SCENE-BASED DISTRIBUTED VIDEO FRAME INTERPOLATION SYSTEM

Anisha Priyadarshini Mohanty, Venkata Bharath Reddy Reddem, & Maitreyi Sinha
Department of Computer Science
University of California, Irvine
{apmohant, vreddem, maitres}@uci.edu

ABSTRACT

Modern video applications increasingly require high-frame-rate content, yet most videos are captured at much lower frame rates, making high-quality slow-motion generation computationally expensive. Although deep learning–based interpolation models such as RIFE enable high-fidelity frame synthesis, they remain costly to apply uniformly across long form or complex videos. Moreover, existing pipelines typically run on a single machine and ignore the fact that videos have highly uneven motion across scenes, resulting in inefficiencies and resource bottlenecks. To address these limitations, we present a scene-based distributed video frame interpolation system that adaptively allocates computation according to motion complexity and parallelizes workload across multiple worker nodes. Our pipeline detects scene boundaries using histogram-based segmentation, estimates motion intensity within each scene, and determines an appropriate interpolation factor that assigns dense interpolation to high-motion segments and fewer intermediate frames to static regions. We distribute scene-level tasks across AWS worker nodes that independently perform RIFE-based interpolation using S3 as shared storage, coordinated by a lightweight master node responsible for scheduling, load balancing, and final video reconstruction. This design achieves efficient horizontal scalability, fault isolation, and significant reductions in end-to-end processing time. Experimental evaluation demonstrates that our motion-aware scheduler improves parallel efficiency compared to naive equal splitting, achieving faster runtime and better workload balance without increasing orchestration overhead. Overall, the proposed system provides a scalable, content-adaptive framework for high-quality slow-motion synthesis and offers a foundation for future distributed video processing pipelines.

1 INTRODUCTION

Modern video applications increasingly demand high frame rate (HFR) content to support smooth playback, slow-motion effects, and improved perceptual quality. However, most real-world videos are captured at standard frame rates (24 to 30 FPS) due to hardware limitations, bandwidth constraints, or storage considerations. Converting these low-frame-rate videos into high-frame-rate output requires inserting additional frames that preserve temporal consistency and visual fidelity. Deep learning based video frame interpolation models such as Huang et al. (2022) (RIFE) have demonstrated significant improvements in the synthesizing of intermediate frames by estimating the motion dynamics between consecutive frames. Despite their accuracy, these models remain computationally expensive and memory intensive, especially when processing long videos or generating high levels of slow motion.

One important thing to note that the videos exhibit highly non uniform motion characteristics across scenes. Some segments remain nearly static, while others contain rapid or complex motion. Interpolating the same number of frames across all scenes is inefficient and can lead to unnecessary computation or temporal artifacts. Moreover, existing pipelines generally operate as monolithic processes on a single machine. This not only limits scalability, but also creates performance bottlenecks for large scale or real-time applications such as sports analysis, film post-production, and multi-camera systems.

To address these limitations, we design a distributed, motion-aware video frame interpolation pipeline specifically for slow-motion video generation. Our system decomposes the source video into semantically meaningful scenes through histogram based scene detection, then estimates motion intensity within each scene using frame difference based metrics. Using these motion scores, we adaptively determine the number of intermediate frames to generate per scene, allocating more interpolation steps to be assigned to high-motion segments and fewer to be assigned to low-motion ones. This creates a content-aware slow-motion effect that preserves perceptual coherence while minimizing over processing.

To scale the computational workload, we distribute scene-level interpolation tasks across multiple worker nodes. Each worker pulls scenes and metadata from a shared S3 bucket, performs RIFE-based interpolation locally, and uploads processed segments back to the bucket. The master node orchestrates the entire pipeline, including job generation, load balancing, and final video reconstruction. This architecture allows our system to leverage parallelism efficiently while isolating heavy GPU computation to worker nodes, thus overcoming memory bottlenecks and enabling horizontal scalability.

Overall, our work demonstrates a fully automated distributed framework for high quality slow-motion video synthesis that adapts to content complexity and scales gracefully with computational resources. Beyond slow motion, the proposed system can serve as a foundation for future research in distributed video processing, real-time media enhancement, cloud-based video frame interpolation services, and adaptive video transformation pipelines.

2 Related Works

Video frame interpolation (VFI) has been an active research area for more than two decades, with applications ranging from frame-rate upconversion and slow-motion generation to video editing and streaming. Classical interpolation techniques relied on optical flow estimation to describe the pixel level motion between consecutive frames. Early works such as Meyer et al. (2015) and Baker et al. (2011) used variational formulations to approximate motion vectors and synthesize intermediate frames. However, these methods struggle with occlusions, nonlinear motion and large motion magnitudes, often producing artifacts.

The emergence of deep learning significantly improved interpolation quality. Methods like Jiang et al. (2018) introduced end-to-end deep networks that jointly predict optical flow and visibility maps. Later models such as Bao et al. (2019), leveraged depth information to reason about occlusions more explicitly. Among recent approaches, Huang et al. (2022) (RIFE) is particularly notable for achieving high quality interpolation with real time performance on modern GPUs. RIFE's efficiency and robustness make it suitable for large scale or distributed processing pipelines.

Although the above methods focus primarily on interpolation quality, relatively few works (Dean & Ghemawat (2008), Huang et al. (2017)) explore the system challenge of processing long videos efficiently. Large scale video processing frameworks such as PySpark Video Processing, ffmpeg batch pipelines and distributed render farms support parallel video operations, but they assume homogeneous compute workloads. They lack mechanisms to adapt splitting or scheduling based on the videos content dynamics such as motion intensity or scene complexity.

Recent studies (Hassanien et al. (2017)) in adaptive video encoding, particularly those using scene based bitrate adaptation, demonstrate that using scene level information improves resource utilization and perceptual quality. However, these insights have not been fully leveraged in the context of video interpolation. None of the prior work specifically addresses distributed motion adaptive frame interpolation, where different video segments may require different amounts of computation depending on their motion complexity.

Our work builds on these ideas by introducing a scene-aware, motion-based load balancing mechanism for parallel video interpolation. Instead of splitting the video into equal frame chunks, we detect scene boundaries and measure their motion intensity, allocating more computational resources (i.e., more interpolation steps) to scenes with higher motion. This strategy not only improves system efficiency, but also results in more natural slow motion outputs, as scenes with high motion benefit from higher interpolation density.

3 Overview of Distributed Video Frame Interpolation System

The Scene-Based Distributed Video Frame Interpolation System is designed to make the generation of high-quality, slow-motion videos much more efficient by taking advantage of distributed computing on multiple nodes. Traditional video interpolation systems process the entire video sequentially, which becomes computationally expensive for long or high-resolution videos. In addition, frame interpolation itself is computationally intensive; applying it uniformly across scenes, regardless of whether they contain fast or slow motion, greatly wastes unnecessary compute time and uses resources inefficiently.

To overcome these, our proposed system segments the input video into distinct scenes, analyzes each scene's motion intensity, and adaptively determines how many intermediate frames should be generated. High-motion segments receive more interpolation density, and low-motion or static segments receive proportionally fewer frames, striking a balance between computational efficiency and visual smoothness. This is the main point in the design to reduce redundant computation and improve output quality. Once the video has been decomposed and analyzed, these scenes are dispatched to multiple AWS worker nodes, which then perform interpolation using the RIFE model-a state-of-the-art approach based on IFNet for real-time intermediate optical flow estimation. The distributed pipeline takes advantage of S3-backed storage for transferring scene fragments and aggregating results. By parallelizing this process and adapting the workload to scene complexity, the system achieves much faster processing while maintaining high visual fidelity.

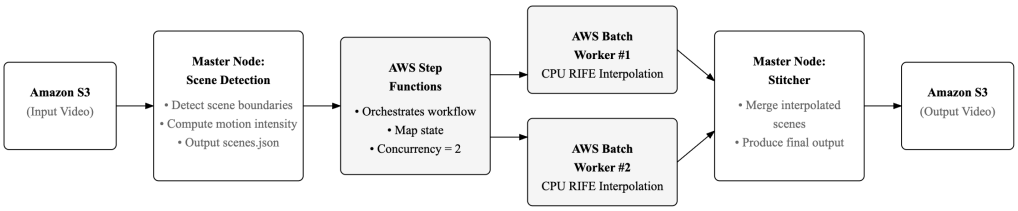

Figure 1: System Architecture

4 System Architecture

4.1 High-Level Architecture

The system follows a master-worker distributed design, where the master node orchestrates all preprocessing and coordination tasks while worker nodes perform the compute heavy frame interpolation. The architecture contains five main stages:

1. Scene Detection
2. Motion Intensity Estimation
3. Load Balancing and Scene Assignment
4. Distributed Frame Interpolation on Worker Nodes
5. Scene Aggregation and Output Generation

This modular decomposition allows each component to scale independently and ensures that the system can efficiently handle large videos or datasets.

4.2 Master Node Responsibilities

4.2.1 Scene Detection

The master initiates processing by running a histogram-based scene boundary detection algorithm. This identifies timestamps where significant visual changes occur, effectively dividing the video into coherent segments. By isolating scenes before interpolation, the system avoids unnecessary computation in low-motion or static regions and enables adaptive processing.

4.2.2 Motion Analysis and Complexity Estimation

For every scene detected, the master calculates a motion intensity, which is used as a measure of the expected interpolation cost. The higher the motion complexity, the more intermediate frames need to be generated, while the simpler scenes require fewer. This step is indispensable for load balancing that is done in an informed manner and also it stops the work from being unevenly distributed across worker nodes.

4.2.3 Dynamic Load Balancing

In effect, the master uses the motion scores to calculate the computation load for each scene and then assigns to the worker nodes through a greedy scheduling strategy. It thus ensures that workers get a roughly equal share of the total workload, thus preventing stragglers and reducing the overall processing time.

4.2.4 S3 Coordination

The master sends the scene fragments to Amazon S3, from where the worker nodes download the tasks assigned to them. After all workers have completed their tasks, the master gets the interpolated results from S3 and combines them into a single slow-motion video.

4.3 Worker Node Responsibilities

Each worker node operates as an independent compute unit, performing:

1. Fetching assigned scenes from S3
2. Running the RIFE frame interpolation model
3. Uploading the interpolated scene back to S3

RIFE provides high-quality intermediate flows using IFNet and is well-suited for distributed environments due to its efficiency and model compactness. The worker nodes require no global state, allowing the system to scale horizontally by simply adding more nodes.

4.4 AWS Distributed Pipeline

The system is deployed on AWS as a fully distributed pipeline:

1. Master Node (EC2): Preprocessing, scheduling, and orchestration
2. S3 Buckets: Object storage for video scenes, intermediate results, and final outputs
3. Worker EC2 Nodes: Parallel RIFE inference
4. Networking & IAM: Allow secure data transfer between nodes and storage

This further ensures reliability, scalability, and fault tolerance in such a way that true parallelism happens due to the decoupling of scene-level tasks, with no cross-worker communication required during the processing. The key strengths of the architecture are adaptive interpolation, which allocates computation resources exactly at the right place; extensive parallelism, significantly reducing the total processing time compared to a single-machine workflow. The scene-based modular design enhances fault tolerance for graceful recoveries from worker failures and also supports future

enhancements like GPU-aware scheduling. Seamless S3 integration reduces communication latencies and bottlenecks common in transfer approaches through centralized locations, while effective load balancing prevents idling workers and maximizes throughput, especially in videos with uneven motion distribution.

4.5 Workflow

The workflow of our distributed video interpolation system is shown in Figure 1. The system operates as a linear, cloud-based pipeline consisting of scene detection, distributed RIFE processing, and final stitching.

4.5.1 Input Upload (S3)

The process begins when the input video is uploaded to an S3 bucket. This acts as the shared storage location for both the master node and worker nodes throughout the pipeline.

4.5.2 Scene Detection (Master Node)

The master node downloads the input video and performs histogram-based scene detection. For each detected scene, the master computes motion intensity scores and stores scene metadata which includes scene boundaries, motion scores, and interpolation factor into a JSON file.

4.5.3 Workflow Orchestration (AWS Step Functions)

AWS Step Functions coordinates the distributed processing pipeline. It launches multiple parallel branches, one per batch of scenes, with a configured concurrency of two in our current setup. Step Functions determines which worker receives which batch of scenes based on the selected load-balancing strategy: equal split or motion-aware.

4.5.4 Distributed RIFE Interpolation (CPU workers)

Each batch is processed by a worker machine running the CPU version of RIFE. The workers download the assigned scene clips from S3, perform frame interpolation according to the batch's interpolation factor, and upload the interpolated scene outputs back to S3. As the workers run independently, this stage exploits parallelism to reduce total runtime.

4.5.5 Stitching and Final Encoding (Master Node)

After all batches finish, the master node downloads the interpolated scenes from S3 and merges them into a single continuous video. This step ensures correct scene ordering, consistent frame rate, and proper temporal alignment. The final stitched video is then uploaded back to S3 as the output artifact.

5 Evaluation

We evaluate our distributed interpolation pipeline along three dimensions:

1. End-to-end performance
2. Worker-level load balance
3. Master-node orchestration overhead

Our goal is to understand whether motion-aware scene scheduling improves parallel efficiency compared to naive equal splitting while maintaining low coordination overhead.

5.1 End-to-End Pipeline Performance

Figure 2 summarizes the total runtime across the three execution strategies. Running the entire pipeline on a single worker requires 5297 seconds. Dividing the video into two equal frame ranges

reduces this to 3138 seconds. The 1.69× speed up proves the benefit of parallelizing interpolation workloads.

Our scene-aware scheduler further reduces total runtime to 2878 seconds, achieving a 1.84× speedup over the single-worker baseline and a 9.0% improvement over equal splitting. This improvement arises because scenes differ substantially in motion intensity, and therefore differ in the number of intermediate frames each segment requires. Equal splitting fails to account for this variation, resulting in a slower worker that delays final completion. By contrast, our scheduler assigns heavier scenes to one worker and lighter scenes to the other. Overall, our scheduler reduces the tail latency that dominates overall runtime.

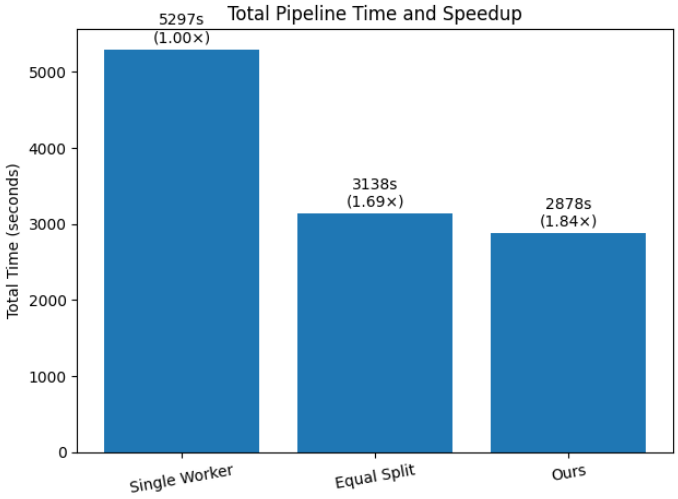

Figure 2: Total pipeline runtime for single-worker execution, equal-split parallelism, and our scene-aware scheduling.

5.2 Worker Load Balance Analysis

To further assess the effect of scheduling on parallel efficiency, we measure the execution time of each worker (Figure 3). Under equal splitting, Worker 1 runs for 2971 seconds while Worker 2 finishes in 2030 seconds which results in a 31.7% imbalance. This imbalance directly translates into worker idle time and explains the suboptimal overall runtime.

With scene-aware scheduling, Worker 1 and Worker 2 complete in 2714 and 2419 seconds respectively: a reduced imbalance of 10.9%. Although both workers still process different amounts of work, the difference is substantially smaller. This aligns with the observation that motion intensity, rather than frame count, is a better predictor of computational load in interpolation pipelines.

Figure 4 provides another view of the balance of workers by reporting the proportion of total work completed by each worker. Equal splitting assigns 59.4% of total work to Worker 1, while our strategy distributes the workload more evenly at 52.9% vs. 47.1%. This visualization reinforces that naive frame count partitioning leads to systemic imbalance, while motion-aware scheduling better approximates true cost.

5.3 Master Node Overhead

Finally, we examine master-side orchestration cost, which includes scene detection, metadata generation, task assignment, uploading scene clips, and final stitching. As shown in Figure 5, both strategies incur similar overheads of roughly 50 seconds. Scene detection takes around 4–5 seconds, assignment around 20–22 seconds, and uploading around 24 seconds, with final concatenation contributing less than one second.

Importantly, these operations account for less than 2% of the total execution time, indicating that the improved performance of our method does not come at the cost of additional coordination or

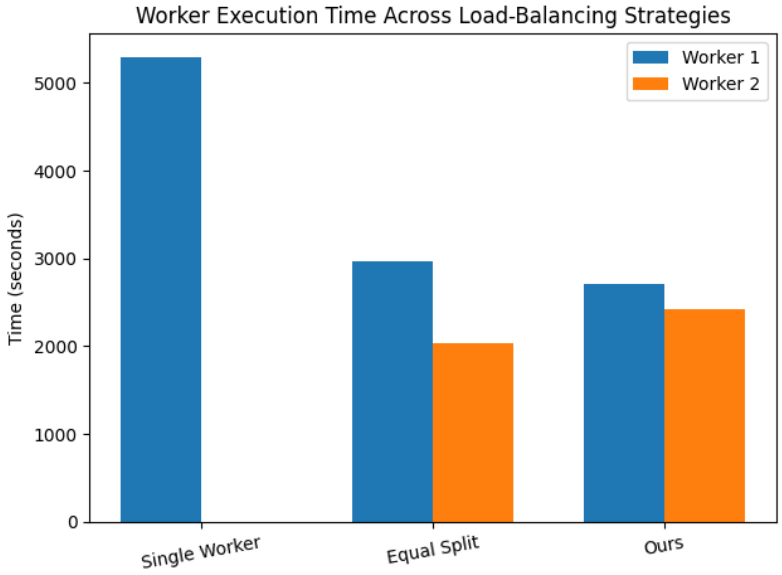

Figure 3: Worker execution time for single-worker, equal-split, and our scene-aware allocation strategies.

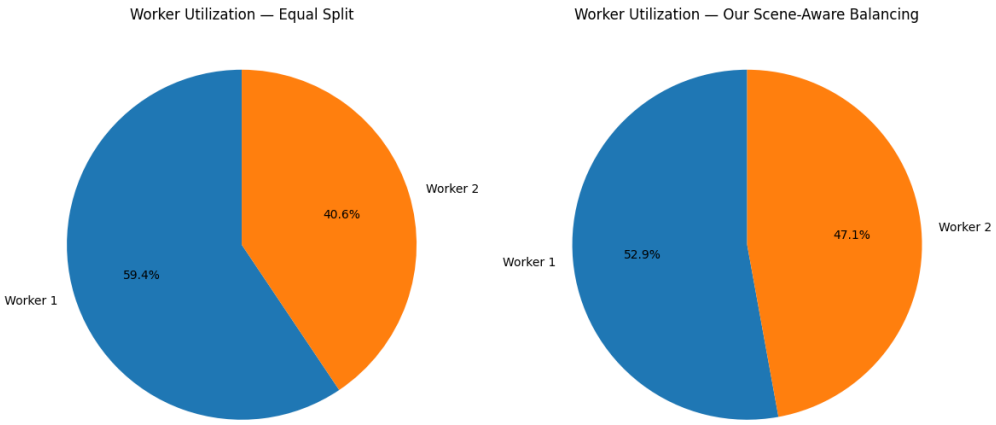

Figure 4: Worker utilization under equal splitting (left) and our motion-aware scheduling (right).

preprocessing overhead. This suggests that the pipeline bottleneck lies almost entirely in worker-side interpolation and that computational load, not orchestration cost, is the primary factor limiting throughput.

6 Discussion

Our results show that motion-aware scheduling yields meaningful improvements in distributed video interpolation by addressing the core challenge of load imbalance across workers. Equal frame splitting naively assumes uniform computational cost, but interpolation workloads vary significantly with motion intensity: high-motion scenes require more intermediate frames and therefore more inference steps, causing one worker to run substantially longer than the other. By assigning scenes proportionally to their estimated motion complexity, our method reduces this imbalance and improves end-to-end performance without introducing additional orchestration overhead. The master node remains lightweight, contributing less than 2% of total runtime, which indicates that the

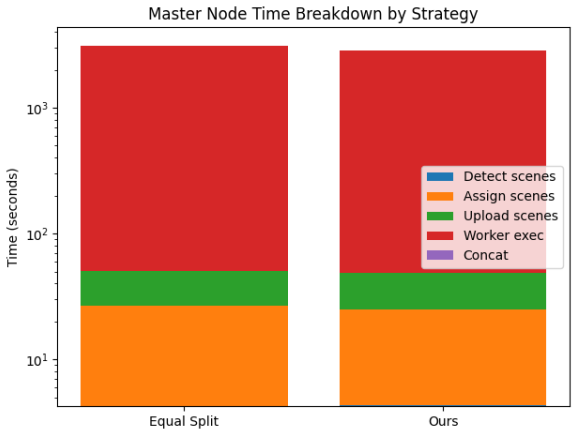

Figure 5: Master node overhead comparison between equal split and our scene-aware method (excluding worker execution).

pipeline bottleneck lies almost entirely on the worker side. This suggests the architecture will scale effectively as more workers are added, though fine-grained task partitioning or dynamic scheduling may be needed for larger clusters. At the same time, the approach has limitations: the motion score is based on simple frame differences and may not capture complex motion types, S3 transfers introduce latency that becomes more pronounced for tiny scenes, and CPU-only RIFE inference constrains the achievable speedups. Nonetheless, the findings demonstrate that even lightweight motion analysis can significantly improve distributed interpolation pipelines, especially for videos with diverse motion profiles, and they highlight clear opportunities for future extensions such as GPU acceleration, dynamic runtime-aware scheduling, and richer motion complexity estimation.

6.1 Challenges

Building a distributed scene based video slow motion system introduced several technical and systems level challenges. First, the pipeline required accurate scene segmentation to avoid splitting at incorrect boundaries, as even small errors produced misaligned interpolation and inconsistent playback. Second, load balancing was difficult because each scene had different motion characteristics, leading to different interpolation costs. Unlike simple frame count based partitioning, our system had to estimate computational load using motion intensity. Third, executing the RIFE model on multiple workers introduced out of memory failures, especially on scenes with high interpolation factors. This required careful management of model loading, batching and cleanup. Additionally, transferring scenes and results between the master and workers, whether locally or using S3, created I/O bottlenecks and synchronization challenges especially for larger videos. Finally, merging outputs with variable frame rates to generate smooth slow motion video required precise ffmpeg handling to avoid dropped frames or temporal artifacts. Despite these challenges, designing and testing each component separately, allowed us to successfully develop a scalable framework for Distributed Video slow motion.

6.2 Future Work

Explore several directions to improve both performance and robustness. First, the current load balancing strategy based on frame count and estimated interpolated frames can be enhanced with dynamic, runtime adaptive scheduling that continuously redistributes scenes based on actual processing rates and worker slowdowns. Integrating a lightweight monitoring service would enable timely detection of stragglers and more informed rescheduling decisions. Another promising direction is to incorporate fault tolerance mechanisms such as checkpointing per-scene progress, allowing the system to recover from worker failures without recomputing entire segments. Additionally, support for heterogeneous clusters can be added by automatically profiling workers and scaling their assigned load according to CPU/GPU capabilities. Finally, implementing distributed storage and parallelized I/O pipelines would address data transfer bottlenecks, especially when working with

large videos stored on remote services like S3. Combining these enhancements would generalize the system to broader workloads and enable real-time large scale video processing.

7 CONCLUSION

In this project, we designed and implemented a distributed system capable of efficiently processing tasks across multiple worker nodes. By decomposing the workload, distributing tasks based on estimated computation cost and coordinating execution through a central controller, we demonstrated that parallelization significantly improves overall throughput compared to a single node baseline. Our evaluation shows that the system achieves good scalability, fault isolation, and improved completion time for large workloads. The project also highlighted common challenges in distributed systems, such as load imbalance, worker heterogeneity, synchronization overhead and consistency issues. Overall, our implementation provides a functional and extensible foundation for exploring real world distributed computing principles, including task scheduling, coordination, and performance measurement.

REFERENCES

Simon Baker, Daniel Scharstein, James P Lewis, Stefan Roth, Michael J Black, and Richard Szeliski. A database and evaluation methodology for optical flow. *International journal of computer vision*, 92(1):1–31, 2011.

Wenbo Bao, Wei-Sheng Lai, Chao Ma, Xiaoyun Zhang, Zhiyong Gao, and Ming-Hsuan Yang. Depth-aware video frame interpolation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 3703–3712, 2019.

Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

Ahmed Hassanien, Mohamed Elgharib, Ahmed Selim, Sung-Ho Bae, Mohamed Hefeeda, and Wojciech Matusik. Large-scale, fast and accurate shot boundary detection through spatio-temporal convolutional neural networks. *arXiv preprint arXiv:1705.03281*, 2017.

Qi Huang, Petchean Ang, Peter Knowles, Tomasz Nykiel, Iaroslav Tverdokhlib, Amit Yajurvedi, Paul Dapolito IV, Xifan Yan, Maxim Bykov, Chuen Liang, et al. Sve: Distributed video processing at facebook scale. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pp. 87–103, 2017.

Zhewei Huang, Tianyuan Zhang, Wen Heng, Boxin Shi, and Shuchang Zhou. Real-time intermediate flow estimation for video frame interpolation. In *European Conference on Computer Vision*, pp. 624–642. Springer, 2022.

Huaizu Jiang, Deqing Sun, Varun Jampani, Ming-Hsuan Yang, Erik Learned-Miller, and Jan Kautz. Super slomo: High quality estimation of multiple intermediate frames for video interpolation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 9000–9008, 2018.

Simone Meyer, Oliver Wang, Henning Zimmer, Max Grosse, and Alexander Sorkine-Hornung. Phase-based frame interpolation for video. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1410–1418, 2015.