Video Data processing with beam

Name: Adesh Abhang

The Challenge — Scaling Video data processing

The Storage & I/O Bottleneck: Massive video files cause traditional "download-then-process" architectures to crash, overwhelming worker nodes' local disks and network bandwidth.

Temporal Data Complexity: Video models require continuous, overlapping frame sequences (sliding windows), making manual data orchestration across distributed workers highly inefficient.

The "Black Box" Inference Trap: Distributed deep learning models can silently fail mathematically (e.g., outputting flattened tensors) without throwing pipeline errors, quietly destroying downstream training data.

Full Pipeline at a Glance

6 stages, GPU-accelerated, auto-scaling

STAGE 1 → Video Segmentation

Split each video using splittable do function

STAGE 2 → Frame Extraction

FPS at which frames will be extracted, resize frame as per the requriment

STAGE 3 → Sliding Windows

We specify the window size and the window stride at this stage

STAGE 4 → Beam Run Inference(GPU)

feature vector per clip via NVIDIA T4

STAGE 5 → Label Alignment

CoGroupByKey joins features + labels per window

STAGE 6 → TFRecord Output

SequenceExample per video as output

How one massive video file becomes 10 parallel GPU tasks

Phase 1

Probe & Scope

- Video metadata probed
- Total work range set
- e.g. Frames 0 ⟶ 60,000

`RestrictionTracker`

Phase 2

Static Splitting

- Work range subdivided
- Chunks of ~6,000 frames
- RestrictionProvider plans work

`RestrictionProvider`

Phase 3

Dynamic Claiming

- try_claim() per frame
- Slow workers get stolen
- 10 GPU arms run in parallel

`try_claim()`

60,000	10×	6,000	~0ms
Total Frames	Parallel Workers	Frames per Chunk	Coordination Overhead

Extraction Engine

- **OpenCV:** Simple, NumPy-based, sequential reading
- **FFmpeg:** Optimized, supports random access

Assign Event Time

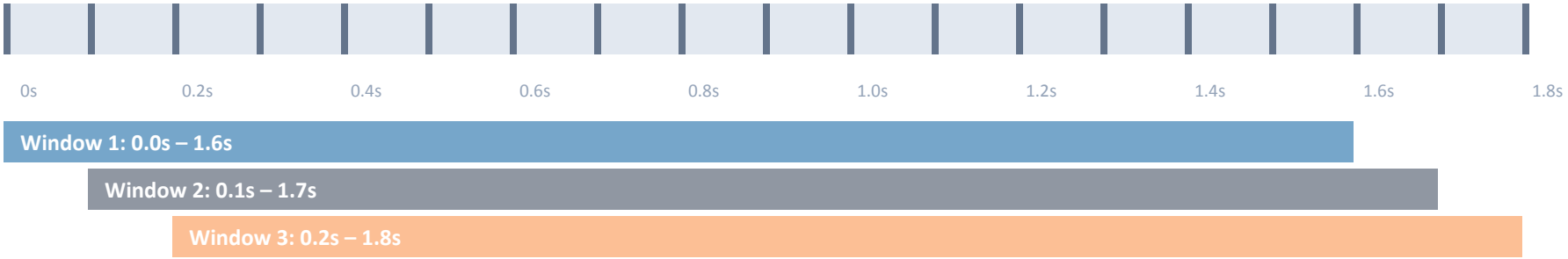
- Timestamp = Frame Index / FPS
- Align to stride (e.g., 0.1s)
- Use: TimestampedValue(frame, timestamp)

Why Timestamps Matter

- Enables Beam **SlidingWindows (event-time based)**
- Ensures sync across different FPS streams
- Critical for correct training data generation

Windowing: Processing Data in Time Slices

Sliding Window — 1.6s wide, moves every 0.1s



Each window = 16 frames ⟶ 1 video clip tensor for X3D model

Why Sliding?

Captures action continuity. A kick that starts at 0.05s shouldn't be missed just because it doesn't align to a fixed boundary.

Code

SlidingWindows(size=1.6, period=0.1)
Each frame appears in 16 windows — same coverage as 16×FixedWindows in one pass.

ML Inference at Scale with RunInference


```
model_handler = KeyedModelHandler(
    PytorchModelHandlerTensor(
        state_dict_path=
model_fuse_path,
        model_class=
X3DWrapper, device='cuda:0'
    )
)

clip_tensors | 'RunInference' >>
RunInference(model_handler)
```

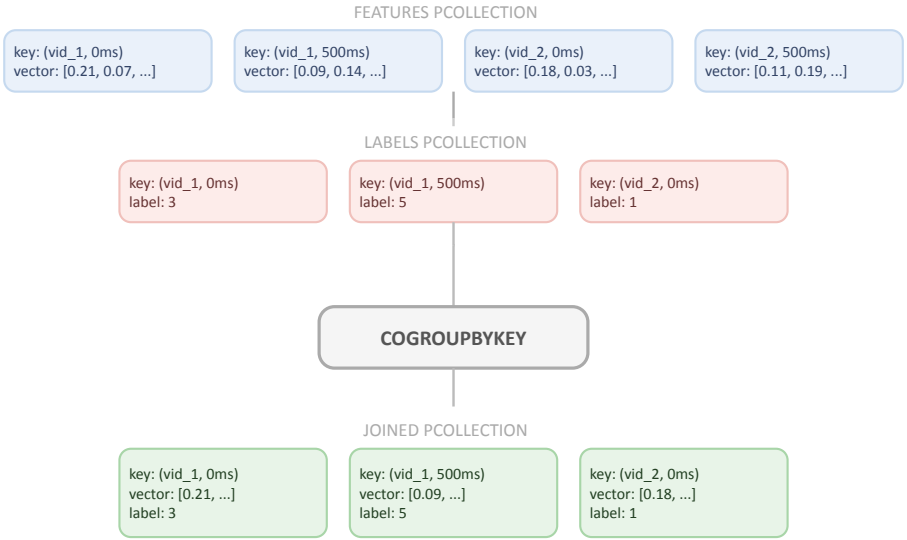
Batching	Keyed	GPU
Beam auto-batches tensors for GPU efficiency	KeyedModelHandler preserves (video_id, clip_id) through inference	Model loads on cuda:0, inference runs with torch.no_grad()

BEAM COLLEGE

CoGroupByKey joins features and labels

RunInference produces two separate PCollections keyed by **(video_id, window_start_id)**: one with extracted feature vectors, one with action labels.

CoGroupByKey joins them by shared key into a unified record — aligning every feature vector with its corresponding label before writing to TFRecord.



features: (clip_key, 2048-dim vector)

labels: (clip_key, action_label_int)

clip_key → {features: [...], labels: [int]}

Both values co-located by shared key — ready for serialisation.

frame_indexes, labels, x3d_features

Written as TF SequenceExample to TFRecord.

Pipeline Results · video_29_1_video_left

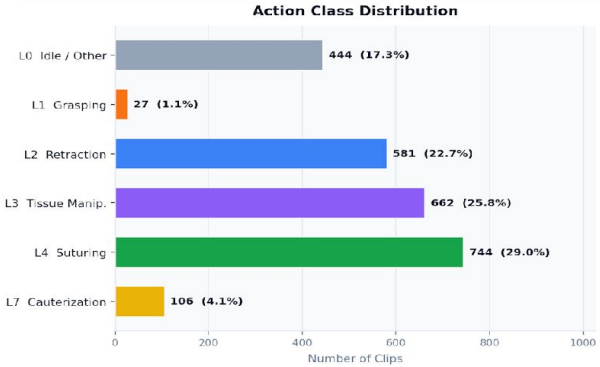
TFRecord · 2,564 clips · 15,378 frames · 6 action classes · frame step = 6

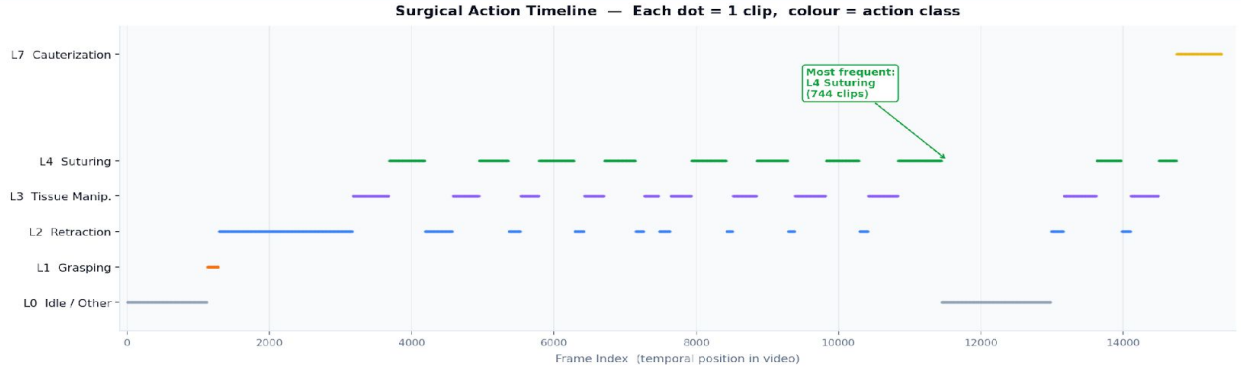
BEAM COLLEGE · 2026

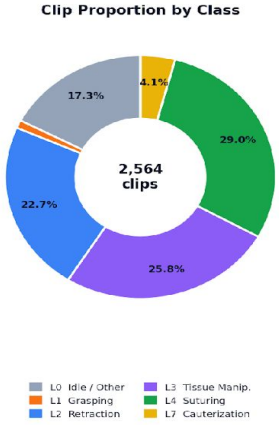
Pipeline Results · video_29_1_video_left

TFRecord · 2,564 clips · 15,378 frames · 6 action classes detected · frame step = 6

BEAM COLLEGE · 2026







2,564	15,378	6 / 8	Frame ÷ 6	2048-dim	SequenceEx.
Total Clips	**Frames Processed**	**Action Classes**	**Sampling Rate**	**Feature Vector**	**Output Format**
Sliding-window clips extracted from video	Raw frames decoded by pipeline	Classes present in this video (labels 5 & 6 absent)	1 clip per 6 frames (10 FPS → clip every 0.6s)	X3D-M embedding per clip (identity output)	TF SequenceExample per video in .tfrecord

Pipeline Metrics at a Glance

BEAM COLLEGE · 2026

10 FPS
sampled from 60 FPS source

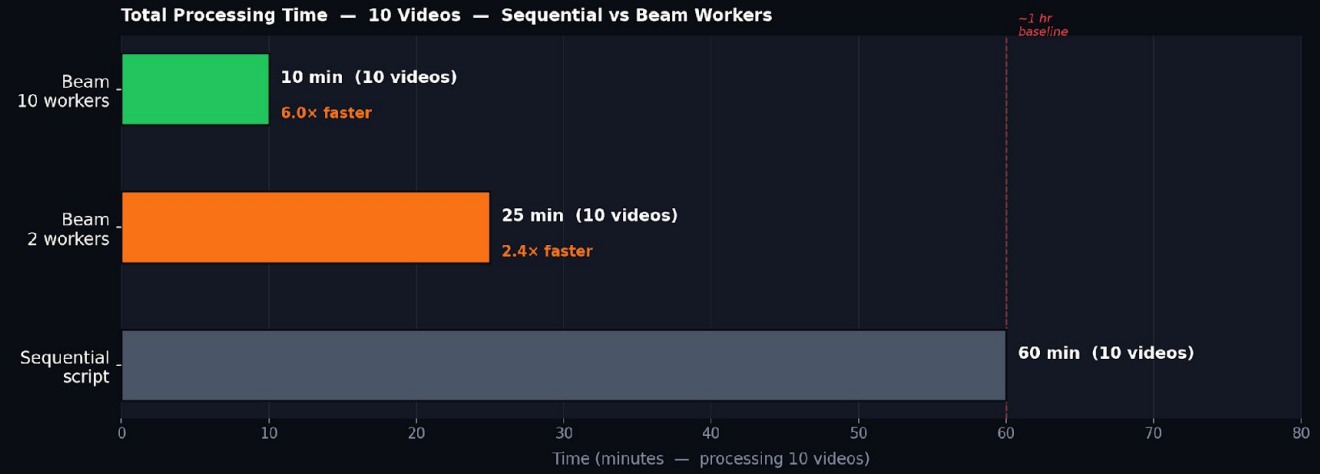
16 frames
per sliding-window clip

1.6 s
window duration

2048-dim
feature vector per clip

2 → 10
Dataflow autoscaling workers

T4 GPU
NVIDIA Tesla on n1-standard-8



Key Takeaways

- **6×** faster with 10 workers
- **2.4×** faster with just 2 workers
- **10** videos processed in parallel
- **50 min** saved vs sequential script

Thank you!

Email: adeshabhang44@gmail.com
Linkedin: www.linkedin.com/in/adesh-abhang-9b5aa7241