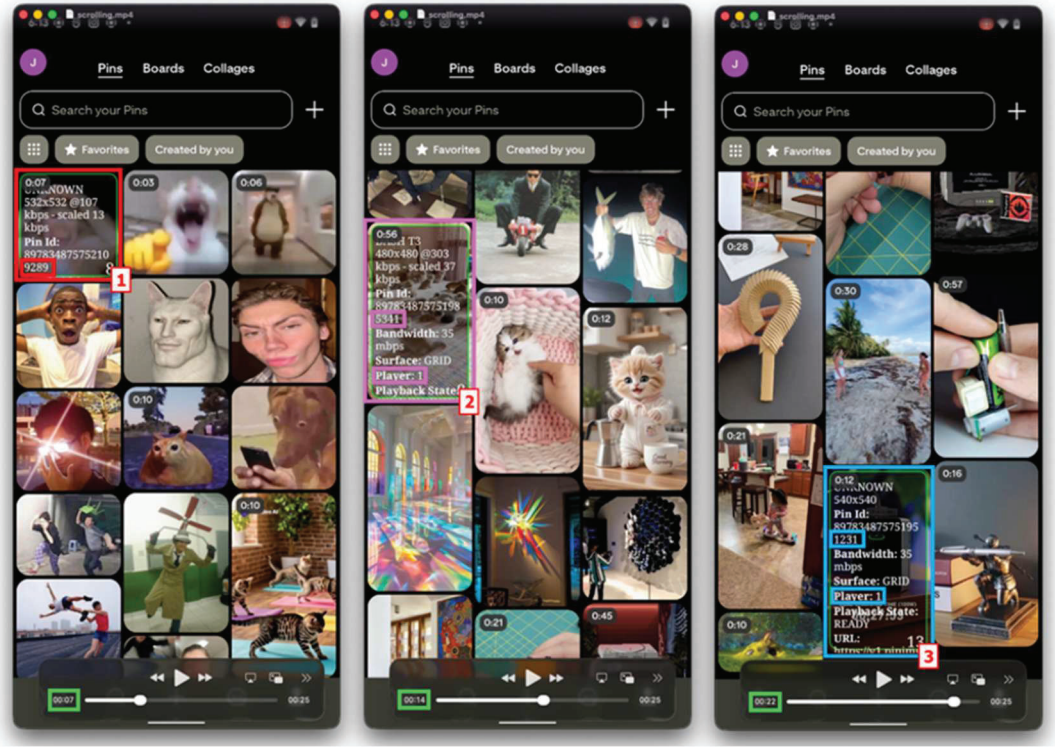
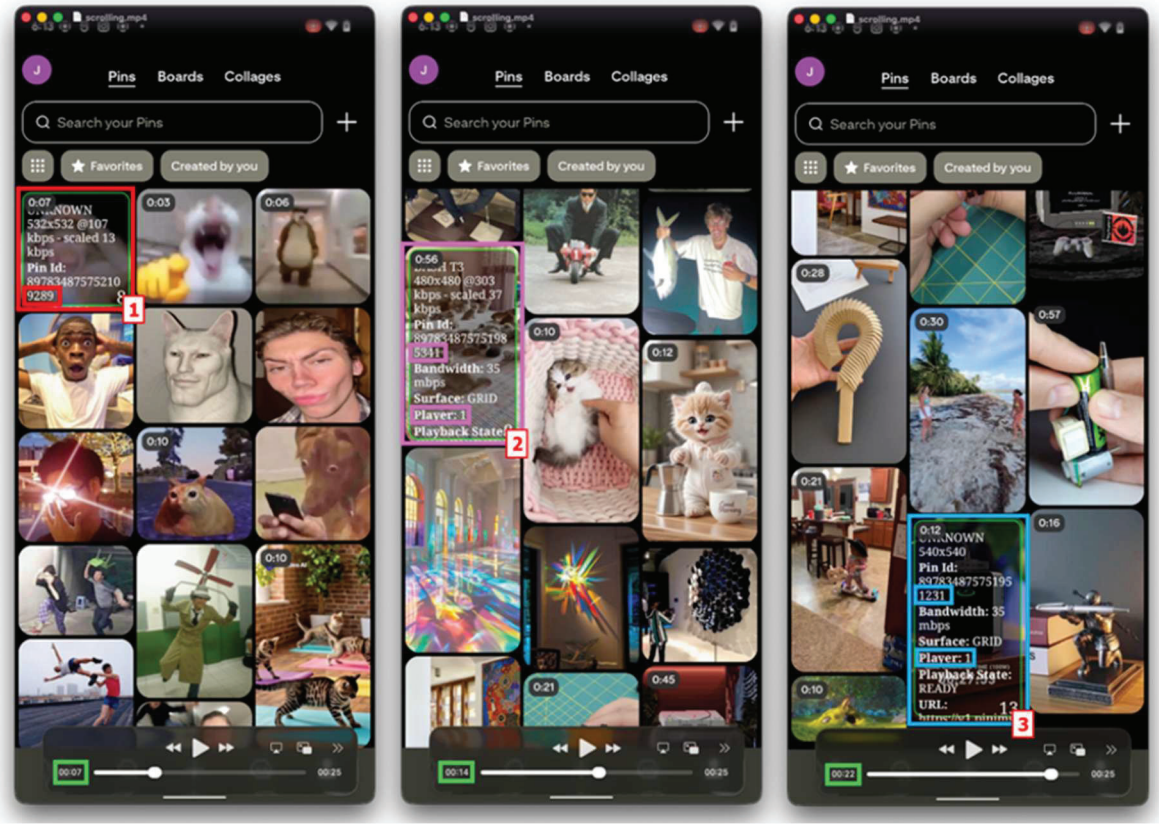
Appendix A

No.	Claim Language	Exemplary Infringement Support
1	A method performed by a digital receiver, the method comprising:	The Accused Pinterest Applications perform a method by a digital receiver. A digital receiver could include, for example, hardware and/or software found in a smartphone, tablet, laptop, TV, or desktop computer. Pinterest makes, uses, offers, sells, or imports products, software, or services on its applications for iOS, iPadOS, Android, and Windows (the "Accused Pinterest Applications"). When the Accused Pinterest Applications are used on a smartphone, tablet, laptop, or desktop, Pinterest's video Pin system performs a method by a digital receiver. For example, an Android smartphone constitutes a "digital receiver."

No.	Claim Language	Exemplary Infringement Support
		 Pinterest offers the Accused Pinterest Applications for use on digital receivers. *Pinterest Home Page*, PINTEREST, https://www.pinterest.com/ (last visited Apr. 14, 2026). (OPENTV_1404_0001040).
1[a]	receiving a selection of a new piece of content for playback during playback of a first piece of content;	The Accused Pinterest Applications perform receiving a selection of a new piece of content for playback during playback of a first piece of content. For example, as a user of the Pinterest Android application scrolls through their feed, the Pinterest Android application receives a selection of a new piece of content (i.e., a new video Pin) for playback while a first piece of content (i.e., a first video Pin) is still playing.

No.	Claim Language	Exemplary Infringement Support
		This is evidenced by operation, for example, of the Pinterest Android application. The three screenshots below were taken progressively as a user scrolled through their feed. As the user scrolls through their feed as depicted in the middle and right screenshots, the Pinterest Android application receives a selection of a third video Pin **[3]** (a new piece of content) for playback while a second video Pin **[2]** (a first piece of content) is still playing.  (OPENTV_1404_0001000.)

No.	Claim Language	Exemplary Infringement Support
		Pinterest's official help documentation describes these user interactions as available during video playback. For mobile platforms, Pinterest states that "Video Pins play on [a] mobile device when 50% of the video Pin is in view. To watch the video in a larger view, tap the Pin." Pinterest, *Manage Video Autoplay Settings*, https://help.pinterest.com/en/article/video-onpinterest (last accessed Nov. 18, 2025). (OPENTV_1404_0000825.) Pinterest's documentation further states that "[v]iewers can tap on the video to play it in a larger view, replay it, or play it with audio." Pinterest, *Video Ads*, https://help.pinterest.com/en/business/article/promoted-video-withautoplay (last accessed Nov. 18, 2025). (OPENTV_1404_0001006.) Either the process of initiating autoplay of a new video Pin or tapping on a new video Pin (while a different video Pin is playing) would constitute receiving a selection of a new piece of content for playback during playback of a first piece of content.
1[b]	in response to the receiving of the selection, transitioning to the new piece of content;	The Accused Pinterest Applications perform in response to the receiving of the selection, transitioning to the new piece of content. For example, in response to the receiving of the selection, the Pinterest Android application transitions to the new piece of content (i.e., the new video Pin). This is evidenced by operation, for example, of the Pinterest Android application. In response to a user scrolling through their feed on the Pinterest Android application, the Pinterest Android application transitions from playing a first piece of content (video Pin **[2]**) to a new piece of content (video Pin **[3]**).

No.	Claim Language	Exemplary Infringement Support
		 (OPENTV_1404_0001000.)
1[c]	determining whether the new piece of content is of a same content type	The Accused Pinterest Applications perform determining whether the new piece of content is of a same content type as the first piece of content, and based on the new piece of content being the same

No.	Claim Language	Exemplary Infringement Support
	as the first piece of content; and based on the new piece of content being the same content type as the first piece of content, determining whether to preserve at least a portion of a playback pipeline used to play back the first piece of content,	content type as the first piece of content, determining whether to preserve at least a portion of a playback pipeline used to play back the first piece of content. For example, the Pinterest Android application determines whether the new piece of content (i.e., the new video pin) is of a same content type as the first piece of content (i.e., a first video Pin); and based on the new piece of content being the same content type as the first piece of content, determining whether to preserve at least a portion of a playback pipeline used to play back the first piece of content. This is evidenced by public information regarding the Pinterest Android application. Pinterest employs a modified version of ExoPlayer for playback of video content on Pinterest's Android application. *See* Medium, *Improving ABR Video Performance at Pinterest*, https://medium.com/pinterest-engineering/improving-abr-video-performance-at-pinterest-f0ea47a6d4fc (last visited Apr. 17, 2026). (OPENTV_1404_0000800-OPENTV_1404_0000814.) An AndroidX Media3 blog post describes an improvement in how ExoPlayer 2.10 reuses audio and video decoders. Olly Woodman, *Improved Decoder Reuse in ExoPlayer*, MEDIUM, https://medium.com/googleexoplayer/improved-decoder-reuse-in-exoplayer-ef4c6d99591d (last visited Nov. 18, 2025). (OPENTV_1404_0000792-OPENTV_1404_0000799.)

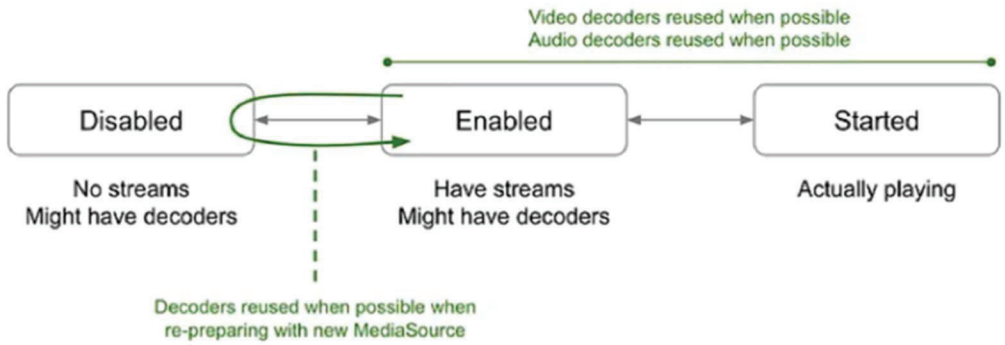
No.	Claim Language	Exemplary Infringement Support
		In order to reuse decoders for video playback, ExoPlayer must, for example, determine whether content shares the same codec: **Requirements for decoder reuse** Decoders cannot always be reused. One obvious requirement is that the decoders held by the player must be the ones that are required to play the new piece of media. Hence it's not possible to reuse a video decoder when switching between VP9 and H.264 playbacks, for example. The full story is significantly more complicated, with multiple modes of decoder reuse and conditions for each one. These details are beyond the scope of this blog post, however an interested reader can find the relevant code blocks here for audio, and here for video. *Id.* (OPENTV_1404_0000796.)

No.	Claim Language	Exemplary Infringement Support

		Publicly available ExoPlayer code for version 2.10 as identified by the AndroidX Media3 blog post includes a "canKeepCodec" function that tests for possible reuse of a video decoder:
		
		501 @Override
		502 protected @KeepCodecResult int canKeepCodec(
		503 MediaCodec codec, MediaCodecInfo codecInfo, Format oldFormat, Format newFormat) {
		504 if (codecInfo.isSeamlessAdaptationSupported(
		505 oldFormat, newFormat, /* isNewFormatComplete= */ true)
		506 && newFormat.width <= codecMaxValues.width
		507 && newFormat.height <= codecMaxValues.height
		508 && getMaxInputSize(codecInfo, newFormat) <= codecMaxValues.inputSize) {
		509 return oldFormat.initializationDataEquals(newFormat)
		510 ? KEEP_CODEC_RESULT_YES_WITHOUT_RECONFIGURATION
		511 : KEEP_CODEC_RESULT_YES_WITH_RECONFIGURATION;
		512 }
		513 return KEEP_CODEC_RESULT_NO;
		514 }
		
		ExoPlayer, *Library*, GITHUB, at lines 501-514, https://github.com/google/ExoPlayer/blob/92a7bb534a706fe917308af48a4958c99b3bb754/library/core/src/main/java/com/google/android/exoplayer2/video/MediaCodecVideoRenderer.java#L502 (last visited Apr. 17, 2026) (annotated). (OPENTV_1404_0000910 at OPENTV_1404_0000918.)

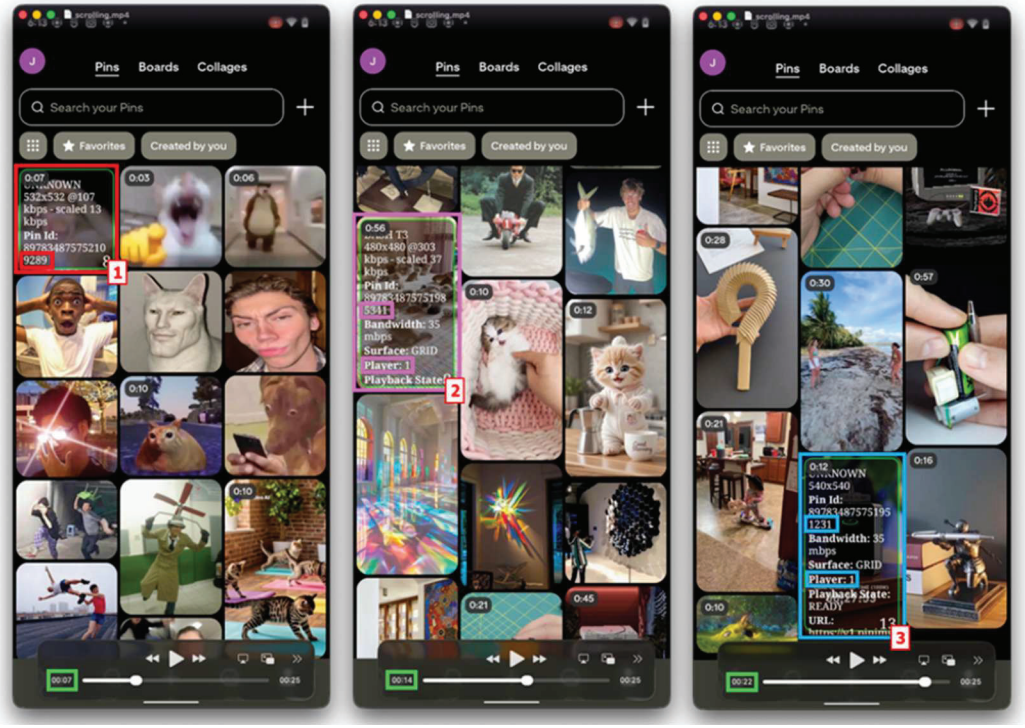
No.	Claim Language	Exemplary Infringement Support

		Similar functionality exists in later versions of Android ExoPlayer by way of the function "canReuseCodec":
		
		1247 @Override
		1248 protected DecoderReuseEvaluation canReuseCodec(
		1249 MediaCodecInfo codecInfo, Format oldFormat, Format newFormat) {
		1250 DecoderReuseEvaluation evaluation = codecInfo.canReuseCodec(oldFormat, newFormat);
		1251
		1252 @DecoderDiscardReasons int discardReasons = evaluation.discardReasons;
		1253 CodecMaxValues codecMaxValues = checkNotNull(this.codecMaxValues);
		1254 if (newFormat.width > codecMaxValues.width || newFormat.height > codecMaxValues.height) {
		1255 discardReasons |= DISCARD_REASON_VIDEO_MAX_RESOLUTION_EXCEEDED;
		1256 }
		1257 if (getMaxInputSize(codecInfo, newFormat) > codecMaxValues.inputSize) {
		1258 discardReasons |= DISCARD_REASON_MAX_INPUT_SIZE_EXCEEDED;
		1259 }
		1260
		1261 return new DecoderReuseEvaluation(
		1262 codecInfo.name,
		1263 oldFormat,
		1264 newFormat,
		1265 discardReasons != 0 ? REUSE_RESULT_NO : evaluation.result,
		1266 discardReasons);
		1267 }
		
		AndroidX, *Media*, GITHUB, at lines 1247-1267, https://github.com/androidx/media/blob/1.6.1/libraries/exoplayer/src/main/java/androidx/media3/exoplayer/video/MediaCodecVideoRenderer.java (last visited Apr. 17, 2026). (OPENTV_1404_0000936 at OPENTV_1404_0000956.)

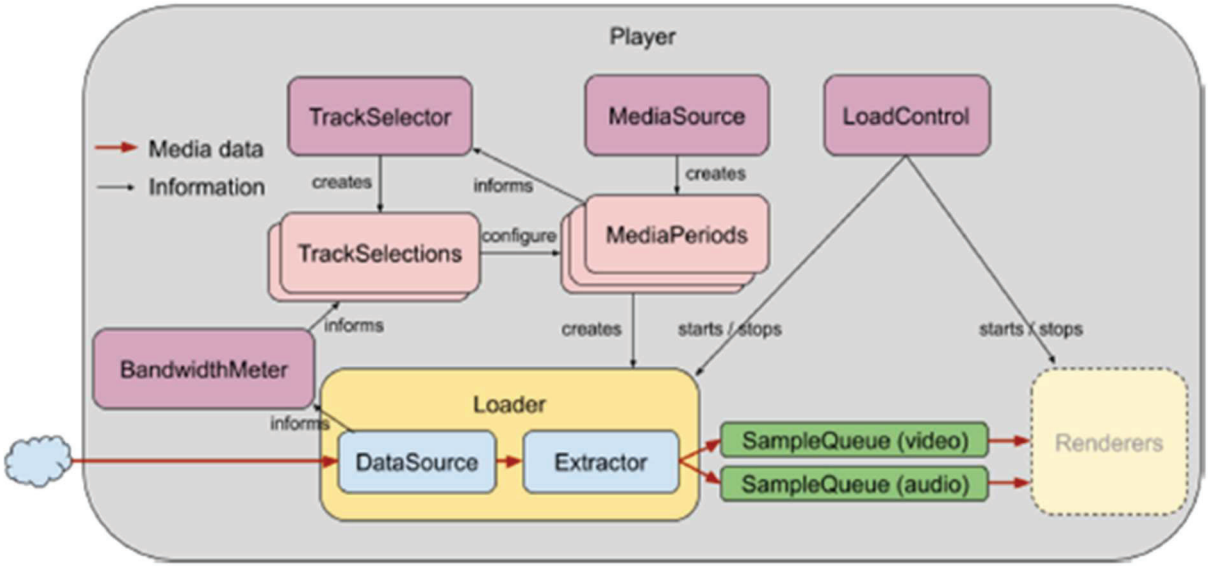
No.	Claim Language	Exemplary Infringement Support
		As of ExoPlayer 2.10, decoders may be reused when re-preparing the player to play a different MediaSource: **Improved decoder reuse in 2.10** In ExoPlayer 2.10 we've made changes that enable audio decoders to be reused. We've also enabled decoder reuse when re-preparing the player to play a different MediaSource, by allowing renderers to hold decoders when transitioning through the disabled state. These changes are summarized in the diagram below.  Decoder reuse in ExoPlayer 2.10 Olly Woodman, *Improved Decoder Reuse in ExoPlayer*, MEDIUM, https://medium.com/googleexoplayer/improved-decoder-reuse-in-exoplayer-ef4c6d99591d (last visited Nov. 18, 2025). (OPENTV_1404_0000792 at OPENTV_1404_0000795.)

No.	Claim Language	Exemplary Infringement Support
		When switching between videos and when re-preparing the player with a different video, a single ExoPlayer instance should be used for all playbacks: **Making sure your app benefits from decoder reuse** To ensure that your app benefits from decoder reuse, it's important that: 1. For use cases where your app switches between playing a number of videos, use a single ExoPlayer instance for all playbacks, rather than instantiating a new instance each time. This is because decoders are only reused within an ExoPlayer instance, not between them. 2. When re-preparing the player with a different video, call `prepare()` with the new `MediaSource` without calling `stop()` first. This is because by default, ExoPlayer will still release decoders when `stop()` is called. Top highlight *Id.* (OPENTV_1404_0000792 at OPENTV_1404_0000796-OPENTV_1404_0000797.)

No.	Claim Language	Exemplary Infringement Support
		Pinterest's Android application implements the decoder reuse functionality of ExoPlayer. This is evidenced, for example, by Pinterest blog posts describing the use of pooling and recycling players: **Pooling Players** Lastly, we built our own cache to pool players as they're needed. Historically, we instantiated new player instances on the fly, which caused significant overhead on both the memory and bandwidth. Here are some high level learnings: **Separate By Encoding** Player instances are effectively holding onto a reference to an underlying decoder based on the last rendered media's encoding type. It takes a measurable amount of work for players to switch context between different decoders. Therefore we pooled our players based on the initial decoding format. By ensuring the recycled players with decoders matching the media's encoding, we removed any latency overhead caused when switching encoding formats. *See* Grey Skold, *Improving the Player on Android*, MEDIUM, https://medium.com/pinterest-engineering/improving-the-player-on-android-8b7faf9009cf (last visited Apr. 17, 2026) (Pinterest Engineering Blog discussing ExoPlayer modifications). (OPENTV_0000815 at OPENTV_1404_0000819-OPENTV_1404_0000820.)

No.	Claim Language	Exemplary Infringement Support
		Pinterest's implementation of the decoder reuse functionality of ExoPlayer is further evidenced by the fact that Pinterest uses a single ExoPlayer instance for playback of video Pins when a user is scrolling through their feed. In particular, scrolling through a user's feed shows how a single player ("Player: 1") is used by both video Pin [2] and [3]:  (OPENTV_1404_0001000.)

No.	Claim Language	Exemplary Infringement Support
1[c](i)	the playback pipeline comprising a source element to manage the downloading of content data for the new piece of content, a demultiplexer, an audio decoder, and a video decoder,	The Accused Pinterest Applications include a playback pipeline comprising a source element to manage the downloading of content data for the new piece of content, a demultiplexer, an audio decoder, and a video decoder. For example, the Pinterest Android application employs multiple player instances, organized as a pool of player instances, to handle playback of individual videos within video Pins. Each player instance is an instance of an ExoPlayer class. *See* AndroidX, *ExoPlayer Module*, https://github.com/androidx/media/tree/1.6.1/libraries/exoplayer (last visited Apr. 17, 2026). (OPENTV_1404_0000783.)

No.	Claim Language	Exemplary Infringement Support
		As illustrated in the image below, the Pinterest Android application (and its implementation of ExoPlayer) include a playback pipeline that includes the media source, demultiplexer, decoder modules, and renderers required to execute playback.  Android Developers, *ExoPlayer*, https://developer.android.com/media/media3/exoplayer/glossary (last visited Apr. 17, 2026). (OPENTV_1404_0000784 at OPENTV_1404_0000788.)

No.	Claim Language	Exemplary Infringement Support
		For example, ExoPlayer, as implemented by Pinterest's Android application, contains a source element in the form of the DefaultTrackSelector class. As described in the open-source ExoPlayer media player library, the DefaultTrackSelector manages the selection and retrieval of media tracks, including adaptive streaming variants, and coordinates the delivery of the appropriate media stream for decoding and playback. • A `TrackSelector` that selects tracks provided by the MediaSource to be consumed by each of the available Renderers. The library provides a default implementation (`DefaultTrackSelector`) suitable for most use cases. A TrackSelector is injected when the player is created. *See* Android Developers, *ExoPlayer*, https://developer.android.com/reference/androidx/media3/exoplayer/ExoPlayer (last visited Nov. 18, 2025) (OPENTV_1404_0000727); *see also* Android Developers, https://developer.android.com/media/media3/exoplayer/track-selection (last visited Apr. 17, 2026) (OPENTV_1404_0001001 at OPENTV_1404_0001001-OPENTV_1404_0001005). The TrackSelector component performs the function of selecting the media tracks to be processed by the available renderers within the playback pipeline. When videos are streamed using MPEG-DASH, a protocol for adaptive bitrate streaming, the available "tracks" represent different encodings of the same video content at varying bitrates. • A track is *selected* if it has been chosen for playback given the current `TrackSelectionParameters`. If multiple tracks within one track group are selected, the player uses these tracks for adaptive playback (for example, multiple video tracks with different bitrates). Note that only one of these tracks will be played at any one time. *See* Android Developers, https://developer.android.com/media/media3/exoplayer/track-selection (last visited Apr. 17, 2026) (annotated). (OPENTV_1404_0001001 at OPENTV_1404_0001002.)

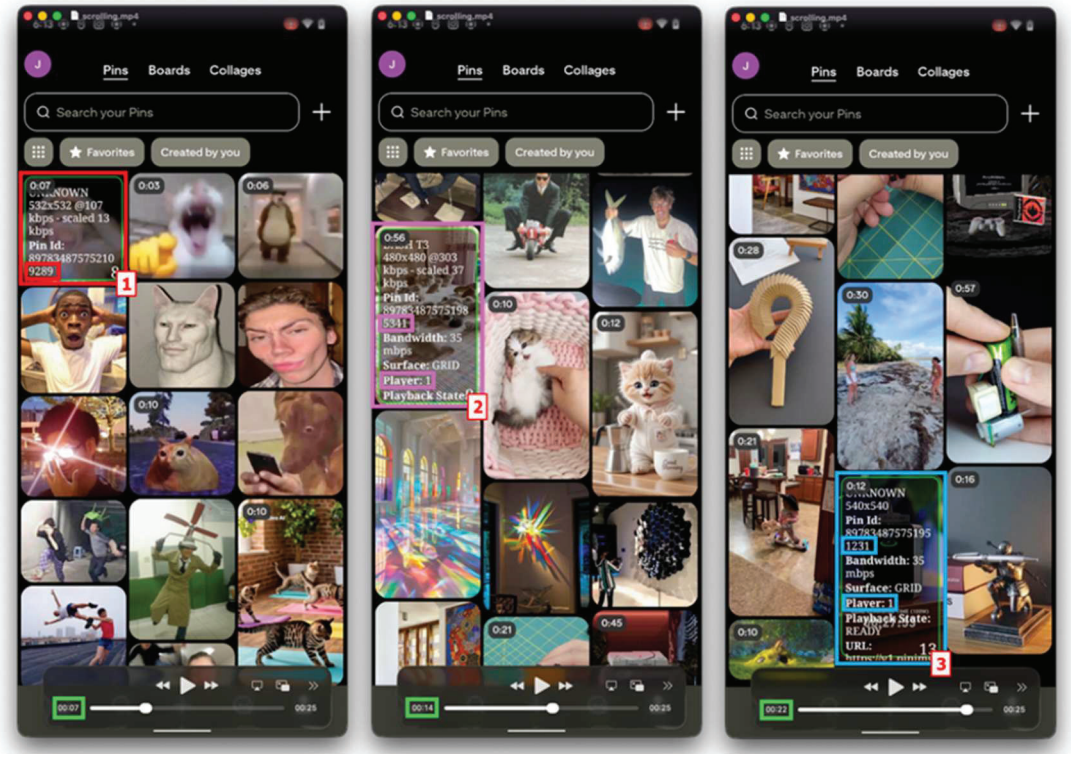
No.	Claim Language	Exemplary Infringement Support
		The TrackSelector component—particularly the DefaultTrackSelector used in ExoPlayer—determines which version of the video will be retrieved and played, selecting the stream most suitable for the current network conditions. To perform this selection, each TrackSelector maintains an associated BandwidthMeter object, which continuously monitors and estimates the available network bandwidth. As described in the open-source implementation, the BandwidthMeter acts as a "component that estimates the network bandwidth" and supplies that data to the TrackSelector for dynamic adaptation. In this configuration, the TrackSelector, together with its BandwidthMeter, manages the download and adaptation of the video stream, selecting the appropriate bitrate and ensuring efficient playback. **BandwidthMeter** Component that estimates the network bandwidth, for example by listening to data transfers. In adaptive streaming, bandwidth estimates can be used to select between different bitrate tracks during playback. For more information, see the component Javadoc. *See* Android Developers, *ExoPlayer*, https://developer.android.com/media/media3/exoplayer/glossary (last visited Apr. 30, 2026). (OPENTV_1404_0000784 at OPENTV_1404_0000789.) The Pinterest Android application utilizes videos encoded within the MP4 container format, as specified in Pinterest's official documentation. *Review Pin Specs*, Pinterest, https://help.pinterest.com/en/article/review-pin-specs (last visited Apr. 17. 2026). (OPENTV_1404_0000995 at OPENTV_1404_0000997.) The MP4 format defines the structure of a data container that encapsulates both audio and video streams within a single file. To play such media, it is technically necessary for the receiving system to download the MP4 file, demultiplex it—that is, unpack the audio and video streams—and subsequently direct those separate streams to

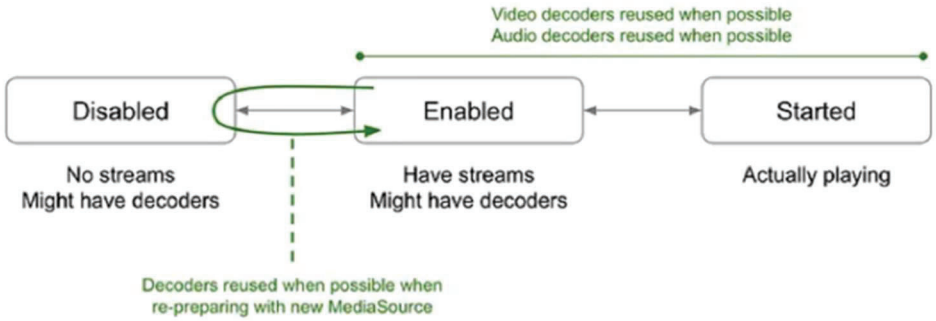
No.	Claim Language	Exemplary Infringement Support
		appropriate audio and video decoders. The decoded outputs are then rendered synchronously by the application's playback engine to present continuous audiovisual playback to the user. Thus, ExoPlayer, as implemented by Pinterest's Android application, contains a "demultiplexer" including a FragmentedMp4Extractor class. The FragmentedMp4Extractor class extracts data from an MP4 container format: FragmentedMp4Extractor Artifact: androidx.media3:media3-extractor View Source Kotlin | Java @UnstableApi public class FragmentedMp4Extractor implements Extractor Android Developers, *FragmentedMp4Extractor*, https://developer.android.com/reference/androidx/media3/extractor/mp4/FragmentedMp4Extractor (last visited Apr. 17, 2026). (OPENTV_1404_0000770.)

No.	Claim Language	Exemplary Infringement Support
		Extracting the data from the container format involves, in particular, separating it into audio and video data. Thus, the FragmentedMp4Extractor is a demultiplexer. *See id.* (OPENTV_1404_0000770 at OPENTV_1404_0000770-OPENTV_1404_0000782.) ExoPlayer, as implemented by Pinterest's Android application, contains an "audio decoder" including a MediaCodecAudioRenderer class as part of a player instance: **MediaCodecAudioRenderer** Artifact: androidx.media3:media3-exoplayer View Source Kotlin | **Java** @UnstableApi public class MediaCodecAudioRenderer extends MediaCodecRenderer implements MediaClock java.lang.Object ↳ androidx.media3.exoplayer.BaseRenderer ↳ androidx.media3.exoplayer.mediacodec.MediaCodecRenderer ↳ androidx.media3.exoplayer.audio.MediaCodecAudioRenderer Decodes and renders audio using `MediaCodec` and an `AudioSink`. Android Developers, *MediaCodecAudioRenderer*, https://developer.android.com/reference/androidx/media3/exoplayer/audio/MediaCodecAudioRenderer (last visited Apr. 17, 2026) (annotated). (OPENTV_1404_0001010.)

No.	Claim Language	Exemplary Infringement Support
		According to Android documentation, the MediaCodecAudioRenderer class "[d]ecodes and renders audio[.]" *Id.* (OPENTV_1404_0001010.) Thus, the MediaCodecAudioRenderer is an audio decoder. ExoPlayer, as implemented by Pinterest's Android application, contains a "video decoder" including a MediaCodecVideoRenderer class: **MediaCodecVideoRenderer** Artifact: androidx.media3:media3-exoplayer View Source Kotlin | Java @UnstableApi public class MediaCodecVideoRenderer extends MediaCodecRenderer implements VideoFrameReleaseControl java.lang.Object ↳ androidx.media3.exoplayer.BaseRenderer ↳ androidx.media3.exoplayer.mediacodec.MediaCodecRenderer ↳ androidx.media3.exoplayer.video.MediaCodecVideoRenderer Known direct subclasses CapturingRendererFactory.CapturingMediaCodecVideoRenderer Decodes and renders video using `MediaCodec`. Android Developers, *MediaCodecVideoRenderer*, https://developer.android.com/reference/androidx/media3/exoplayer/video/MediaCodecVideoRenderer (last visited Apr. 17, 2026) (annotated). (OPENTV_1404_0000863.)

No.	Claim Language	Exemplary Infringement Support
		According to Android documentation, the MediaCodecVideoRenderer class "[d]ecodes and renders video." *Id*. (OPENTV_1404_0000863.) Thus, the MediaCodecVideoRenderer is a video decoder. It necessarily follows that the Pinterest Android application implements a playback pipeline containing the components described in this limitation. Without such a pipeline—including the necessary source, demultiplexer, audio and video decoder, and rendering elements—it would be technically impossible for the application to perform MP4 playback as observed. The functioning of these components within Pinterest's video playback framework therefore demonstrates the existence of the claimed playback pipeline.
1[c](ii)	the preserving of at least the portion of the playback pipeline comprising preserving one or more of the source element, the demultiplexer, the audio decoder, or the video decoder used to obtain the first piece of content for the new piece of content.	The Accused Pinterest Applications perform the method wherein based on the new piece of content being the same content type as the first piece of content, the preserving of at least the portion of the playback pipeline comprising preserving one or more of the source element, the demultiplexer, the audio decoder, or the video decoder used to obtain the first piece of content for the new piece of content. For example, the Pinterest Android application, when using ExoPlayer version 2.10 or greater, preserves at least the audio and/or video decoder used for a first video Pin to playback a second selected video Pin.

No.	Claim Language	Exemplary Infringement Support
		This is evidenced by the fact that the Pinterest Android application will re-use the same ExoPlayer instance ("Player: 1") for at least each of video Pins [2] and [3] as the user scrolls through a feed:  (OPENTV_1404_0001000.)

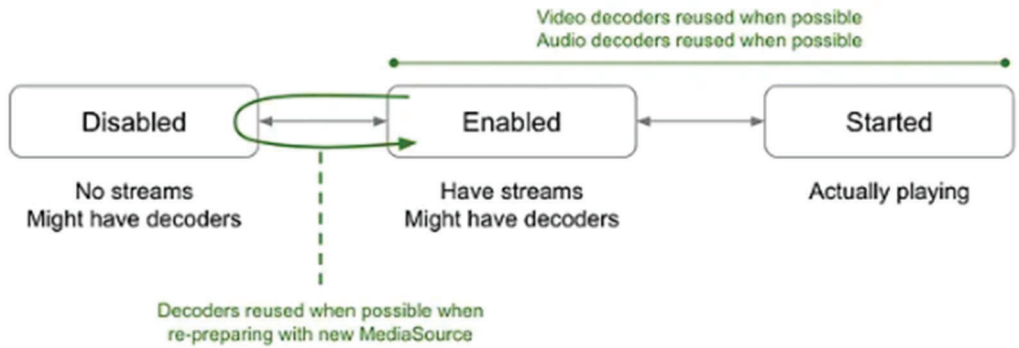
No.	Claim Language	Exemplary Infringement Support
		This reuse of ExoPlayer instances shows the reuse of audio and video decoders when playing different media sources: **Improved decoder reuse in 2.10** In ExoPlayer 2.10 we've made changes that enable audio decoders to be reused. We've also enabled decoder reuse when re-preparing the player to play a different MediaSource, by allowing renderers to hold decoders when transitioning through the disabled state. These changes are summarized in the diagram below.  Decoder reuse in ExoPlayer 2.10 Olly Woodman, *Improved Decoder Reuse in ExoPlayer*, MEDIUM, https://medium.com/googleexoplayer/improved-decoder-reuse-in-exoplayer-ef4c6d99591d (last visited Apr. 17, 2026). (OPENTV_1404_0000792 at OPENTV_1404_0000795.)

No.	Claim Language	Exemplary Infringement Support
		Pinterest blog posts similarly describe the reuse of player instances (including an underlying decoder): **Separate By Encoding** Player instances are effectively holding onto a reference to an underlying decoder based on the last rendered media's encoding type. It takes a measurable amount of work for players to switch context between different decoders. Therefore we pooled our players based on the initial decoding format. By ensuring the recycled players with decoders matching the media's encoding, we removed any latency overhead caused when switching encoding formats. Grey Skold, *Improving the Player on Android*, MEDIUM, https://medium.com/pinterest-engineering/improving-the-player-on-android-8b7faf9009cf (last visited Apr. 17, 2026) (Pinterest Engineering Blog discussing ExoPlayer modifications) (annotated). (OPENTV_1404_0000815 at OPENTV_1404_0000819-OPENTV_1404_0000820.)
2	The method of claim 1, wherein the determining whether to preserve comprises: detecting codec information for the new piece of content, the codec	The Accused Pinterest Applications perform the method of claim 1, wherein determining whether to preserve comprises: detecting codec information for the new piece of content, the codec information indicating types of decoders needed for the new piece of content.

No.	Claim Language	Exemplary Infringement Support
	information indicating types of decoders needed for the new piece of content; and	For example, the Pinterest Android application detects codec information for a new video Pin to determine whether an existing player instance may be reused: **Separate By Encoding** Player instances are effectively holding onto a reference to an underlying decoder based on the last rendered media's encoding type. It takes a measurable amount of work for players to switch context between different decoders. Therefore we pooled our players based on the initial decoding format. By ensuring the recycled players with decoders matching the media's encoding, we removed any latency overhead caused when switching encoding formats. Grey Skold, *Improving the Player on Android*, MEDIUM, https://medium.com/pinterest-engineering/improving-the-player-on-android-8b7faf9009cf (last visited Apr. 17, 2026) (Pinterest Engineering Blog discussing ExoPlayer modifications) (annotated). (OPENTV_1404_0000815 at OPENTV_1404_0000819-OPENTV_1404_0000820.) And as further described below, one of the requirements for decoder reuse in ExoPlayer, as implemented by Pinterest's Android application, is that "decoders held by the player must be the ones that are required to play the new piece of media."

No.	Claim Language	Exemplary Infringement Support
		Thus, Pinterest's Android application must detect codec information for the new piece of content as claimed. **Requirements for decoder reuse** Decoders cannot always be reused. One obvious requirement is that the decoders held by the player must be the ones that are required to play the new piece of media. Hence it's not possible to reuse a video decoder when switching between VP9 and H.264 playbacks, for example. The full story is significantly more complicated, with multiple modes of decoder reuse and conditions for each one. These details are beyond the scope of this blog post, however an interested reader can find the relevant code blocks here for audio, and here for video. Olly Woodman, *Improved Decoder Reuse in ExoPlayer*, MEDIUM, https://medium.com/googleexoplayer/improved-decoder-reuse-in-exoplayer-ef4c6d99591d (last visited Apr. 17, 2026). (OPENTV_1404_0000792 at OPENTV_1404_0000796.)
2[a]	determining whether the audio decoder and the video decoder in the playback pipeline are of the types indicated by the codec information.	The Accused Pinterest Applications perform determining whether the audio decoder and the video decoder in the playback pipeline are of the types indicated by the codec information. For example, the Pinterest Android application, when using ExoPlayer version 2.10 or greater, determines whether the audio decoder and the video decoder in the playback pipeline are of the types indicated by the codec information for the new video Pin.

No.	Claim Language	Exemplary Infringement Support
		Pinterest engineers describe how the Pinterest Android application matches recycled players with a media's (i.e., video Pin's) encoding to determine whether a player may be reused. **Separate By Encoding** Player instances are effectively holding onto a reference to an underlying decoder based on the last rendered media's encoding type. It takes a measurable amount of work for players to switch context between different decoders. Therefore we pooled our players based on the initial decoding format. By ensuring the recycled players with decoders matching the media's encoding, we removed any latency overhead caused when switching encoding formats. Grey Skold, *Improving the Player on Android*, MEDIUM, https://medium.com/pinterest-engineering/improving-the-player-on-android-8b7faf9009cf (last visited Apr. 17, 2026) (annotated). (OPENTV_1404_0000815 at OPENTV_1404_0000819-OPENTV_1404_0000820.)

No.	Claim Language	Exemplary Infringement Support
		And as further described below, ExoPlayer allows for the reuse of both audio and video decoders: **Improved decoder reuse in 2.10** In ExoPlayer 2.10 we've made changes that enable audio decoders to be reused. We've also enabled decoder reuse when re-preparing the player to play a different MediaSource, by allowing renderers to hold decoders when transitioning through the disabled state. These changes are summarized in the diagram below.  Decoder reuse in ExoPlayer 2.10 Olly Woodman, *Improved Decoder Reuse in ExoPlayer*, MEDIUM, https://medium.com/googleexoplayer/improved-decoder-reuse-in-exoplayer-ef4c6d99591d (last visited Apr. 17, 2026). (OPENTV_1404_0000792 at OPENTV_1404_0000795.)

No.	Claim Language	Exemplary Infringement Support
		One of the requirements for decoder reuse in ExoPlayer, as implemented by Pinterest's Android application, is that "decoders held by the player must be the ones that are required to play the new piece of media." Thus, Pinterest's Android application must determine whether the audio decoder and the video decoder in the playback pipeline are of the types indicated by the codec information. **Requirements for decoder reuse** Decoders cannot always be reused. One obvious requirement is that the decoders held by the player must be the ones that are required to play the new piece of media. Hence it's not possible to reuse a video decoder when switching between VP9 and H.264 playbacks, for example. The full story is significantly more complicated, with multiple modes of decoder reuse and conditions for each one. These details are beyond the scope of this blog post, however an interested reader can find the relevant code blocks here for audio, and here for video. *Id.* (OPENTV_1404_0000796.)

No.	Claim Language	Exemplary Infringement Support

		Publicly available ExoPlayer code for version 2.10 as identified by the AndroidX Media3 blog post includes a "canKeepCodec" function that tests for possible reuse of video or audio decoders:
		
		501 @Override
		502 protected @KeepCodecResult int canKeepCodec(
		503 MediaCodec codec, MediaCodecInfo codecInfo, Format oldFormat, Format newFormat) {
		504 if (codecInfo.isSeamlessAdaptationSupported(
		505 oldFormat, newFormat, /* isNewFormatComplete= */ true)
		506 && newFormat.width <= codecMaxValues.width
		507 && newFormat.height <= codecMaxValues.height
		508 && getMaxInputSize(codecInfo, newFormat) <= codecMaxValues.inputSize) {
		509 return oldFormat.initializationDataEquals(newFormat)
		510 ? KEEP_CODEC_RESULT_YES_WITHOUT_RECONFIGURATION
		511 : KEEP_CODEC_RESULT_YES_WITH_RECONFIGURATION;
		512 }
		513 return KEEP_CODEC_RESULT_NO;
		514 }
		
		ExoPlayer, *Library*, GITHUB, at lines 501-514, https://github.com/google/ExoPlayer/blob/92a7bb534a706fe917308af48a4958c99b3bb754/library/core/src/main/java/com/google/android/exoplayer2/video/MediaCodecVideoRenderer.java#L502 (last visited Apr. 17, 2026) (annotated). (OPENTV_1404_0000910 at OPENTV_1404_0000918.)

No.	Claim Language	Exemplary Infringement Support

		365 @Override
		366 protected @KeepCodecResult int canKeepCodec(
		367 MediaCodec codec, MediaCodecInfo codecInfo, Format oldFormat, Format newFormat) {
		368 // TODO: We currently rely on recreating the codec when encoder delay or padding is non-zero.
		369 // Re-creating the codec is necessary to guarantee that onOutputFormatChanged is called, which
		370 // is where encoder delay and padding are propagated to the sink. We should find a better way to
		371 // propagate these values, and then allow the codec to be re-used in cases where this would
		372 // otherwise be possible.
		373 if (getCodecMaxInputSize(codecInfo, newFormat) > codecMaxInputSize
		374 || oldFormat.encoderDelay != 0
		375 || oldFormat.encoderPadding != 0
		376 || newFormat.encoderDelay != 0
		377 || newFormat.encoderPadding != 0) {
		378 return KEEP_CODEC_RESULT_NO;
		379 } else if (codecInfo.isSeamlessAdaptationSupported(
		380 oldFormat, newFormat, /* isNewFormatComplete= */ true)) {
		381 return KEEP_CODEC_RESULT_YES_WITHOUT_RECONFIGURATION;
		382 } else if (areCodecConfigurationCompatible(oldFormat, newFormat)) {
		383 return KEEP_CODEC_RESULT_YES_WITH_FLUSH;
		384 } else {
		385 return KEEP_CODEC_RESULT_NO;
		386 }
		387 }
		
		ExoPlayer, *Library*, GITHUB, at lines 365-387, https://github.com/google/ExoPlayer/blob/92a7bb534a706fe917308af48a4958c99b3bb754/library/core/src/main/java/com/google/android/exoplayer2/audio/MediaCodecAudioRenderer.java#L366 (last visited Apr. 17, 2026) (annotated). (OPENTV_1404_0000827 at OPENTV_1404_0000833.)

No.	Claim Language	Exemplary Infringement Support

		Similar functionality for video decoder reuse exists in later versions of Android ExoPlayer by way of the function "canReuseCodec":
		
		1247 @Override
		1248 protected DecoderReuseEvaluation canReuseCodec(
		1249 MediaCodecInfo codecInfo, Format oldFormat, Format newFormat) {
		1250 DecoderReuseEvaluation evaluation = codecInfo.canReuseCodec(oldFormat, newFormat);
		1251
		1252 @DecoderDiscardReasons int discardReasons = evaluation.discardReasons;
		1253 CodecMaxValues codecMaxValues = checkNotNull(this.codecMaxValues);
		1254 if (newFormat.width > codecMaxValues.width || newFormat.height > codecMaxValues.height) {
		1255 discardReasons |= DISCARD_REASON_VIDEO_MAX_RESOLUTION_EXCEEDED;
		1256 }
		1257 if (getMaxInputSize(codecInfo, newFormat) > codecMaxValues.inputSize) {
		1258 discardReasons |= DISCARD_REASON_MAX_INPUT_SIZE_EXCEEDED;
		1259 }
		1260
		1261 return new DecoderReuseEvaluation(
		1262 codecInfo.name,
		1263 oldFormat,
		1264 newFormat,
		1265 discardReasons != 0 ? REUSE_RESULT_NO : evaluation.result,
		1266 discardReasons);
		1267 }
		
		AndroidX, *Media*, GITHUB, at lines 1247-1267, https://github.com/androidx/media/blob/1.6.1/libraries/exoplayer/src/main/java/androidx/media3/exoplayer/video/MediaCodecVideoRenderer.java (last visited Apr. 17, 2026). (OPENTV_1404_0000936 at OPENTV_1404_0000956.)

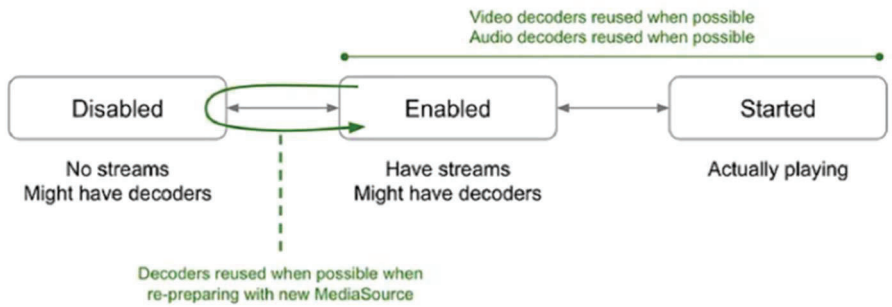
No.	Claim Language	Exemplary Infringement Support

		Similar functionality for audio decoder reuse exists in later versions of Android ExoPlayer by way of the function "canReuseCodec":
		
		512 @Override
		513 protected DecoderReuseEvaluation canReuseCodec(
		514 MediaCodecInfo codecInfo, Format oldFormat, Format newFormat) {
		515 DecoderReuseEvaluation evaluation = codecInfo.canReuseCodec(oldFormat, newFormat);
		516
		517 @DecoderDiscardReasons int discardReasons = evaluation.discardReasons;
		518 if (isBypassPossible(newFormat)) {
		519 // We prefer direct audio playback so that for multi-channel tracks the audio is not downmixed
		520 // to stereo.
		521 discardReasons |= DecoderReuseEvaluation.DISCARD_REASON_AUDIO_BYPASS_POSSIBLE;
		522 }
		523 if (getCodecMaxInputSize(codecInfo, newFormat) > codecMaxInputSize) {
		524 discardReasons |= DISCARD_REASON_MAX_INPUT_SIZE_EXCEEDED;
		525 }
		526
		527 return new DecoderReuseEvaluation(
		528 codecInfo.name,
		529 oldFormat,
		530 newFormat,
		531 discardReasons != 0 ? REUSE_RESULT_NO : evaluation.result,
		532 discardReasons);
		533 }
		
		AndroidX, *Media*, GITHUB, at lines 512-533, https://github.com/androidx/media/blob/7ce3aa2619b19009e4799319b4dd694a4a7577df/libraries/exoplayer/src/main/java/androidx/media3/exoplayer/audio/MediaCodecAudioRenderer.java#L513 (last visited Apr. 17, 2026). (OPENTV_1404_0000842 at OPENTV_1404_0000850.)

No.	Claim Language	Exemplary Infringement Support
		162 /** 163 * @param decoderName The name of the decoder. 164 * @param oldFormat The {@link Format} for which the decoder was previously configured. 165 * @param newFormat The new {@link Format} being evaluated. 166 * @param result The {@link DecoderReuseResult result} of the evaluation. 167 * @param discardReasons One or more {@link DecoderDiscardReasons reasons} why the decoder cannot 168 * be reused, or {@code 0} if reuse is possible. 169 */ 170 public DecoderReuseEvaluation(171 String decoderName, 172 Format oldFormat, 173 Format newFormat, 174 @DecoderReuseResult int result, 175 @DecoderDiscardReasons int discardReasons) { 176 checkArgument(result == REUSE_RESULT_NO || discardReasons == 0); 177 checkArgument(!TextUtils.isEmpty(decoderName)); 178 this.decoderName = decoderName; 179 this.oldFormat = checkNotNull(oldFormat); 180 this.newFormat = checkNotNull(newFormat); 181 this.result = result; 182 this.discardReasons = discardReasons; 183 } AndroidX, *Media*, GITHUB, at lines 162-183, https://github.com/androidx/media/blob/7ce3aa2619b19009e4799319b4dd694a4a7577df/libraries/exoplayer/src/main/java/androidx/media3/exoplayer/DecoderReuseEvaluation.java#L37 (last visited Apr. 17, 2026). (OPENTV_1404_0000723 at OPENTV_1404_0000725.)
5	The method of claim 2, wherein the detecting the codec information	The Accused Pinterest Applications perform the method of claim 2, wherein the detecting the codec information for the new piece of content comprises examining prefetched metadata for the new piece of content.

No.	Claim Language	Exemplary Infringement Support
	for the new piece of content comprises examining prefetched metadata for the new piece of content.	For example, the Pinterest Android application detects codec information for the new piece of content (i.e., new video Pin) including by examining prefetched metadata (e.g., the manifest file) for the new piece of content. **Steps to start video playbacks** Startup latency is a key metric for evaluating video performance at Pinterest. Therefore, we will first examine the steps of starting an ABR video. To initiate playback, clients must obtain both the manifest files and partial media files from the CDN through network requests. Manifest files are typically small-size files but provide essential metadata about video streams, including their resolutions, bitrates, etc. In contrast, the actual video and audio content is encoded in the media files. Downloading resources in both file types takes time and is the primary contributor to users' perceived latency. *See* Medium, *Improving ABR Video Performance at Pinterest*, https://medium.com/pinterest-engineering/improving-abr-video-performance-at-pinterest-f0ea47a6d4fc (last visited Apr. 17, 2026) (annotated). (OPENTV_1404_0000800 at OPENTV_1404_0000801.)

No.	Claim Language	Exemplary Infringement Support
		Pinterest engineers describe how manifests are prefetched (i.e., fetched before the new piece of content): Our work aims to reduce latency in manifest loading (highlighted in Figure 1), thereby improving overall video startup performance. Although manifest files are small in size, downloading them adds a non-trivial amount of overhead on video startup. This is particularly noticeable with HLS, where each video contains multiple manifests, resulting in multiple network round trips. As illustrated in Figure 1, after receiving video URLs from the API endpoint, players begin the streaming process by downloading the main manifest playlist from the CDN. The content within the main manifest provides URLs for the rendition-level manifest playlists, prompting players to send subsequent requests to download them. Finally, based on network conditions, players download the appropriate media files to start playback. Acquiring manifests is a prerequisite for downloading media bytes, and reducing this latency leads to faster loading times and a better viewing experience. In the next section, we will share our solution to this problem. *Id.* (annotated). (OPENTV_1404_0000800 at OPENTV_1404_0000802.)
9	The method of claim 1, wherein the receiving the selection, transitioning to the new piece of content, determining whether the	The Accused Pinterest Applications perform the method of claim 1, wherein the receiving the selection, transitioning to the new piece of content, determining whether the new piece of content is of a same content type as the first piece of content, and determining whether to preserve is performed by a player of the digital receiver.

No.	Claim Language	Exemplary Infringement Support
	new piece of content is of a same content type as the first piece of content, and determining whether to preserve is performed by a player of the digital receiver.	For example, the Pinterest Android application includes a player (e.g., an implementation of ExoPlayer version 2.10 or greater) that receives the selection, transitions to the new piece of content, determines whether the new piece of content is of a same content type as the first piece of content, and determines whether to preserve at least a portion of the playback pipeline. **Improved decoder reuse in 2.10** In ExoPlayer 2.10 we've made changes that enable audio decoders to be reused. We've also enabled decoder reuse when re-preparing the player to play a different MediaSource, by allowing renderers to hold decoders when transitioning through the disabled state. These changes are summarized in the diagram below.  Decoder reuse in ExoPlayer 2.10 Olly Woodman, *Improved Decoder Reuse in ExoPlayer*, MEDIUM, https://medium.com/googleexoplayer/improved-decoder-reuse-in-exoplayer-ef4c6d99591d (last visited Nov. 18, 2025). (OPENTV_1404_0000792 at OPENTV_1404_0000795.)

No.	Claim Language	Exemplary Infringement Support
		Making sure your app benefits from decoder reuse To ensure that your app benefits from decoder reuse, it's important that: 1. For use cases where your app switches between playing a number of videos, use a single ExoPlayer instance for all playbacks, rather than instantiating a new instance each time. This is because decoders are only reused within an ExoPlayer instance, not between them. 2. When re-preparing the player with a different video, call `prepare()` with the new `MediaSource` without calling `stop()` first. This is because by default, ExoPlayer will still release decoders when `stop()` is called. Top highlight (OPENTV_1404_0000792 at OPENTV_1404_0000796-OPENTV_1404_0000797.) *See also*: • ExoPlayer, *Library*, GITHUB, at lines 501-514, https://github.com/google/ExoPlayer/blob/92a7bb534a706fe917308af48a4958c99b3bb754/library/core/src/main/java/com/google/android/exoplayer2/video/MediaCodecVideoRenderer.java#L502 (last visited Apr. 17, 2026) (OPENTV_1404_0000910 at OPENTV_1404_0000918); • ExoPlayer, *Library*, GITHUB, at lines 365-387, https://github.com/google/ExoPlayer/blob/92a7bb534a706fe917308af48a4958c99b3bb754/library/core/src/main/java/com/google/android/exoplayer2/audio/MediaCodecAudioRenderer.java#L366 (last visited Apr. 17, 2026) (OPENTV_1404_0000827 at OPENTV_1404_0000833); • AndroidX, *Media*, GITHUB, at lines 1247-1267, https://github.com/androidx/media/blob/1.6.1/libraries/exoplayer/src/main/java/androidx/medi

No.	Claim Language	Exemplary Infringement Support
		a3/exoplayer/video/MediaCodecVideoRenderer.java (last visited Apr. 17, 2026) (OPENTV_1404_0000936 at OPENTV_1404_0000956); • AndroidX, *Media*, GITHUB, at lines 512-533, https://github.com/androidx/media/blob/7ce3aa2619b19009e4799319b4dd694a4a7577df/libraries/exoplayer/src/main/java/androidx/media3/exoplayer/audio/MediaCodecAudioRenderer.java#L513 (last visited Apr. 17, 2026) (OPENTV_1404_0000842 at OPENTV_1404_0000850); and • AndroidX, *Media*, GITHUB, at lines 162-183, https://github.com/androidx/media/blob/7ce3aa2619b19009e4799319b4dd694a4a7577df/libraries/exoplayer/src/main/java/androidx/media3/exoplayer/DecoderReuseEvaluation.java#L37 (last visited Apr. 17, 2026) (OPENTV_1404_0000723 at OPENTV_1404_0000725).
11	A system comprising: one or more hardware processors;	The Accused Pinterest Applications when operating on, for example, a smartphone, tablet, laptop, TV, or desktop computer are a system comprising one or more hardware processors.

No.	Claim Language	Exemplary Infringement Support
		For example, the Google Pixel 6 running the Pinterest Android application uses the Google Tensor processor: Pixel 6 **Operating system**: Android 12 **Display**: • Full-screen 6.4-inch (163 mm)[1] display • 20:9 aspect ratio • FHD+ (1080 x 2400) OLED at 411 ppi • Smooth Display (up to 90 Hz[2]) • Always-on display • At a Glance • Now Playing **Dimensions and Weight[1]**: **Dimensions** • 6.2 height x 2.9 width x 0.4 depth (inches) • 158.6 height x 74.8 width x 8.9 depth (mm) **Weight** • 207 g • 7.30 oz **Battery**: • Beyond 24-hour battery life[3] • Up to 48-hour battery life with Extreme Battery Saver[3] • Minimum 4524 mAh • Typical 4614 mAh[4] **Memory and Storage**: **Memory** • 8 GB LPDDR5 RAM **Storage** • **US, CA, JP, AU, TW:** 128 GB /256 GB UFS 3.1 storage[5] • **UK, DE, ES, IE, IT, FR, SG:** 128 GB UFS 3.1 storage[5] **Processors**: • Google Tensor • Titan M2™ security coprocessor *Pixel Phone Help*, GOOGLE, https://support.google.com/pixelphone/answer/7158570?hl=en#zippy=%2Cpixel (last visited Apr. 23, 2026) (annotated). (OPENTV_1404_0000982 at OPENTV_1404_0000985-OPENTV_1404_0000986.)

No.	Claim Language	Exemplary Infringement Support
11[a]	and a storage device storing instructions that configure the one or more hardware processors to perform operations comprising:	The Accused Pinterest Applications when operating on, for example, a smartphone, tablet, laptop, TV, or desktop computer comprise a storage device storing instructions that configure the one or more hardware processors to perform operations.

No.	Claim Language	Exemplary Infringement Support

		For example, the Google Pixel 6 includes volatile and non-volatile memory for storing the Pinterest Android application including 128 GB or 256 GB UFS 3.1 storage and 8 GB RAM:
		
		Pixel 6
		
		Operating system: Android 12
		
		Display:
		• Full-screen 6.4-inch (163 mm)[1] display
		• 20:9 aspect ratio
		• FHD+ (1080 x 2400) OLED at 411 ppi
		• Smooth Display (up to 90 Hz[2])
		• Always-on display
		  • At a Glance
		  • Now Playing
		
		Dimensions and Weight[1]:
		Dimensions
		• 6.2 height x 2.9 width x 0.4 depth (inches)
		• 158.6 height x 74.8 width x 8.9 depth (mm)
		Weight
		• 207 g
		• 7.30 oz
		
		Battery:
		• Beyond 24-hour battery life[3]
		• Up to 48-hour battery life with Extreme Battery Saver[3]
		• Minimum 4524 mAh
		• Typical 4614 mAh[4]
		
		Memory and Storage:
		Memory
		• 8 GB LPDDR5 RAM
		Storage
		• **US, CA, JP, AU, TW:** 128 GB /256 GB UFS 3.1 storage[5]
		• **UK, DE, ES, IE, IT, FR, SG:** 128 GB UFS 3.1 storage[5]
		
		Pixel Phone Help, GOOGLE, https://support.google.com/pixelphone/answer/7158570?hl=en#zippy=%2Cpixel (last visited Apr. 23, 2026) (annotated). (OPENTV_1404_0000982 at OPENTV_1404_0000985-OPENTV_1404_0000986.)

No.	Claim Language	Exemplary Infringement Support
		The Pinterest Android application stored on the Google Pixel 6 includes instructions that configure the one or more hardware processors to perform the following operations.
11[b]	receiving a selection of a new piece of content for playback during playback of a first piece of content;	The Accused Pinterest Applications include instructions to perform receiving a selection of a new piece of content for playback during playback of a first piece of content *See* claim 1[a].
11[c]	in response to the receiving of the selection, transitioning to the new piece of content;	The Accused Pinterest Applications include instructions to perform in response to the receiving of the selection, transitioning to the new piece of content. *See* claim 1[b].
11[d]	determining whether the new piece of content is of a same content type as the first piece of content; and based on the new piece of content being the same content type as the first piece of content, determining whether to preserve at least a portion of a	The Accused Pinterest Applications include instructions to perform determining whether the new piece of content is of a same content type as the first piece of content; and based on the new piece of content being the same content type as the first piece of content, determining whether to preserve at least a portion of a playback pipeline used to play back the first piece of content. *See* claim 1[c].

No.	Claim Language	Exemplary Infringement Support
	playback pipeline used to play back the first piece of content,	
11[d](i)	the playback pipeline comprising a source element to manage the downloading of content data for the new piece of content, a demultiplexer, an audio decoder, and a video decoder,	The Accused Pinterest Applications include instructions wherein the playback pipeline comprises a source element to manage the downloading of content data for the new piece of content, a demultiplexer, an audio decoder, and a video decoder. *See* claim 1[c](i).
11[d](ii)	the preserving of at least the portion of the playback pipeline comprising preserving one or more of the source element, the demultiplexer, the audio decoder, or the video decoder used to obtain the first piece of content for the new piece of content.	The Accused Pinterest Applications include instructions to perform the preserving of at least the portion of the playback pipeline of Pinterest's video Pin system comprises preserving one or more of the source element, the demultiplexer, the audio decoder, or the video decoder used to obtain the first piece of content for the new piece of content. *See* claim 1[c](ii).

No.	Claim Language	Exemplary Infringement Support
12	The system of claim 11, wherein the determining whether to preserve comprises: detecting codec information for the new piece of content, the codec information indicating types of decoders needed for the new piece of content; and	The Accused Pinterest Applications include instructions to perform the system of claim 11, wherein the determining whether to preserve comprises: detecting codec information for the new piece of content, the codec information indicating types of decoders needed for the new piece of content. *See* claim 2.
12[a]	determining whether the audio decoder and the video decoder in the playback pipeline are of the types indicated by the codec information.	The Accused Pinterest Applications include instructions to perform determining whether the audio decoder and the video decoder in the playback pipeline are of the types indicated by the codec information. *See* claim 2[a].
15	The system of claim 12, wherein the detecting the codec information for the new piece of content comprises examining prefetched	The Accused Pinterest Applications include instructions to perform the system of claim 12, wherein the detecting the codec information for the new piece of content comprises examining prefetched metadata for the new piece of content. *See* claim 5.

No.	Claim Language	Exemplary Infringement Support
	metadata for the new piece of content.	
20	A machine-storage medium storing instructions	The Accused Pinterest Applications when operating on, for example, a smartphone, tablet, laptop, TV, or desktop computer comprise a machine-storage medium storing instructions.

No.	Claim Language	Exemplary Infringement Support
		For example, the Google Pixel 6 running the Pinterest Android application uses a machine-storage medium, including volatile and non-volatile memory for storing the Pinterest Android application including 128 GB or 256 GB UFS 3.1 storage and 8 GB RAM: Pixel 6 **Operating system**: Android 12 **Display**: • Full-screen 6.4-inch (163 mm)[1] display • 20:9 aspect ratio • FHD+ (1080 x 2400) OLED at 411 ppi • Smooth Display (up to 90 Hz[2]) • Always-on display • At a Glance • Now Playing **Dimensions and Weight[1]**: **Dimensions** • 6.2 height x 2.9 width x 0.4 depth (inches) • 158.6 height x 74.8 width x 8.9 depth (mm) **Weight** • 207 g • 7.30 oz **Battery**: • Beyond 24-hour battery life[3] • Up to 48-hour battery life with Extreme Battery Saver[3] • Minimum 4524 mAh • Typical 4614 mAh[4] **Memory and Storage**: **Memory** • 8 GB LPDDR5 RAM **Storage** • **US, CA, JP, AU, TW:** 128 GB /256 GB UFS 3.1 storage[5] • **UK, DE, ES, IE, IT, FR, SG:** 128 GB UFS 3.1 storage[5] *Pixel Phone Help*, GOOGLE, https://support.google.com/pixelphone/answer/7158570?hl=en#zippy=%2Cpixel (last visited Apr. 28, 2026) (annotated). (OPENTV_1404_0000982 at OPENTV_1404_0000985-OPENTV_1404_0000986.)

No.	Claim Language	Exemplary Infringement Support
		The Pinterest Android application stored on the Google Pixel 6 includes instructions that are stored on the aforementioned machine-storage medium. *See* claim 11[a].
20[a]	that when executed by one or more hardware processors of a machine, cause the machine to perform operations comprising:	The Accused Pinterest Applications when operating on, for example, a smartphone, tablet, laptop, TV, or desktop computer include instructions that when executed by one or more hardware processors of a machine, cause the machine to perform operations.

No.	Claim Language	Exemplary Infringement Support
		For example, the Google Pixel 6 running the Pinterest Android application uses the Google Tensor processor: Pixel 6 **Operating system**: Android 12 **Display**: • Full-screen 6.4-inch (163 mm)[1] display • 20:9 aspect ratio • FHD+ (1080 x 2400) OLED at 411 ppi • Smooth Display (up to 90 Hz[2]) • Always-on display • At a Glance • Now Playing **Dimensions and Weight[1]**: **Dimensions** • 6.2 height x 2.9 width x 0.4 depth (inches) • 158.6 height x 74.8 width x 8.9 depth (mm) **Weight** • 207 g • 7.30 oz **Battery**: • Beyond 24-hour battery life[3] • Up to 48-hour battery life with Extreme Battery Saver[3] • Minimum 4524 mAh • Typical 4614 mAh[4] **Memory and Storage**: **Memory** • 8 GB LPDDR5 RAM **Storage** • **US, CA, JP, AU, TW:** 128 GB /256 GB UFS 3.1 storage[5] • **UK, DE, ES, IE, IT, FR, SG:** 128 GB UFS 3.1 storage[5] **Processors**: • Google Tensor • Titan M2™ security coprocessor *Pixel Phone Help*, GOOGLE, https://support.google.com/pixelphone/answer/7158570?hl=en#zippy=%2Cpixel (last visited Apr. 28, 2026) (annotated). (OPENTV_1404_0000982 at OPENTV_1404_0000985-OPENTV_1404_0000986.)

No.	Claim Language	Exemplary Infringement Support
		The Pinterest Android application stored on the Google Pixel 6 example includes instructions, executed by the aforementioned hardware processor, that cause the machine to perform the following operations. *See* claim 11; claim 11[a].
20[b]	receiving a selection of a new piece of content for playback during playback of a first piece of content;	The Accused Pinterest Applications include instructions to perform receiving a selection of a new piece of content for playback during playback of a first piece of content. *See* claim 1[a]; claim 11[b].
20[c]	in response to the receiving of the selection, transitioning to the new piece of content;	The Accused Pinterest Applications include instructions to perform in response to the receiving of the selection, transitioning to the new piece of content. *See* claim 1[b]; claim 11[c].
20[d]	determining whether the new piece of content is of a same content type as the first piece of content; and based on the new piece of content being the same content type as the first piece of content, determining	The Accused Pinterest Applications include instructions to perform determining whether the new piece of content is of a same content type as the first piece of content; and based on the new piece of content being the same content type as the first piece of content, determining whether to preserve at least a portion of a playback pipeline used to play back the first piece of content. *See* claim 1[c]; claim 11[d].

No.	Claim Language	Exemplary Infringement Support
	whether to preserve at least a portion of a playback pipeline used to play back the first piece of content,	
20[d](i)	the playback pipeline comprising a source element to manage the downloading of content data for the new piece of content, a demultiplexer, an audio decoder, and a video decoder,	The Accused Pinterest Applications include instructions wherein the playback pipeline comprises a source element to manage the downloading of content data for the new piece of content, a demultiplexer, an audio decoder, and a video decoder. *See* claim 1[c](i); claim 11[d](i).
20[d](ii)	the preserving of at least the portion of the playback pipeline comprising preserving one or more of the source element, the demultiplexer, the audio decoder, or the video decoder used to obtain the first piece of content	The Accused Pinterest Applications include instructions to perform the preserving of at least the portion of the playback pipeline of Pinterest's video Pin system comprises preserving one or more of the source element, the demultiplexer, the audio decoder, or the video decoder used to obtain the first piece of content for the new piece of content. *See* claim 1[c](ii); claim 11[d](ii).

No.	Claim Language	Exemplary Infringement Support
	for the new piece of content.	