. 126–132. DOI: 10.25080/Majora-7b98e3ed-013.
- [4] S. Andrist, D. Bohus, and A. Feniello, “Demonstrating a Framework for Rapid Development of Physically Situated Interactive Systems,” IEEE, Mar. 2019, pp. 668–668, ISBN: 978-1-5386-8555-6. DOI: 10.1109/HRI.2019.8673067.
- [5] *Unpack human behavior*, Mar. 2022.
- [6] J. Hassan, J. Leong, and B. Schneider, “Multimodal Data Collection Made Easy: The EZ-MMLA Toolkit: A data collection website that provides educators and researchers with easy access to multimodal data streams..” ACM, Apr. 2021, pp. 579–585, ISBN: 978-1-4503-8935-8. DOI: 10.1145/3448139.3448201.
- [7] C. Lugaresi, J. Tang, H. Nash, *et al.*, *MediaPipe: A Framework for Building Perception Pipelines*, Jun. 2019.
- [8] J. Wagner, F. Lingenfelder, T. Baur, I. Damian, F. Kistler, and E. André, “The social signal interpretation (SSI) framework: Multimodal signal processing and recognition in real-time,” ACM, Oct. 2013, pp. 831–834, ISBN: 978-1-4503-2404-5. DOI: 10.1145/2502081.2502223.
- [9] J. Kreps, N. Narkhede, and J. Rao, “Kafka: A Distributed Messaging System for Log Processing,” p. 7,



Fig. 5. Benchmark results for face detection and sentiment analysis pipelines. For face detection: (a.) CPU usage, (b.) memory usage, (c.) step time, (d.) payload size. For sentiment analysis: (e.) CPU usage, (f.) memory usage, (g.) step time, (h.) payload size, each depicting the average per node, per pipeline configuration.

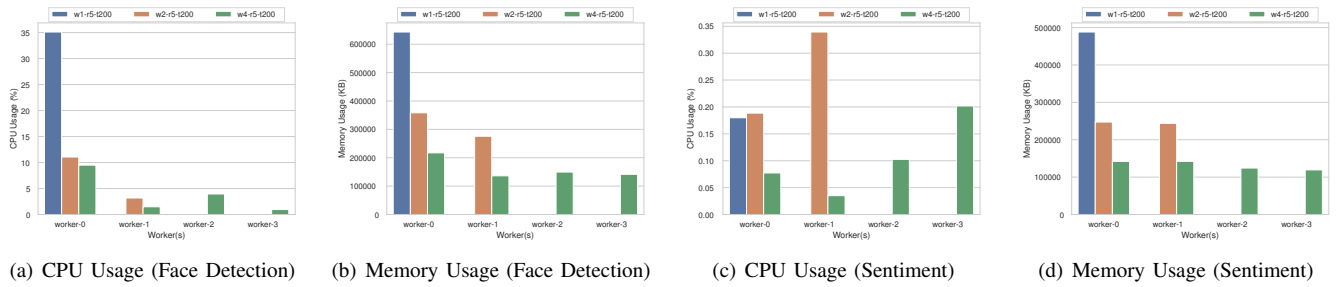


Fig. 6. Benchmark results for per worker resource consumption. Average CPU Usage per worker in face detection (a.) and sentiment analysis (c.) pipelines. Average Memory Usage per Worker in face detection (b.) and sentiment (d.) pipelines.

[10] J. Dean and S. Ghemawat, “MapReduce: Simplified data processing on large clusters,” *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008, ISSN: 0001-0782, 1557-7317. DOI: 10.1145/1327452.1327492.

[11] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, “Discretized streams: Fault-tolerant streaming computation at scale,” ACM, Nov. 2013, pp. 423–438, ISBN: 978-1-4503-2388-8. DOI: 10.1145/2517349.2522737.

[12] SZABIST, M. Hussain Iqbal, and T. Rahim Soomro, “Big Data Analysis: Apache Storm Perspective,” *International Journal of Computer Trends and Technology*, vol. 19, no. 1, pp. 9–14, Jan. 2015, ISSN: 22312803. DOI: 10.14445/22312803/IJCTT-V19P103.

[13] M. V. Bordin, D. Griebler, G. Mencagli, C. F. R. Geyer, and L. G. L. Fernandes, “DSPBench: A Suite of Benchmark Applications for Distributed Data Stream Processing Systems,” *IEEE Access*, vol. 8, pp. 222 900–222 917, 2020, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3043948.

[14] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, 2001, pp. I–I. DOI: 10.1109/CVPR.2001.990517.

[15] S. Bird and E. Loper, “NLTK: The natural language toolkit,” in *Proceedings of the ACL Interactive Poster and Demonstration Sessions*, Association for Computational Linguistics, Jul. 2004, pp. 214–217.

[16] C. Hutto and E. Gilbert, “Vader: A parsimonious rule-based model for sentiment analysis of social media text,” *Proceedings of the International AAAI Conference on Web and Social Media*, vol. 8, no. 1, pp. 216–225, May 2014. DOI: 10.1609/icwsm.v8i1.14550.