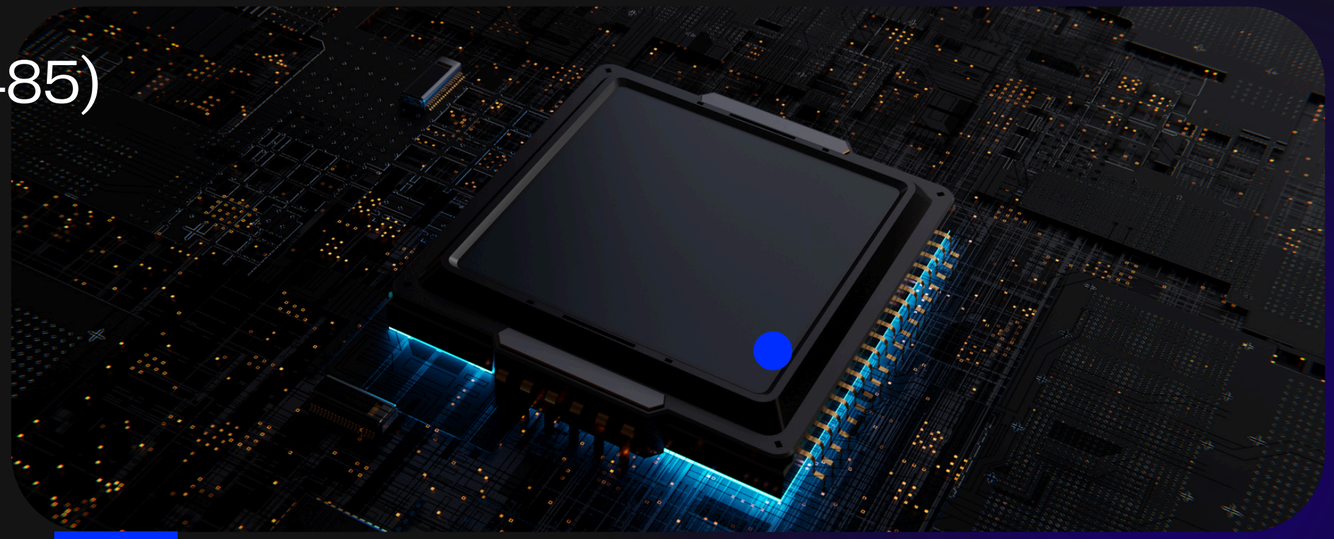
Northern Lights Detection

Group8:
GAITE Santana Ysabelle Perez (57865485)
Chuang Kwok Tai (57838735)
Chiu Chi Yin (57848243)
Lam Ching Hoi(57308299)



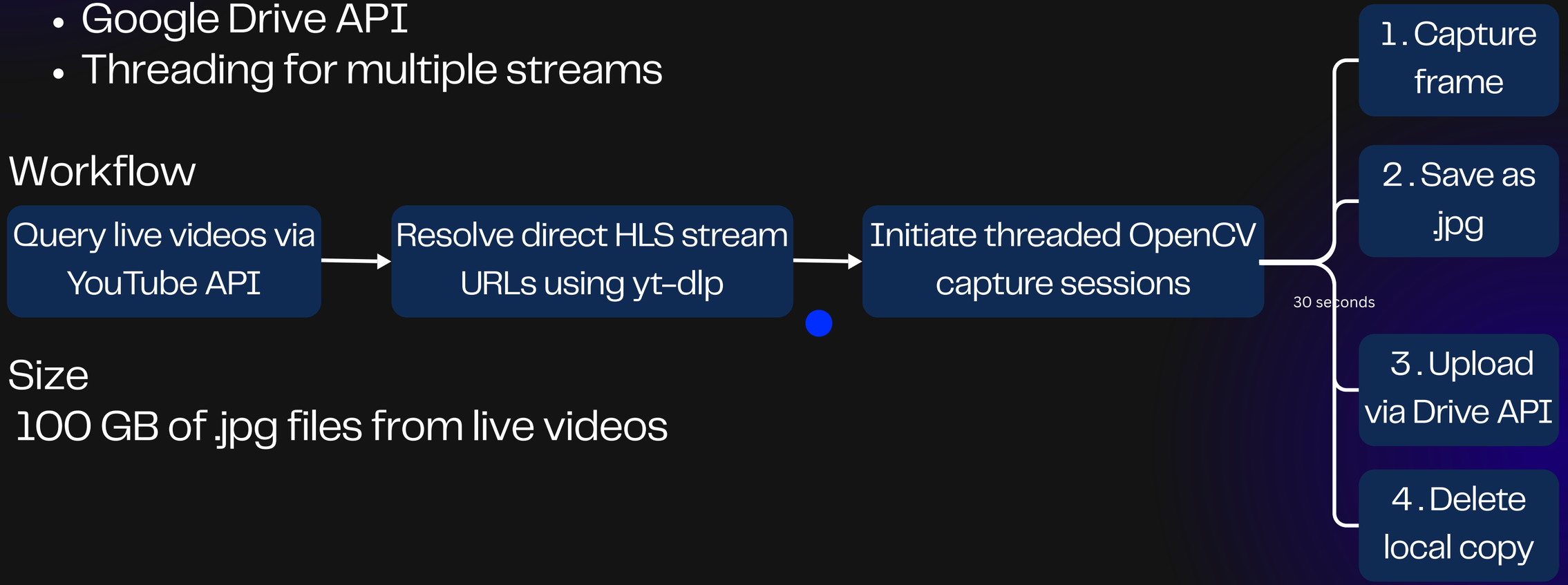
20 Nov 2025

Data Collection

Tools & APIs

- Python, OpenCV, yt-dlp, YouTube Data API v3
- Google Drive API
- Threading for multiple streams

Workflow

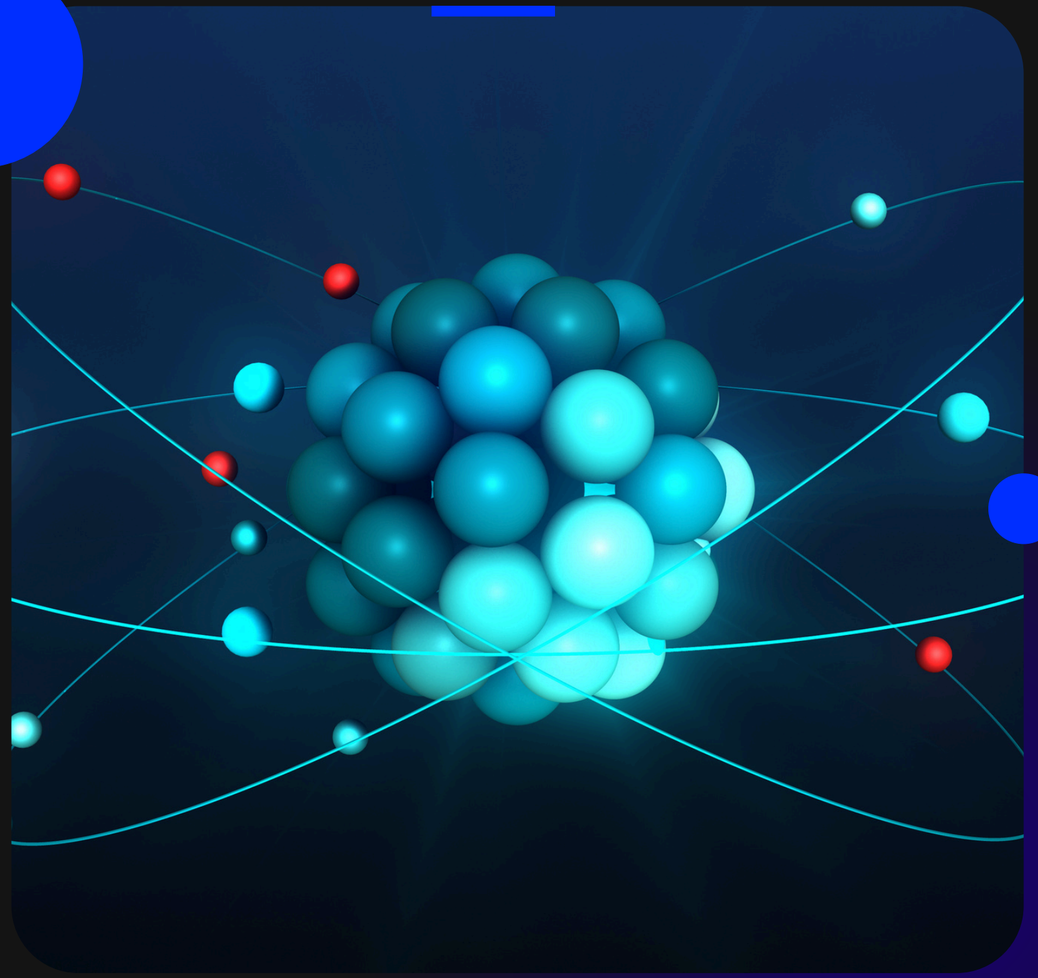


Size

100 GB of .jpg files from live videos

Cluster Setup on GCP Dataproc

Environment: Hadoop + Spark 3.5 on Dataproc
Nodes: 1 master, 10 workers (n4-standard-2)
Specs: 2 vCPU, 8 GB RAM per node
Image: Debian 12, Python 3.11.8
Tools enabled: Jupyter, Zookeeper,Hive
Total resources: 82.21 GB memory, 26 vCores



FFmpeg Threading & Video Processing

FFmpeg 6.1.1, open source, C++/C

h264 codec

Thread model: auto, cores x 1.5

27 March, 2024

Data for Expirement

54 vides (16GB)

2277frames (8GB)

We first use a small amount of data to do the experiment

Using the whole dataset during model training

FFmpeg Parallel Decoding Scalability Experiment on macOS vs HTCC1(Cityu CS lab)

purpose: to understand how many executors/cores we need

Item	MacBook Pro (M4)	HTCC Server Node
CPU Architecture	Apple Silicon (ARM-based)	Intel Xeon Platinum 8280 (x86-64)
CPU Cores / Threads	10 cores / 10 threads	28 physical cores / 56 threads
Memory	16 GB unified memory (UMA)	10 GB RAM (distributed memory model)
Local / Scratch Disk	Fast internal NVMe SSD	**2 GB buffer disk per slot**

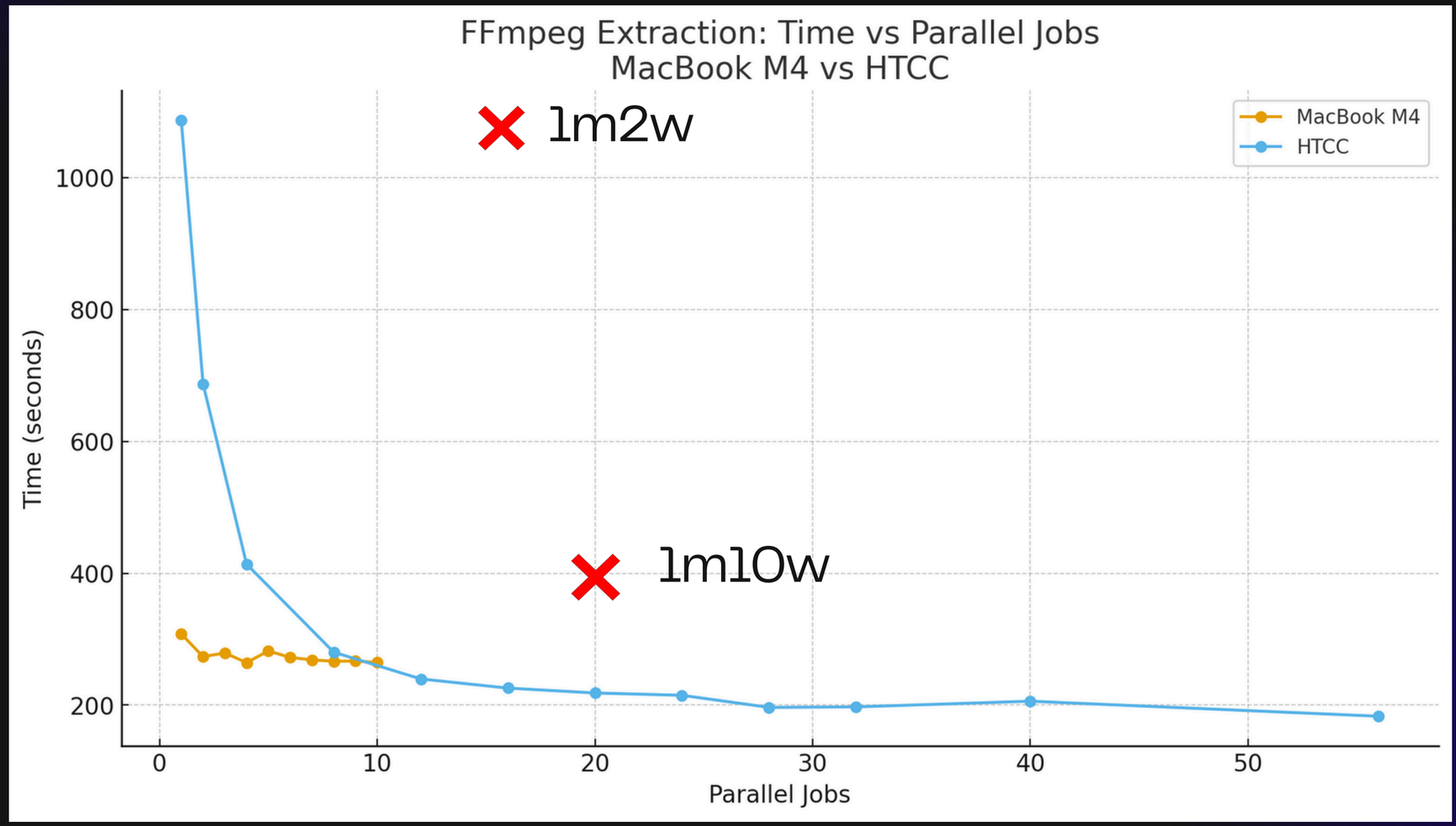
FFmpeg Parallel Decoding Scalability Experiment on macOS vs HTCC1(Cityu CS lab)

Both PC runs the same Bash script:

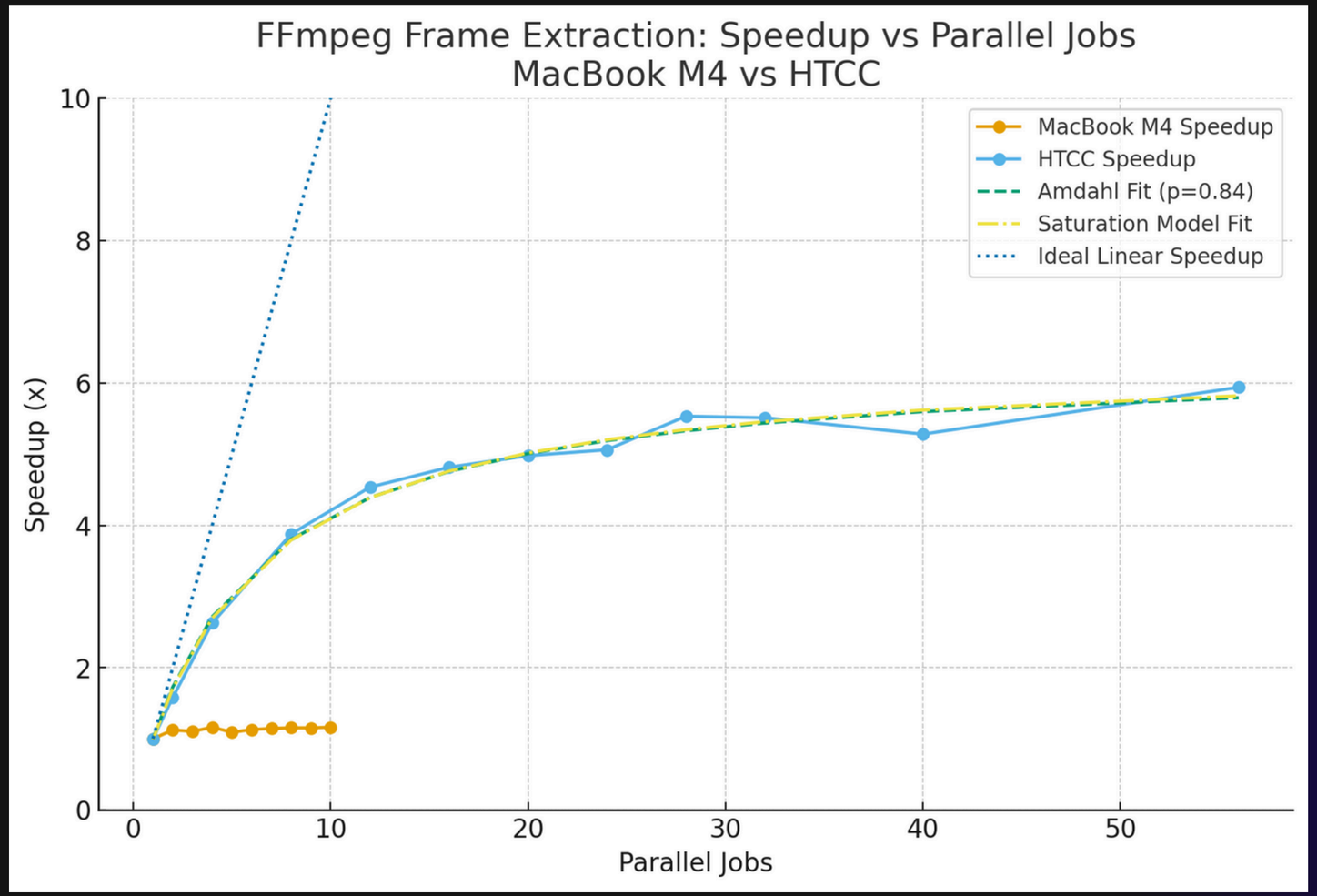
Bash background jobs (&)

- Process-level parallelism

FFmpeg Parallel Decoding Scalability Experiment on macOS vs HTCC1(Cityu CS lab)



FFmpeg Parallel Decoding Scalability Experiment on macOS vs HTCC



FFmpeg Parallel Decoding Scalability Experiment on macOS vs HTCC

Why does MacBook run so fast

1. Memory
2. Mac has hardware video decoding engines, HTCC CPU uses software to decode

Feature	LPDDR5X (MacBook M4)	DDR4-2993
Technology Generation	LPDDR5X (Low-Power, an evolution of DDR5)	DDR4 (Older Generation)
Raw Data Rate	**7500 MT/s** (Base M4)	**2993 MT/s**
System Bandwidth	**120 GB/s** (Base M4)	**~47.9 GB/s** (Dual-Channel)

Hadoop Cluster Parallelism Impact Experiment

Purpose: to proof scale out > scale up

Item	Cluster A	Cluster B
Master Nodes	1	1
Worker Nodes	10	2
Machine Type	n4-standard-2	n4-standard-8
vCPU per Node	2 vCPU	8 vCPU
Memory per Node	8 GB	32 GB
Total Worker vCPUs	10 × 2 = **20 vCPU**	2 × 8 = **16 vCPU**
Total Worker Memory	10 × 8 = **80 GB**	2 × 32 = **64 GB**

Hadoop Cluster Parallelism Impact Experiment

Distributing Video Paths Across Partitions

a. Creates an **RDD** from video file paths(**sc.parallelize(), mapPartitions()**)
b. numSlices=54 splits the list into 54 partitions
c. Each partition is processed by a separate task

Each worker:

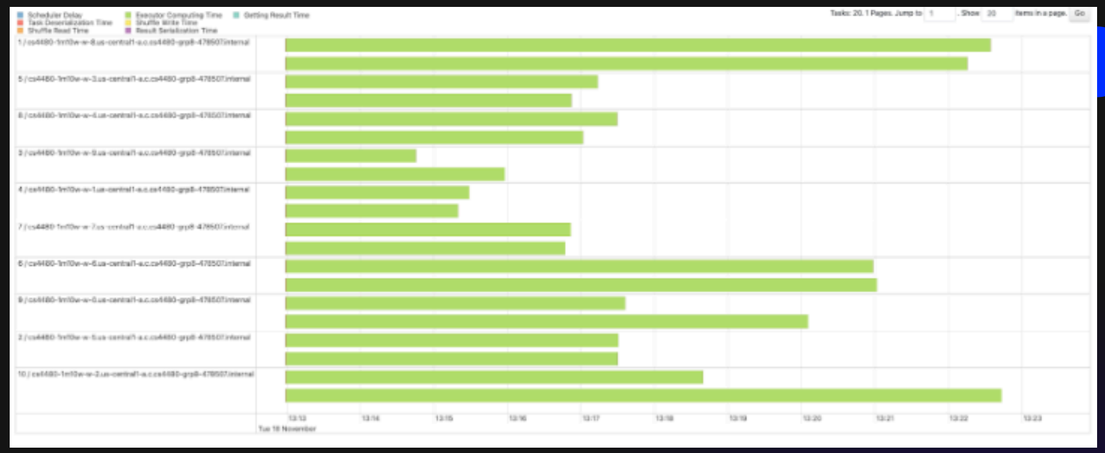
a. **Downloads video from HDFS to local temp**
b. Runs FFmpeg to extract frames
c. **Uploads frames back to HDFS**
d. Cleans up local files

Hadoop Cluster Parallelism Impact Experiment

54 partitions(1 video per task):

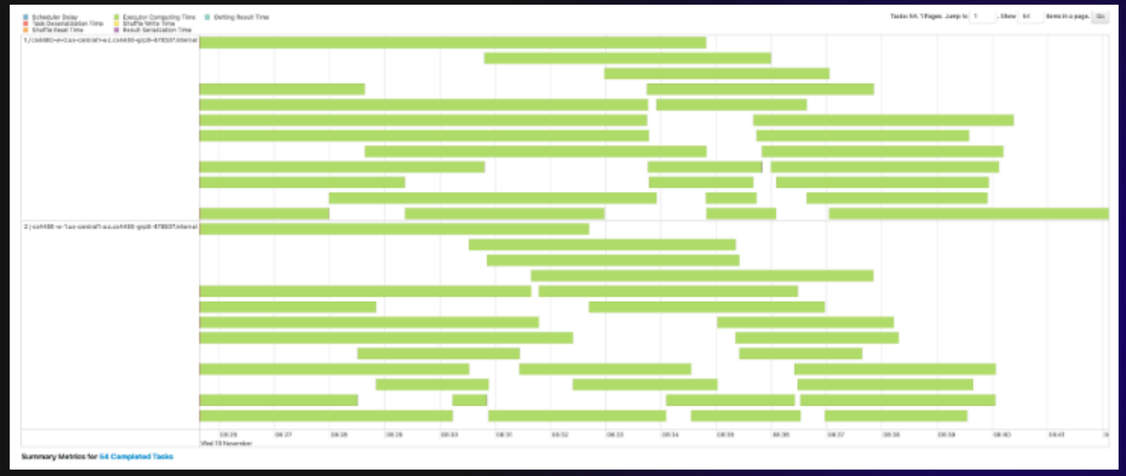
1m10w: ~8mins

1m2w: ~17mins



More workers

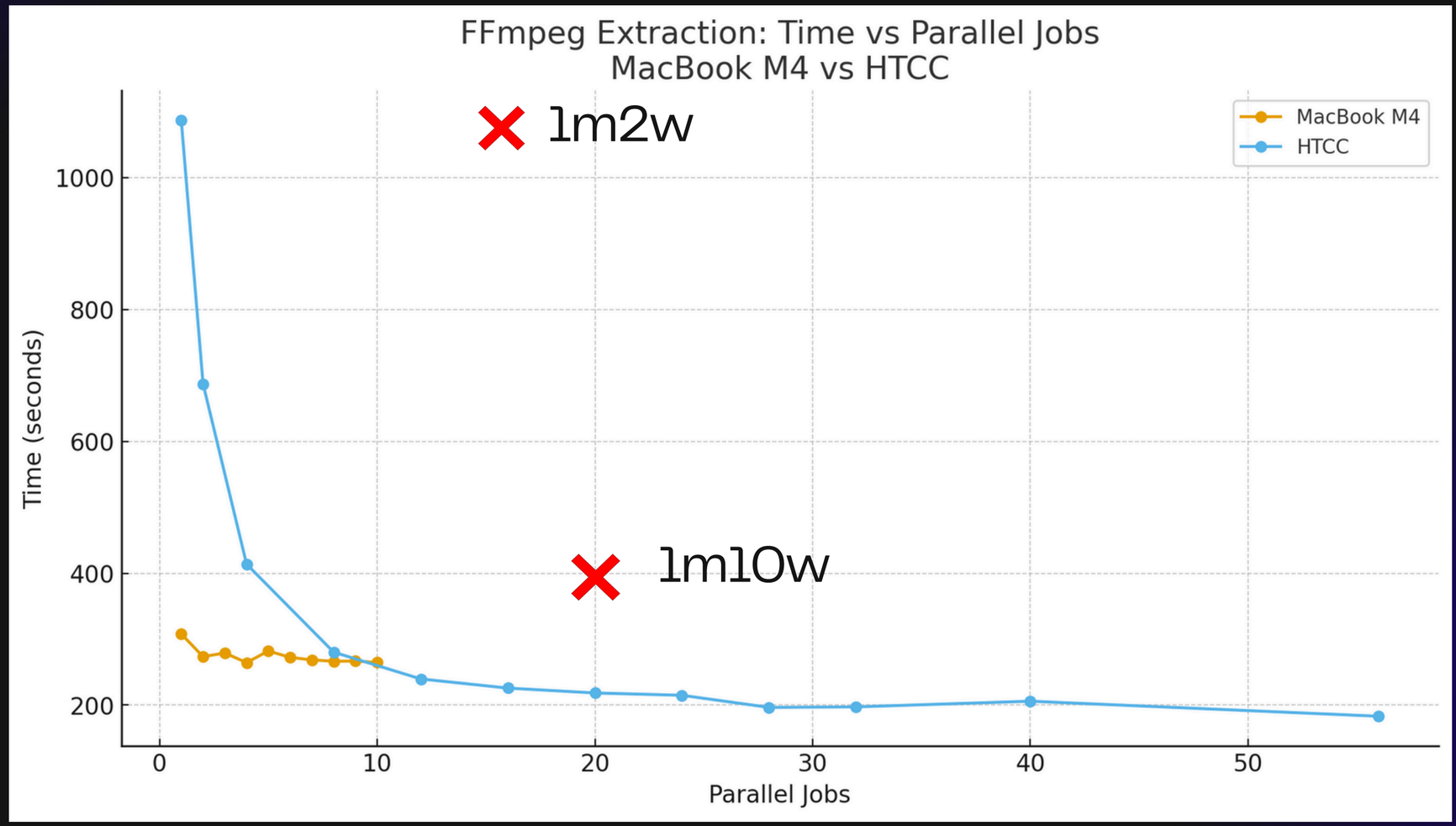
→ Higher total IO throughput → Faster

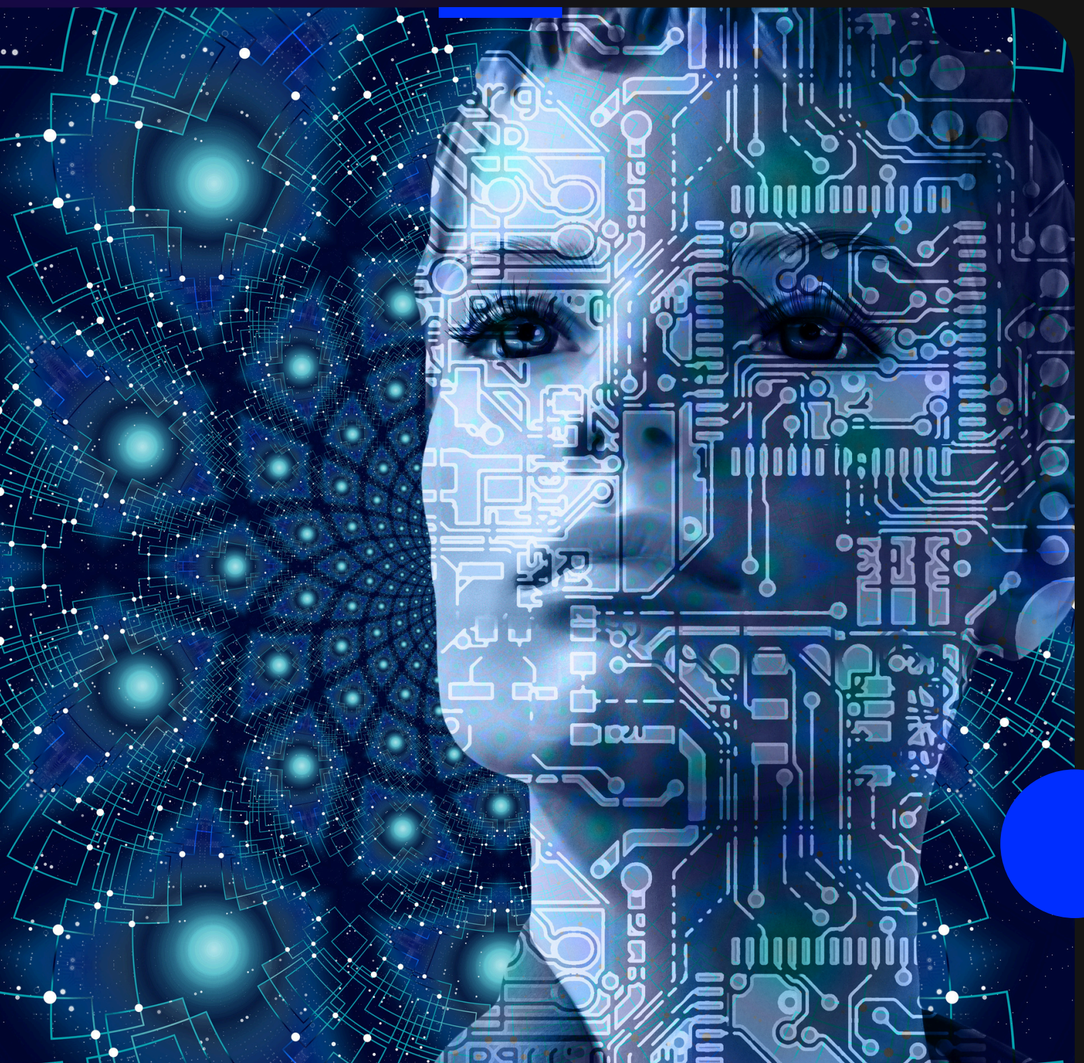


Keys

- Even CPU–heavy tasks, in **Big Data** (HDFS) need high throughput

FFmpeg Parallel Decoding Scalability Experiment on macOS vs HTCC





Ongoing Works

Now we understand the trade-off

1. Label the frame (multithreading, Hive)
2. Train the model (HDFS Pyspark)

Label the frames

We are prompting **Gemini 2.0-fash-lite** to label the frame
Using **Hive** to store the data

Ideal approach in **Big Data: HDFS → RDD → Gemini**

- Limitation: DataProc have network
- Need gcloudcli SSH tunnel forwarding to connect the master node
    - High network latency
- Rate Limit, network

Practial approach: **Local** → Gemini → HDFS(thought googlecli)

- IO heavy → multithreading (python 3.14 free threaded mode with async)

Model traning

Like the Hadoop cluster expirement

Pyspark → RDD(train/val/test) → Model traning

Q&A

Thank you