Quick checks

1. Make sure the live URL is HTTPS
Browsers block camera and mic on HTTP. Replit Deployments are HTTPS, so use the deployed link, not the old preview URL.

2. Use a user gesture
Autoplay is restricted. Start playback in a click event and set the video to muted and playsinline.

```
<video id="cam" autoplay muted
playsinline style="width:100%"></video>
<button id="start">Start Camera</button>
<script>

document.getElementById('start').addEven
```

```
tListener('click', async () => {
    try {
      const stream = await
navigator.mediaDevices.getUserMedia({
video: true, audio: false });
      const v =
document.getElementById('cam');
      v.srcObject = stream;
      await v.play();
    } catch (e) {
      console.error(e);
      alert('Camera blocked or not available.
Check site permissions.');
    }
  });
</script>
```

3. Add iOS and Android friendly attrs
iOS Safari needs muted and playsinline for inline playback. Keep both, even if you also show controls.

4. Check permissions on the live domain
Your browser stores camera permissions per origin. After you deploy, click the lock icon in the address bar and allow Camera for the deployed domain.

5. Use correct file paths for static videos
If you are playing a file, put it in a public or static folder and use a root–relative path. Example with Express:

```
import express from 'express';
const app = express();
app.use(express.static('public')); // place
video files in /public
app.listen(3000);
```

HTML:

```
<video controls playsinline
style="width:100%">
  <source src="/my-video.mp4"
type="video/mp4" />
</video>
```

Avoid deep relative paths like ../../my-video.mp4 since your deploy build may change folder structure.

6. Serve the right MIME type
Your server must send video/mp4 for mp4, application/vnd.apple.mpegurl for m3u8, and video/mp2t for ts segments. If you only see audio or nothing, your codec may not be supported. H.264 video with AAC audio is the safest baseline for mp4.

7. Do not cross origins unless CORS is set
If the video file is on a different domain, set CORS on the file server. Easiest fix is to host the file from the same origin as your app.

8. Handle HLS streams on non Safari browsers
If you are using .m3u8, Safari plays it natively, Chrome needs hls.js.

```
<video id="player" controls playsinline></
video>
<script src="https://cdn.jsdelivr.net/npm/
hls.js@latest"></script>
<script>
  const url = '/stream.m3u8';
  const video =
document.getElementById('player');
  if (video.canPlayType('application/
vnd.apple.mpegURL')) {
    video.src = url;
  } else if (Hls.isSupported()) {
    const hls = new Hls();
    hls.loadSource(url);
    hls.attachMedia(video);
  } else {
    console.error('HLS not supported');
  }
</script>
```

9. Content Security Policy can block media
If you set a CSP header, add media and connect sources that include your video host.

Content–Security–Policy: default–src 'self'; media–src 'self' https: blob:; connect–src 'self' https:; img–src 'self' data: https: blob:;

10. React or Vite build quirks

Place assets under public, then reference with /file.mp4, since bundlers may fingerprint files.

For Next.js, place under public as well and reference with /file.mp4.

If you import a video into React, confirm the build outputs it and the URL is valid in the network tab.

Debug like a pro

Open DevTools on the live site, check Console for media or permission errors.

Network tab, filter by media, confirm the video request status is 200 and the MIME type is correct.

Right click the page, View Source, confirm your video src is pointing where you think it is.

On mobile Safari, if the video goes fullscreen automatically, you likely missed playsinline or muted.

If nothing shows and no errors appear, try adding controls temporarily to see if the element is rendering.

Minimal working examples

Camera preview

```
<video id="cam" autoplay muted
playsinline style="width:100%"></video>
<button id="start">Start Camera</button>
<script>
  const start =
document.getElementById('start');
  start.addEventListener('click', async () =>
{
    try {
      const stream = await
navigator.mediaDevices.getUserMedia({
video: { facingMode: 'user' } });
      const v =
document.getElementById('cam');
      v.srcObject = stream;
      await v.play();
    } catch (err) {
      console.error(err.name, err.message);
      alert('Allow camera on this site and try
again.');
    }
  });
</script>
```

File playback

```
<video controls playsinline
style="width:100%">
  <source src="/sample.mp4" type="video/
mp4">
  Your browser does not support the video
tag.
</video>
```

Common Replit gotchas

Preview versus Deployment are different origins. Re−grant camera on the deployed domain.

Static files not copied, place them in public or configure your server to serve the folder you reference at build time.

If you switched to Deployments, confirm your server listens on the port Replit expects and serves static correctly.

If you use environment variables for a remote media URL, confirm they are set in the Deployment environment, not only in the Dev run.

If you paste a snippet of how you load the video now, I will pinpoint the exact fix and rewrite it line for line.