

Project structure: React+Vite+typescript

I have Inbuilt HTML react video loader in my project which loads videos frame by frame manner what I want to do is I want to annotate keypoints using d3.js(<https://www.npmjs.com/package/d3?activeTab=readme>) library on top of the each frame of video, it should plot keypoints and save keypoints in csv format

To implement video frame annotation with D3.js keypoint plotting and CSV export in a React+Vite+TypeScript project, here's a structured approach:

Video Annotation Setup

1. Install Dependencies:

```
npm install d3 @types/d3
```

2. Video Frame Component:

```
import { useEffect, useRef, useState } from 'react';
import * as d3 from 'd3';

interface KeyPoint { x: number; y: number; label: string }

const VideoAnnotator = () => {
  const videoRef = useRef<HTMLVideoElement>(null);
  const svgRef = useRef<SVGSVGElement>(null);
  const [keyPoints, setKeyPoints] = useState<KeyPoint[]>([]);

  // Initialize D3
  useEffect(() => {
    const svg = d3.select(svgRef.current);

    // Drag behavior for keypoints
    const drag = d3.drag<SVGCircleElement, KeyPoint>()
      .on('drag', (event, d) => {
        d3.select(event.target)
          .attr('cx', d.x = event.x)
          .attr('cy', d.y = event.y);
        setKeyPoints([...keyPoints]);
      });

    // Draw keypoints
    svg.selectAll('circle')
      .data(keyPoints)
      .join('circle')
      .attr('cx', d => d.x)
      .attr('cy', d => d.y)
      .attr('r', 5);
  });
};
```

```

        .style('fill', 'red')
        .call(drag);
    }, [keyPoints]);

    // Add new keypoint on video click
    const handleVideoClick = (e: React.MouseEvent<HTMLDivElement>) => {
        const rect = e.currentTarget.getBoundingClientRect();
        setKeyPoints([...keyPoints, {
            x: e.clientX - rect.left,
            y: e.clientY - rect.top,
            label: `Point ${keyPoints.length + 1}`
        }]);
    };

    return (
        <div onClick={handleVideoClick}>
            <video ref={videoRef} controls width={800} height={450}>
                <source src="/sample.mp4" type="video/mp4" />
            </video>
            <svg ref={svgRef} width={800} height={450} style={{ position: 'absolute', top: 0 }}>
            </div>
        );
    };

```

CSV Export Implementation

Choose either a library-based or native approach:

Option 1: Using react-csv [\[1\]](#) [\[2\]](#)

```

import { CSVLink } from 'react-csv';

const ExportButton = ({ data }: { data: KeyPoint[] }) => (
    <CSVLink
        data={data}
        headers={[
            { label: "X Position", key: "x" },
            { label: "Y Position", key: "y" },
            { label: "Label", key: "label" }
        ]}
        filename="annotations.csv"
    >
        Export to CSV
    </CSVLink>
);

```

Option 2: Native Implementation [\[3\]](#) [\[4\]](#)

```

const exportCSV = (data: KeyPoint[]) => {
    const csvContent = [
        ['x', 'y', 'label'],
        ...data.map(p => [p.x, p.y, p.label])
    ].map(row => row.join(',')).join('\n');

```

```

const blob = new Blob([csvContent], { type: 'text/csv' });
const url = URL.createObjectURL(blob);
const link = document.createElement('a');
link.href = url;
link.download = 'annotations.csv';
document.body.appendChild(link);
link.click();
document.body.removeChild(link);
};

```

Project Structure Recommendations

```

src/
├── components/
│   ├── VideoAnnotator.tsx
│   └── CSVExporter.tsx
├── utils/
│   └── csvUtils.ts
├── types/
│   └── keypoints.ts
└── App.tsx

```

Key considerations:

1. Use `useEffect` for D3 DOM manipulation [\[5\]](#) [\[6\]](#)
2. Maintain annotation state separately from video frames
3. Implement frame synchronization using `requestAnimationFrame`
4. Add validation for coordinate boundaries relative to video dimensions
5. Consider adding undo/redo functionality for annotations

This implementation allows precise keypoint annotation on video frames with real-time coordinate tracking and flexible CSV export options. The D3.js integration provides smooth drag interactions while maintaining React's component structure.

✱✱

I have

1. HTML5 <video>
2. react-player
3. plyr
4. vidstack react
5. ffmpeg
6. fabric
7. konva

8. video.js

9. opencv.js libraries which would be most suitable for keypoint annotation task with *d3.js* frame by frame video handling suggest me to do the task

To implement keypoint annotation with frame-by-frame video handling using your listed libraries, here's the optimal combination considering D3.js integration and React compatibility:

Recommended Stack

1. **Video Handling:** Vidstack React + HTML5 <video>
2. **Annotation Layer:** D3.js + Konva
3. **Frame Processing:** FFmpeg (preprocessing) + OpenCV.js (optional analysis)

Implementation Strategy

1. Video Frame Control

```
// Using Vidstack React for frame-accurate control
import { MediaPlayer } from '@vidstack/react';

<MediaPlayer
  src="/video.mp4"
  onTimeUpdate={({ currentTime }) => handleFrameUpdate(currentTime)}
/>
```

Vidstack provides better React integration than raw HTML5 video while maintaining frame access^[7] ^[8].

2. SVG Overlay with D3.js

```
// SVG overlay component
const AnnotationCanvas = ({ videoRef }) => {
  const svgRef = useRef<SVGSVGElement>(null);

  useEffect(() => {
    const svg = d3.select(svgRef.current);
    // D3 annotation logic here
  }, [videoRef.currentTime]);

  return <svg ref={svgRef} style={{ position: 'absolute' }} />;
};
```

Use D3 for scalable vector annotations with mouse interaction support^[8] ^[9].

3. Frame Synchronization

```
// Frame handler using requestVideoFrameCallback
const handleFrameUpdate = (timestamp) => {
  videoRef.current.requestVideoFrameCallback((now, metadata) => {
    setCurrentFrame(metadata.mediaTime);
  });
};
```

```
});  
};
```

This API provides precise frame timing data [\[7\]](#) [\[10\]](#).

Library Suitability Analysis

Library	Best For	Why Recommended
Vidstack React	Frame-accurate video control	Modern React API with TS support
D3.js	Scalable vector annotations	Precise coordinate mapping
FFmpeg	Keyframe extraction/preprocessing	Batch frame processing
OpenCV.js	Computer vision analysis	Optional posture estimation

Avoid These Combinations

1. **React-Player/Plyr**: Abstracted APIs limit frame-level access
2. **Fabric/Konva**: Canvas-based solutions conflict with D3's SVG approach
3. **Video.js**: jQuery dependency creates bundle bloat

Key Integration Points

1. Use `transform: translate()` to align SVG overlay with video element
2. Implement frame navigation controls using Vidstack's API
3. For performance, use OpenCV.js WebAssembly backend for real-time analysis:

```
// Optional OpenCV integration  
cv.imshow('outputCanvas', frame);  
const keypoints = cv.findContours(frame);
```

This stack balances React compatibility, annotation precision, and video control while maintaining TypeScript type safety. For CSV export, use the D3-dsv module paired with your existing implementation [\[8\]](#) [\[11\]](#).



If we use ffmpeg, will we have to open source our code due to it's licensing terms?

You do **not** have to open source your entire code just because you use FFmpeg, but it depends on how you use it and which parts of FFmpeg you include:

- **FFmpeg is licensed under LGPL 2.1+ by default.** If you use only the LGPL parts (and not the optional GPL components), you can use FFmpeg in a closed-source commercial application, provided you comply with LGPL requirements. This usually means:

- If you dynamically link FFmpeg (use it as a shared library or call it as an external process), you do **not** have to open source your application, but you must allow users to replace the FFmpeg library with their own version^{[12] [13] [14]}.
- If you statically link FFmpeg, you must either provide object files so users can relink with a different FFmpeg version, or open source your application under LGPL^[14].
- **If you use any GPL components of FFmpeg** (such as certain codecs or optional modules), your entire application must be released under the GPL, which does require you to open source your code under GPL terms^{[12] [14] [15]}.
- **If you use FFmpeg as a separate executable** (e.g., calling ffmpeg via command line from your app), you generally avoid the need to open source your code, as long as you do not include GPL-only features and comply with LGPL terms^{[13] [14]}.

Summary:

You can use FFmpeg in a closed-source, commercial product without open sourcing your code if:

- You use only LGPL parts (not GPL components)
- You dynamically link or use it as a separate process
- You comply with LGPL requirements (allowing users to replace FFmpeg)

If you use GPL components, you must open source your code under the GPL^{[12] [14] [15]}. Always review the specific FFmpeg build and its included modules to ensure compliance.

✱✱

what can we use instead of FFmpeg, because we want it closed source for commercial purpose.

If you want to avoid FFmpeg due to its LGPL/GPL licensing and keep your application fully closed source for commercial use, here are the most suitable alternatives:

Commercial-Friendly Alternatives to FFmpeg

1. GStreamer

- GStreamer is a powerful multimedia framework that supports video/audio processing, playback, and transcoding.
- It is licensed under LGPL, similar to FFmpeg, but is often easier to use in a modular way that avoids GPL components.
- GStreamer is widely used in commercial products and has robust bindings for multiple languages^[16].
- Make sure to use only LGPL plugins and avoid GPL-licensed modules.

2. HandBrake

- HandBrake is a popular video transcoder, but it is primarily a desktop application and not a developer library^[17].
- It is open source and uses GPL, so it is not suitable if you want to avoid open source obligations.

3. AWS Elemental MediaTailor and Similar Cloud Services

- AWS Elemental MediaTailor and other cloud-based video processing services (e.g., Google Cloud Video Intelligence, Azure Media Services) are fully commercial and do not impose open source requirements on your code^[17].
- You pay for usage, but your application can remain closed source.

4. VideoLAN (VLC)

- VLC is open source and uses GPL, which is not suitable for closed source commercial applications^[17].

5. Commercial SDKs (e.g., MainConcept, Intel Media SDK)

- MainConcept and Intel Media SDK offer robust commercial video processing SDKs with closed source, royalty-based licensing.
- These are widely used in the industry for professional/commercial video processing needs.

6. OpenCV

- OpenCV is licensed under Apache 2.0, which is permissive and suitable for closed source commercial use^[16].
- It is excellent for video frame extraction, processing, and computer vision tasks, though it is not a full replacement for FFmpeg's encoding/decoding capabilities.

7. Rhozet (Harmonic Carbon Coder)

- Rhozet is a commercial, enterprise-grade video transcoding solution. It is proprietary and used by large companies for scalable video processing^[18].
- It is expensive but offers professional support and no open source licensing requirements.

Summary Table

Alternative	License	Suitable for Closed Source?	Notes
GStreamer (LGPL modules)	LGPL	Yes (if LGPL modules only)	Avoid GPL plugins; modular architecture
AWS Elemental, Azure, etc.	Commercial	Yes	Pay-per-use, no code disclosure
MainConcept, Intel SDK	Commercial	Yes	Paid, professional SDKs
OpenCV	Apache 2.0	Yes	Great for frame processing
Rhozet (Carbon Coder)	Commercial	Yes	Enterprise-grade, expensive
HandBrake, VLC	GPL	No	Not suitable for closed source

Recommendation

- **GStreamer** (with LGPL-only plugins) or a **commercial SDK** (like MainConcept or Intel Media SDK) are the best alternatives for local processing.
- **Cloud services** (AWS Elemental, Azure Media, Google Cloud) are best if you prefer not to deal with licensing at all and are comfortable with cloud-based processing.
- **OpenCV** is excellent for frame extraction and analysis, but not for full video transcoding.

Always review the specific license of the modules/plugins you use, and consult legal counsel for commercial deployment. [\[16\]](#) [\[17\]](#) [\[18\]](#)

✱✱

lets go with openCV

OpenCV is well-suited for your requirements: it is licensed under Apache 2.0, which is business-friendly and allows you to keep your application closed source. Here's how you can use OpenCV for frame-by-frame video handling and annotation:

How to Use OpenCV for Frame-by-Frame Video Handling

- **Read and Process Frames:** OpenCV treats a video as a sequence of images (frames). You can use `cv2.VideoCapture()` to open a video file and then loop through each frame with `.read()` [\[19\]](#) [\[20\]](#) [\[21\]](#) [\[22\]](#).
- **Access Video Properties:** Use `.get()` with properties like `cv2.CAP_PROP_FRAME_COUNT`, `cv2.CAP_PROP_FPS`, `cv2.CAP_PROP_FRAME_WIDTH`, and `cv2.CAP_PROP_FRAME_HEIGHT` to retrieve metadata [\[20\]](#) [\[23\]](#) [\[22\]](#).
- **Display/Annotate Frames:** Each frame is a NumPy array, which you can process or annotate using OpenCV drawing functions or by exporting the frame to another library (like D3.js in a web context) [\[22\]](#).
- **Save Processed Video:** Use `cv2.VideoWriter()` to save annotated or processed frames back to a video file [\[19\]](#) [\[22\]](#).

Example: Basic Frame-by-Frame Processing

```
import cv2

cap = cv2.VideoCapture('input.mp4')
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break
    # Process or annotate the frame here
    cv2.imshow('Frame', frame)
    if cv2.waitKey(25) & 0xFF == ord('q'):
        break
```



```
cap.release()
cv2.destroyAllWindows()
```

Advantages of Using OpenCV

- **No open source obligation:** Apache 2.0 license allows closed source commercial use.
- **Direct frame access:** Easily process, annotate, and export frames.
- **Flexible:** Works with both local files and webcam streams^{[20] [23] [22]}.
- **Export capability:** Can save processed frames as images or videos^{[19] [22]}.

Integration with D3.js

If you want to annotate frames with D3.js (typically in a web browser), you can:

- Use OpenCV (Python or OpenCV.js) to extract frames and keypoint data.
- Send frames and/or keypoint data to the frontend for visualization and annotation with D3.js.
- Save/export keypoints as CSV from your React/D3.js frontend.

Summary:

OpenCV is a robust, commercial-friendly choice for frame-by-frame video processing and annotation, without any requirement to open source your code^[22].

✱

1. <https://www.npmjs.com/package/react-csv>
2. <https://stackoverflow.com/questions/66894037/react-export-to-csv-no-headers-in-the-csv-file/66894305>
3. <https://dev.to/graciesharma/implementing-csv-data-export-in-react-without-external-libraries-3030>
4. <https://www.clearpeople.com/blog/export-data-to-csv-with-typescript-without-format-issues>
5. <https://blog.logrocket.com/using-d3-js-v6-with-react/>
6. <https://medium.com/@imranfarooq0306/using-react-with-d3-js-around-react-c6fe042d6364>
7. <https://stackoverflow.com/questions/49750014/add-annotation-text-to-particular-frame-in-video-using-javascript-html5>
8. <https://www.griddynamics.com/blog/using-d3-js-with-react-js>
9. <https://observablehq.com/@hydrosquall/d3-annotation-with-d3-line-chart>
10. <https://superuser.com/questions/1830532/how-to-extract-keyframe-timestamps-from-video-using-ffmpeg>
11. <https://www.v7labs.com/blog/keypoint-annotation-guide>
12. <https://www.ffmpeg.org/legal.html>
13. <https://opensource.stackexchange.com/questions/10850/using-ffmpeg-only-for-decoding-in-closed-source-commercial-application>
14. <https://github.com/arthenica/ffmpeg-kit/wiki/Using-FFmpegKit-in-Commercial-Applications>
15. <http://www.swains.plus.com/ffmpeg-test/legal.html>

16. <https://stackshare.io/ffmpeg/alternatives>
17. <https://www.g2.com/products/ffmpeg/competitors/alternatives>
18. <https://serverfault.com/questions/185675/is-there-anything-better-than-ffmpeg>
19. https://docs.opencv.org/4.x/dd/d43/tutorial_py_video_display.html
20. <https://opencv.org/blog/reading-and-writing-videos-using-opencv/>
21. <https://app.studyraid.com/en/read/14915/513867/processing-video-frames-sequentially>
22. <https://www.33rdsquare.com/working-with-images-and-videos-using-opencv/>
23. <https://dlab.berkeley.edu/news/processing-videos-python-opencv>