DION особенности

Система ВКС

Для корпоративной коммуникации

Ограничения доступа

Безопасность

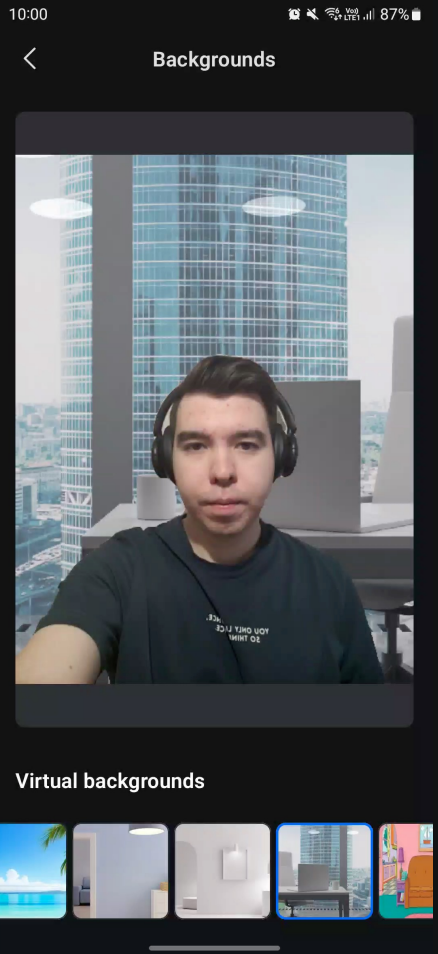
Много режимов

И много одновременных пользователей

Высокая нагрузка

Важна производительность


10:00
87%
Backgrounds
Virtual backgrounds
None
Blur


10:00
87%
Backgrounds
Virtual backgrounds

DIXON

Bitmap

Render Script

DIXON

Allocation

+100mb APK

CPU image processing

Deprecated


Solutions
Framework

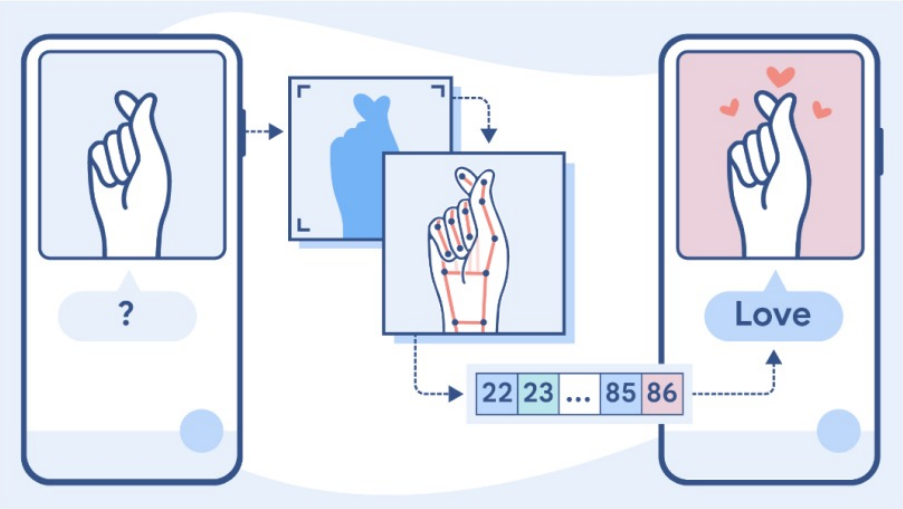
		Framework	Solutions	
Source code	Open source	Open source	Open source	Proprietary
Application area	Any scenarios	Any scenarios	Individual cases	Individual cases
Customization	Very high	High	Mid	Not customizable
Complexity	Very high	High	Mid	Mid

MediaPipe

MediaPipe Solutions

DXON

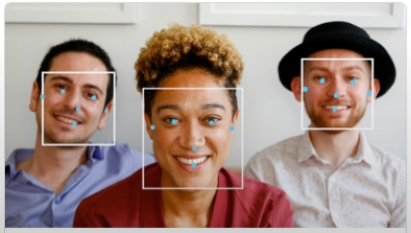
Complex on-device ML, simplified



We've abstracted away the complexities of making on-device ML customizable, production-ready, and accessible across platforms.

- Lightweight ML models all while preserving accuracy
- Domain-specific processing including vision, text, and audio
- End-to-end acceleration across CPU and GPU
- Complex pipeline graph with multiple models and states
- Cross-platform deployment to Android, iOS, web, bare metal

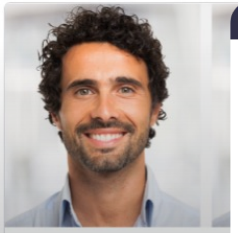
MediaPipe Solutions



Face Detection

Identify faces in images and video.

See demo | Code examples



Face Landmark Detect

Identify facial features for visual ef

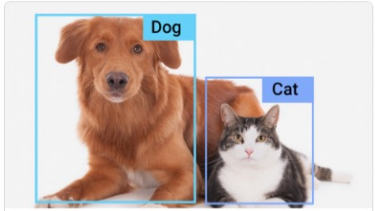
See demo | Code examp

Image Embedding

Embed images into feature vectors.

See demo | Code examples

Vision examples



Object Detection

Track and label objects in images and video.

See demo | Code examples

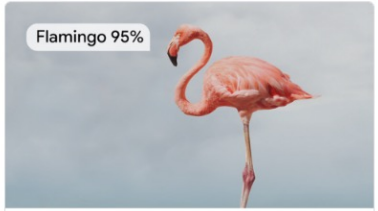


Image Classification

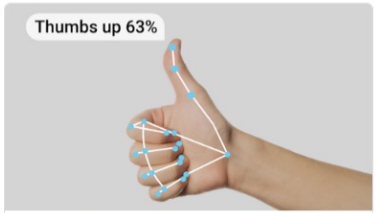
Identify content in images and video.

See demo | Code examples

Hand Landmark

Identify and track hands and fingers.

See demo | Code examples



Hand Gesture Recognition

Identify and recognize hand gestures.

See demo | Code examples

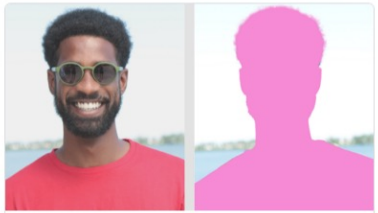


Image Segmentation

Locate objects and create image masks with labels.

See demo | Code examples

Interactive Segmentation

Select objects interactively and create image masks.

See demo | Code examples

MediaPipe Solutions

DXON

Text examples

Input: "Great movie with a classic plot. I will recommend this to everyone."

Output:

Text Classification

Classify text into relevant tags.

See demo | Code examples

I read a research paper on dwarf galaxies and black holes.

Text Embedding

Embed texts into feature

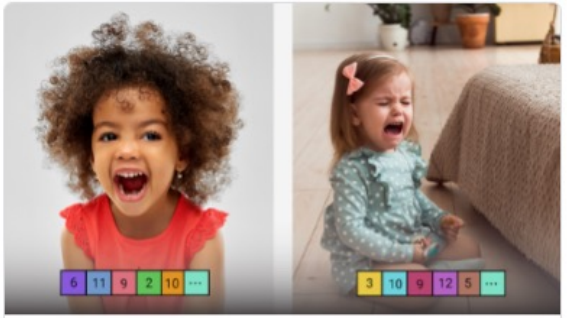
See demo

Audio examples

Audio Classification

Classify sounds into relevant tags.

See demo | Code examples

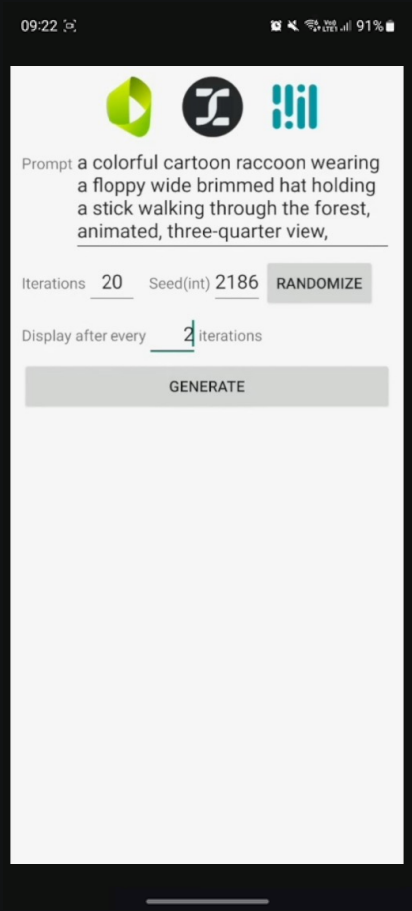


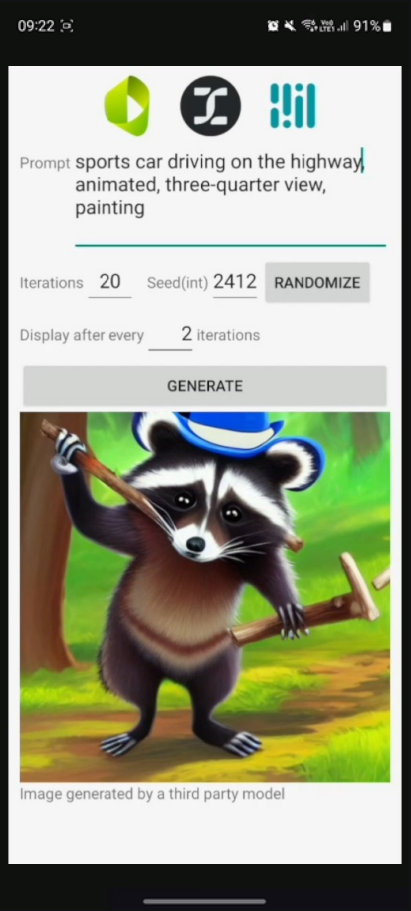
Audio Embedding

Embed audio into feature vectors.

Coming soon

Image Generation

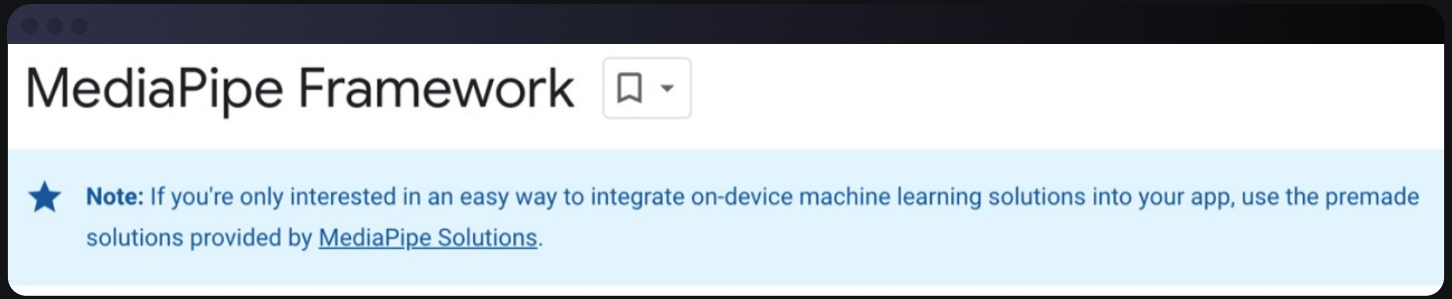




MediaPipe Framework

DION

MediaPipe Framework is the low-level component used to build efficient on-device machine learning pipelines, similar to the premade MediaPipe Solutions



C ++

Tensor Flow Lite

Java

ObjC

Objective-C

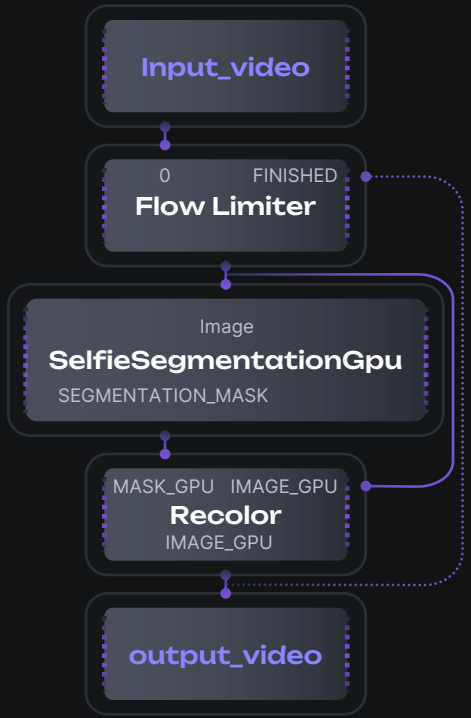
Bazel

Framework concepts

DION

Graph

*.pbtxt –> *.binarypb



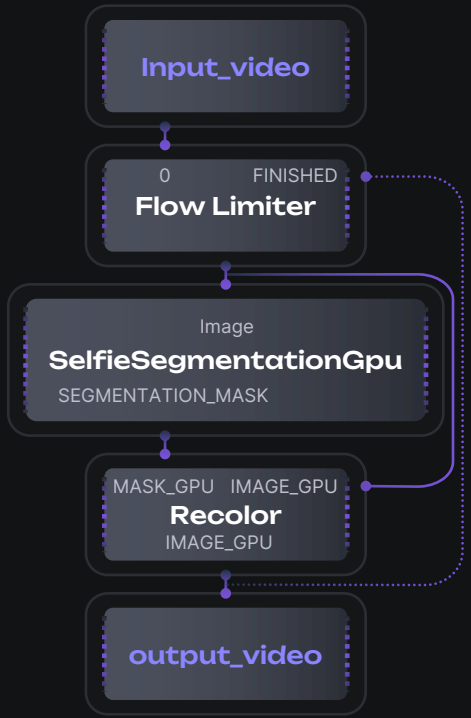
Framework concepts

DION

Graph

Streams

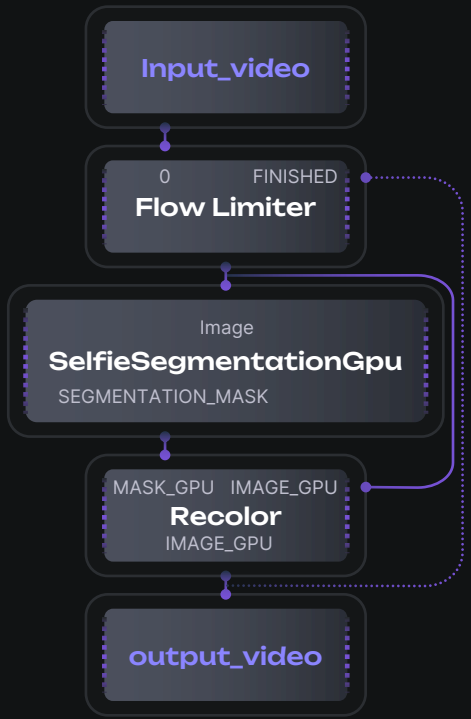
*.pbtxt –> *.binarypb



Framework concepts

```
# GPU buffer. (GpuBuffer)
input_stream: "input_video"

# Output image with rendered results. (GpuBuffer)
output_stream: "output_video"
```

*.pbtxt –> *.binarypb



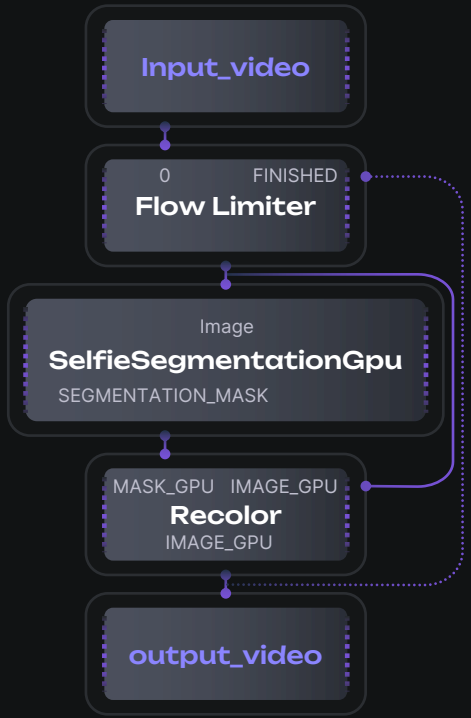
Framework concepts

DIXON

Graph

Streams

*.pbtxt –> *.binarypb



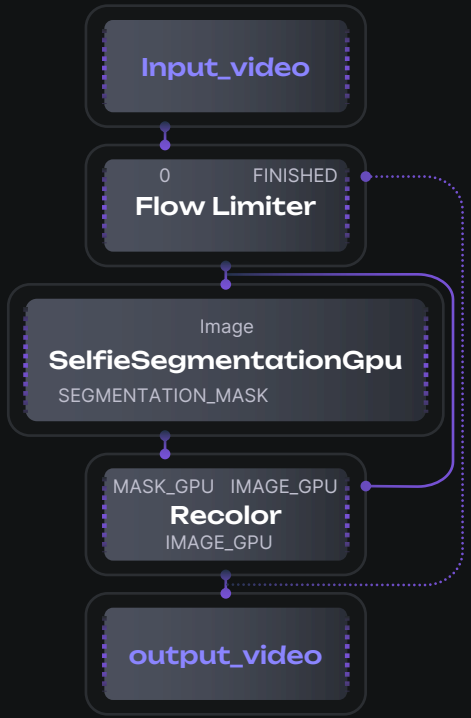
Framework concepts

Graph

Streams

Nodes/Calculators

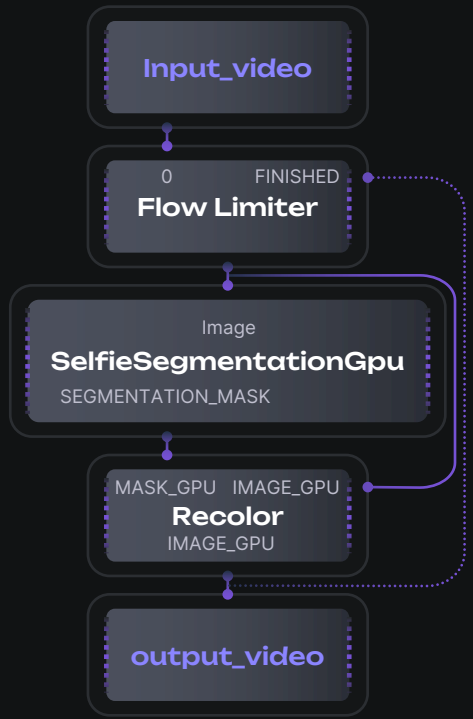
*.pbtxt –> *.binarypb



Framework concepts

```
# Colors the selfie segmentation with the color specified in the option.
node {
  calculator: "RecolorCalculator"
  input_stream: "IMAGE_GPU:throttled_input_video"
  input_stream: "MASK_GPU:segmentation_mask"
  output_stream: "IMAGE_GPU:output_video"
  node_options: {
    [type.googleapis.com/mediapipe.RecolorCalculatorOptions] {
      color { r: 0 g: 0 b: 255 }
      mask_channel: RED
      invert_mask: true
      adjust_with_luminance: false
    }
  }
}
```

*.pbtxt –> *.binarypb

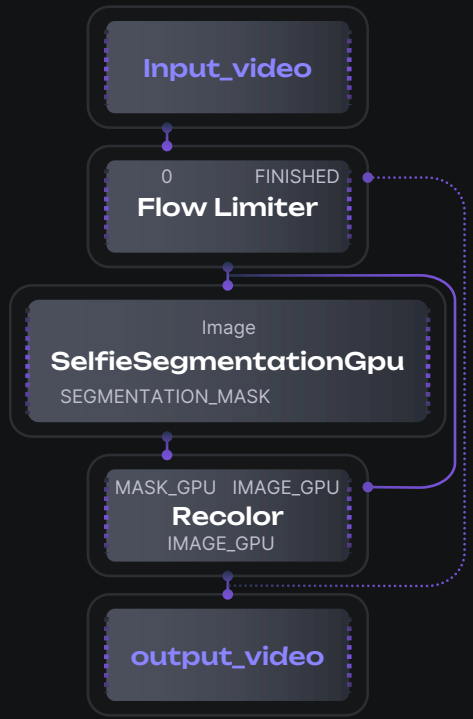


Framework concepts

DION

```
# Subgraph that performs selfie segmentation.
node {
  calculator: "SelfieSegmentationGpu"
  input_stream: "IMAGE:throttled_input_video"
  output_stream: "SEGMENTATION_MASK:segmentation_mask"
}
```

*.pbtxt –> *.binarypb



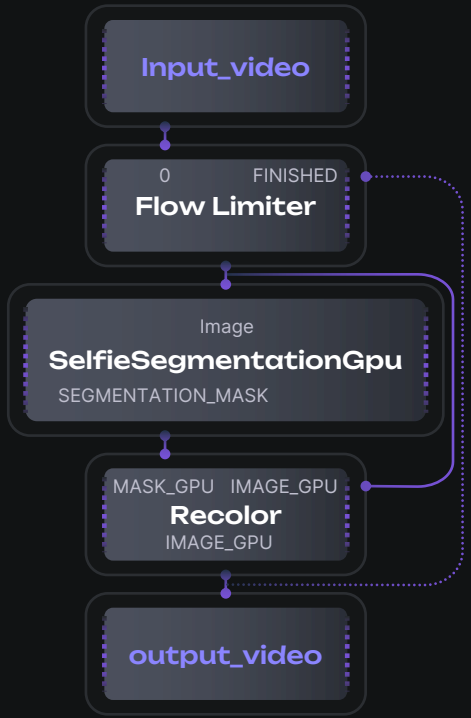
Framework concepts

Graph

Streams

Nodes/Calculators

*.pbtxt –> *.binarypb



Framework concepts

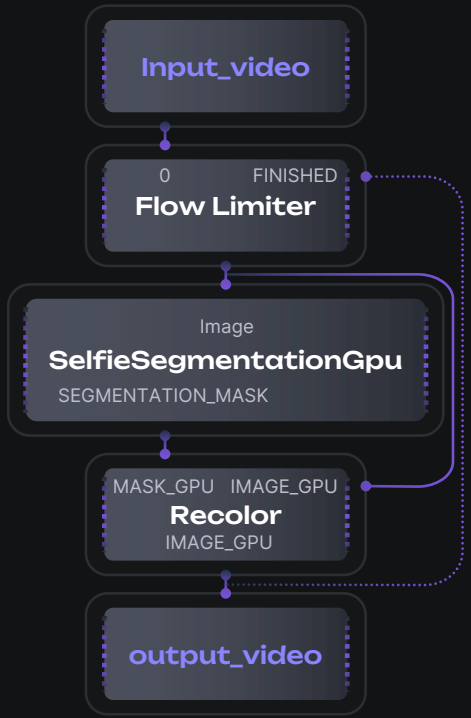
Graph

Streams

Nodes/Calculators

Packet

*.pbtxt –> *.binarypb




09:59
88%
Selfie Segmentation

GPU usage

GPU usage

DICON

Realtime rendering

Rendering + effects + analysis

GPU usage

DION

Flexible Placement

Efficient Data Transfer

Optimized CPU-GPU Transfer

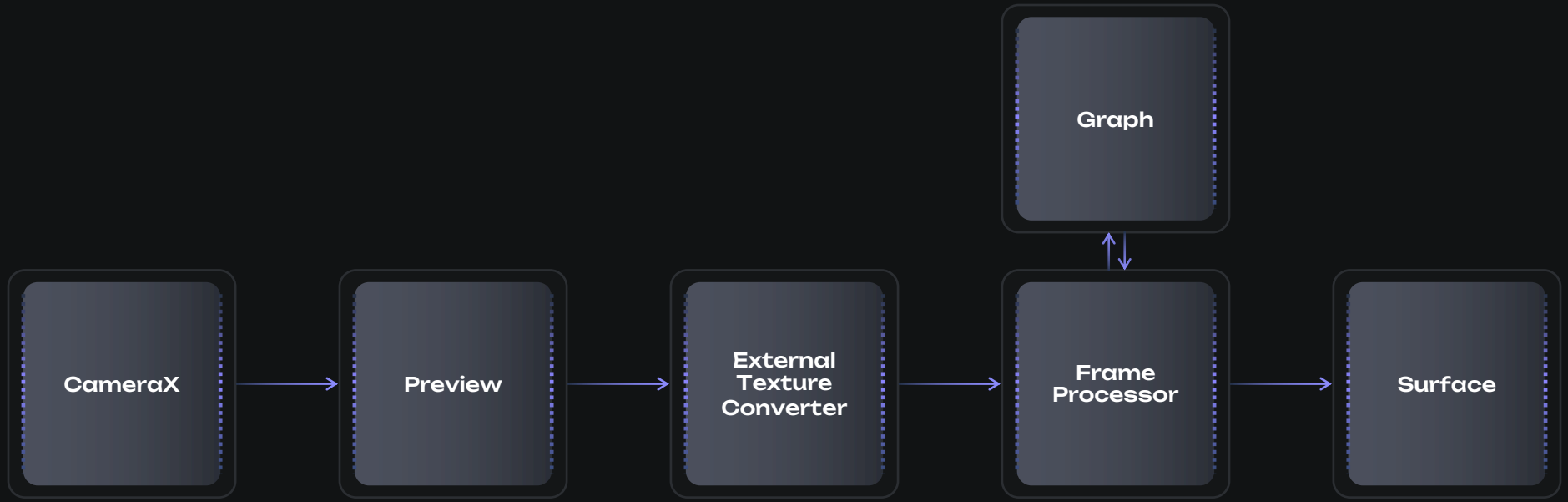
Platform-Specific Implementation

Combining CPU and GPU Operations

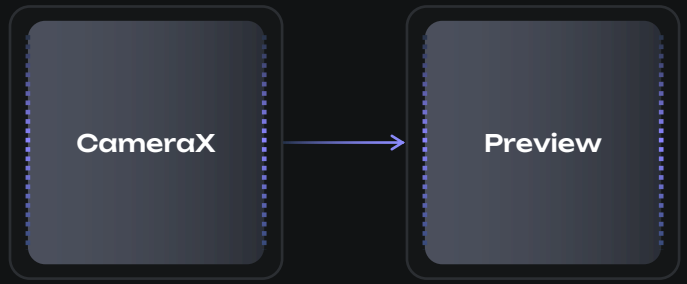
DION
MediaPipe

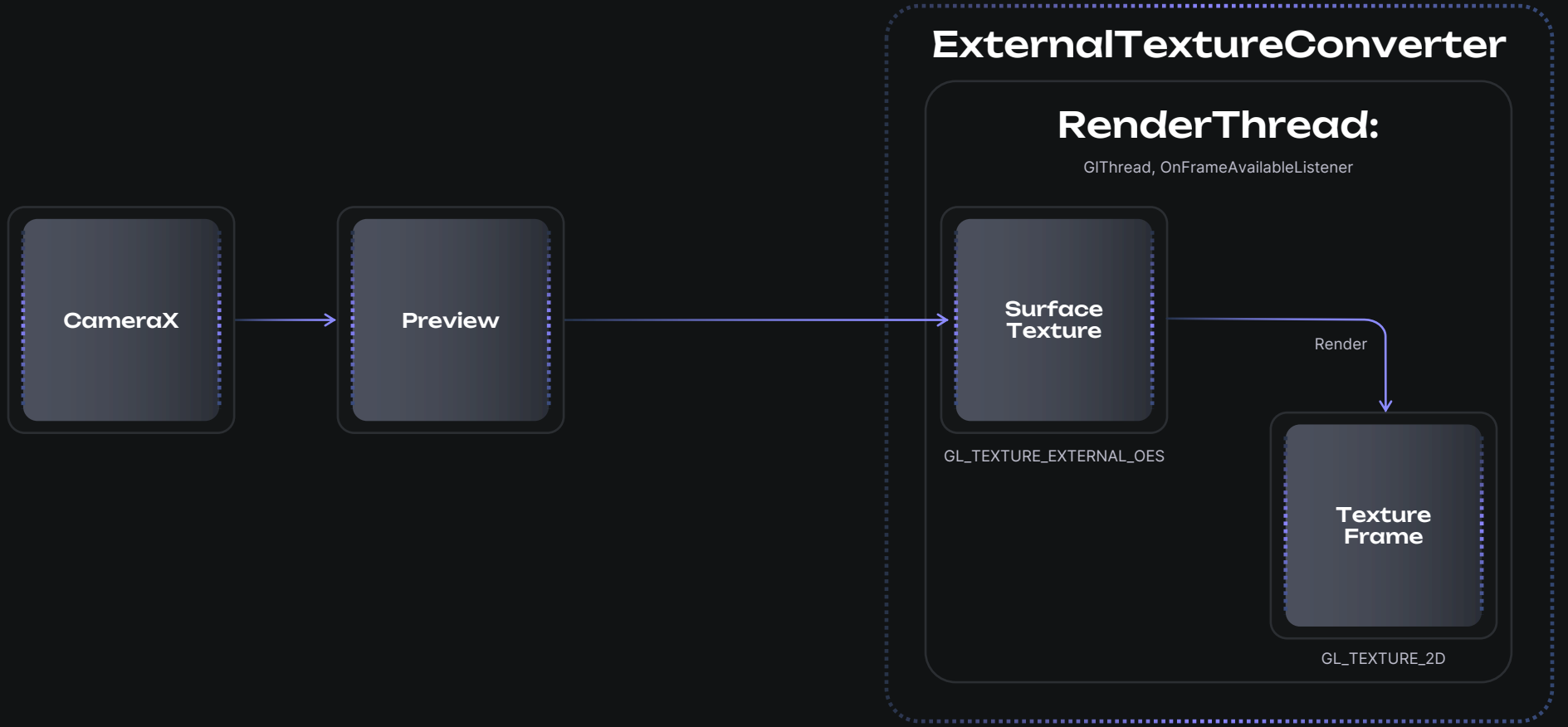
DION

```
interface VideoCapturer {
        public void initialize(
                SurfaceTextureHelper surfaceTextureHelper,
                Context applicationContext,
                CapturerObserver capturerObserver
        )
}
```


Graph
CameraX
Preview
External Texture Converter
Frame Processor
Surface

```
preview.setSurfaceProvider(
    executor
) { request: SurfaceRequest ->
    val surface = Surface(surfaceTexture)
    request.provideSurface(surface, executor) { surface.release() }
}
```




ExternalTextureConverter
RenderThread:
GlThread, OnFrameAvailableListener
CameraX
Preview
Surface Texture
GL_TEXTURE_EXTERNAL_OES
Render
Texture Frame
GL_TEXTURE_2D

TextureFrame

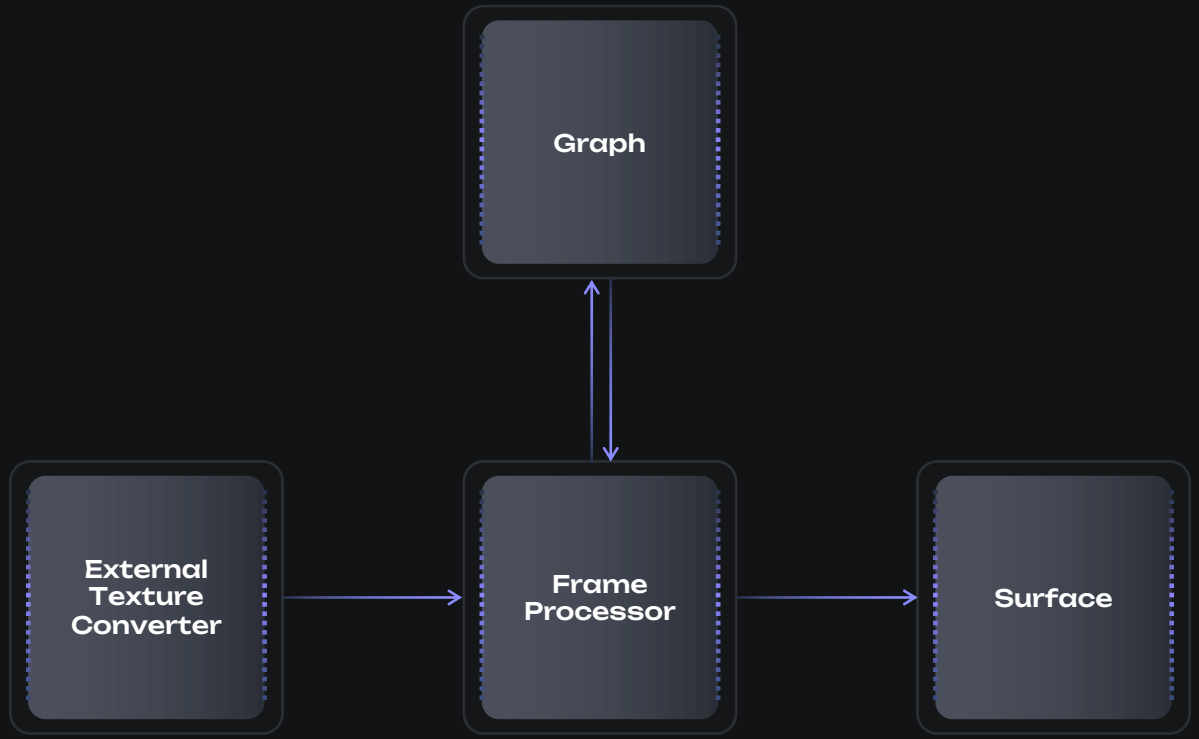
DION

```
interface TextureFrame{
  int getTextureName();
  int getWidth();
  int getHeight();
  long getTimestamp();
  void release();
}
```

```
interface TextureFrameConsumer {
  void onNewFrame(TextureFrame frame);
}
```

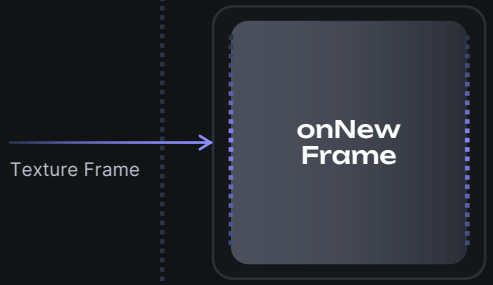
```
interface TextureFrameProducer {
  void setConsumer(TextureFrameConsumer next);
}
```

```
interface TextureFrameProcessor extends TextureFrameProducer, TextureFrameConsumer {}
```

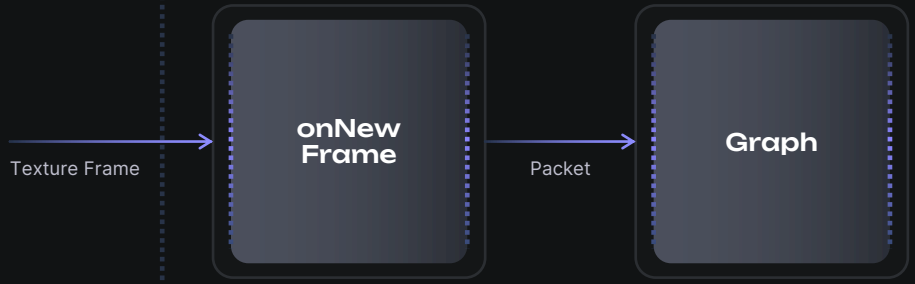

Graph
External Texture Converter
Frame Processor
Surface

```
public class FrameProcessor implements TextureFrameProcessor {

  private Graph mediapipeGraph;
  private AndroidPacketCreator packetCreator;
  private String videoInputStream;
  private String videoOutputStream;

  public FrameProcessor(
      Context context,
      long parentNativeContext,
      String graphName,
      String inputStream,
      String outputStream) {
    mediapipeGraph = new Graph();
    mediapipeGraph.loadBinaryGraph(graphName); // *.binarypb

    packetCreator = new AndroidPacketCreator(mediapipeGraph);
    videoInputStream = inputStream;
    videoOutputStream = outputStream;
    mediapipeGraph.setParentGlContext(parentNativeContext);
  }
}
```

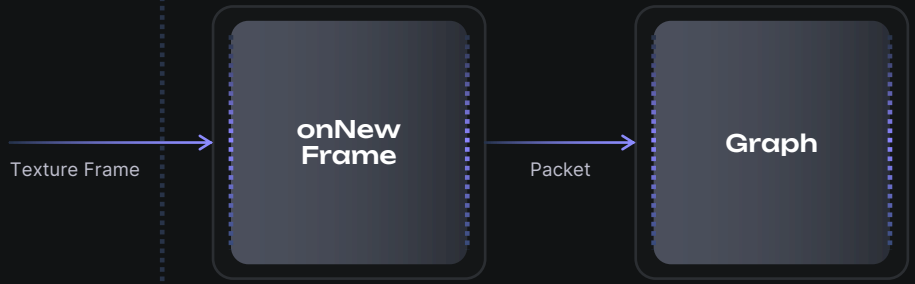
Frame Processor



Frame Processor

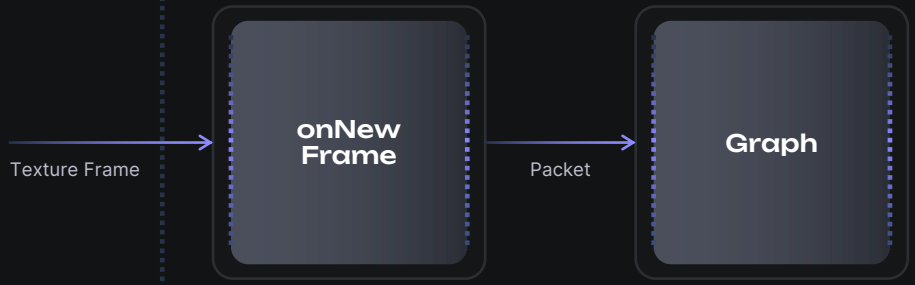


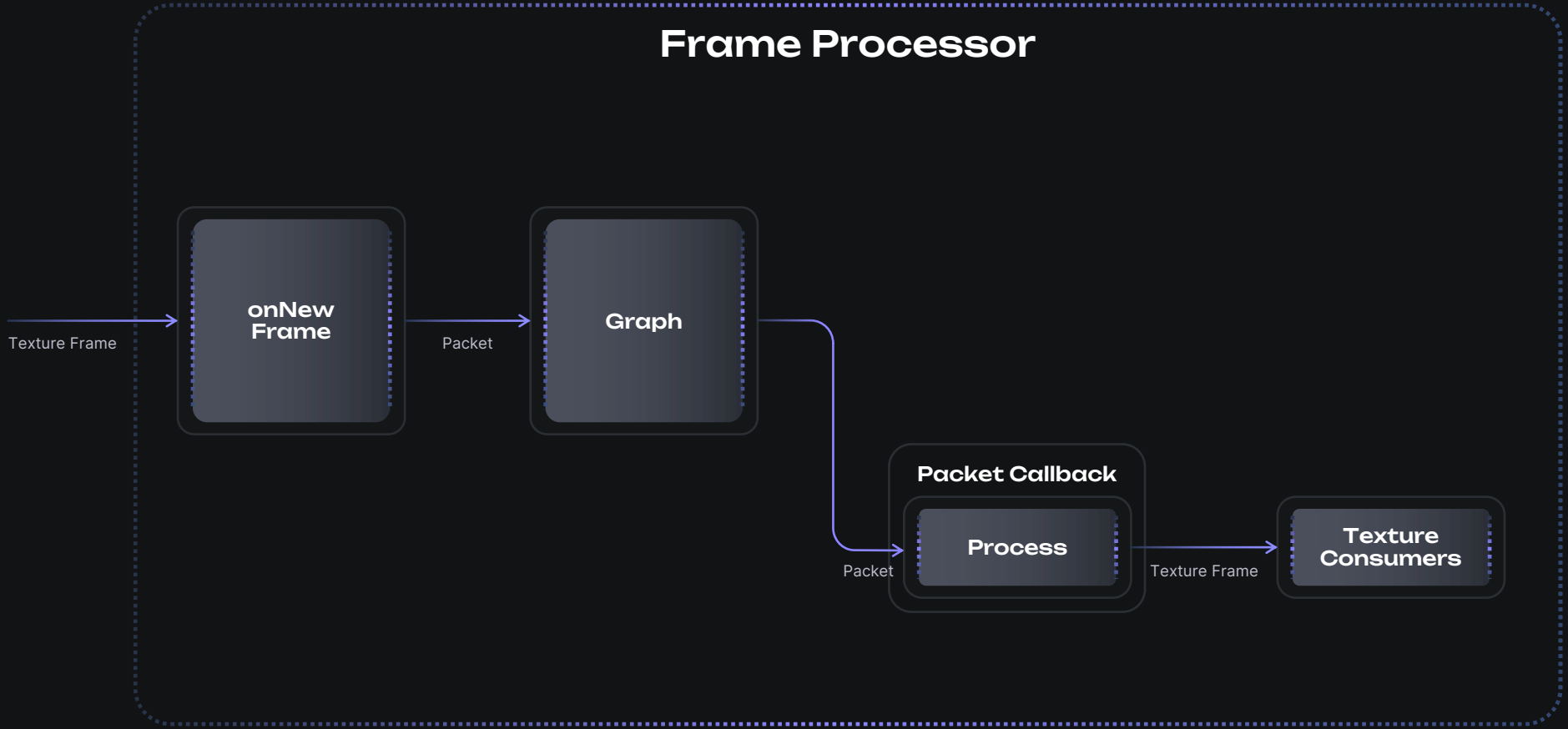
Frame Processor



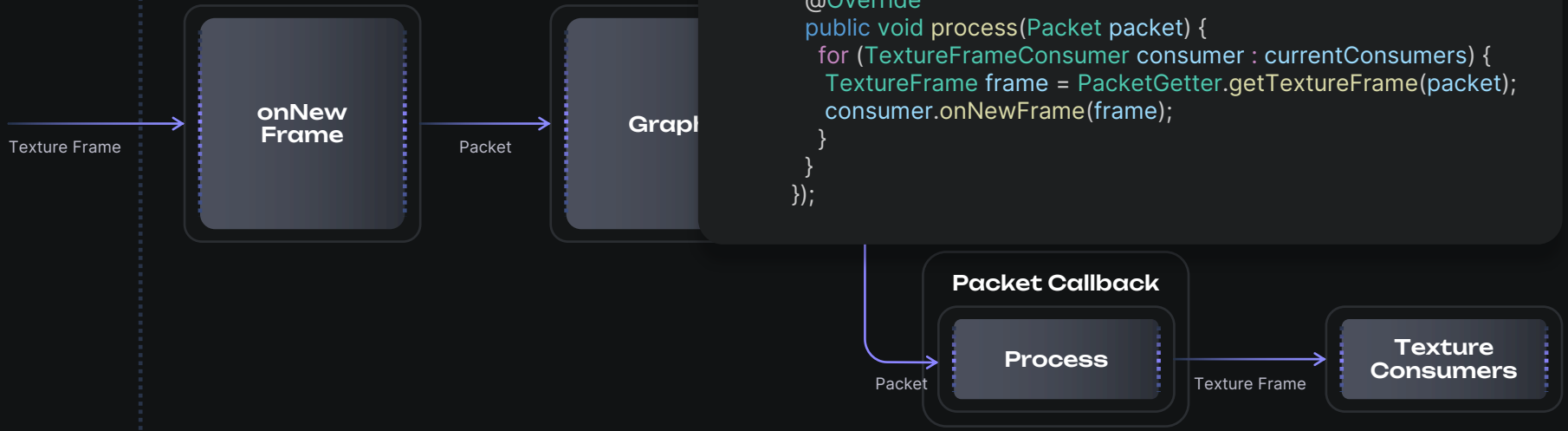

```
@Override
public void onNewFrame(TextureFrame frame) {
    Packet imagePacket = packetCreator.createGpuBuffer(frame);
    mediapipeGraph.addConsumablePacketToInputStream(
        videoInputStream, imagePacket, timestamp
    );
}
```

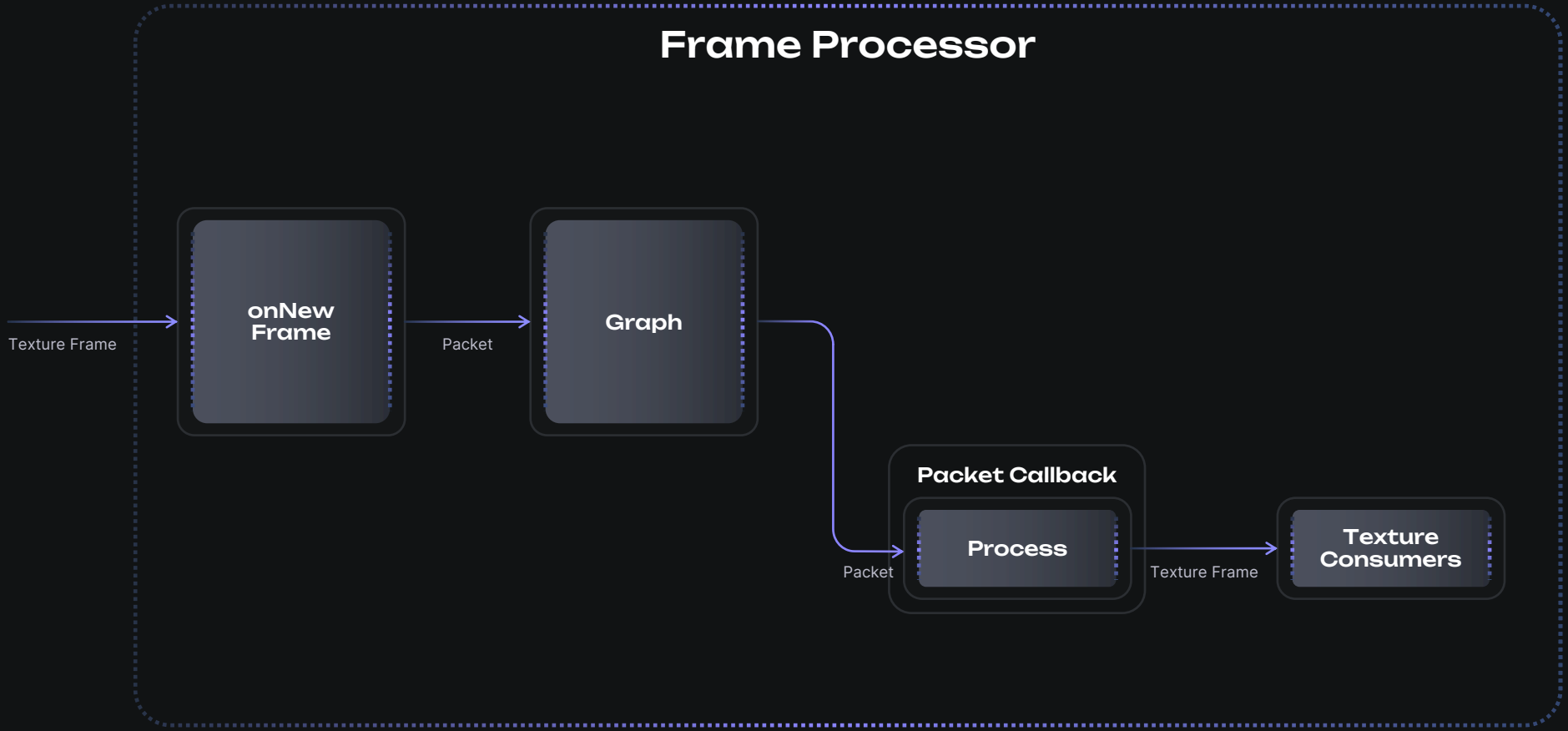
Frame Processor

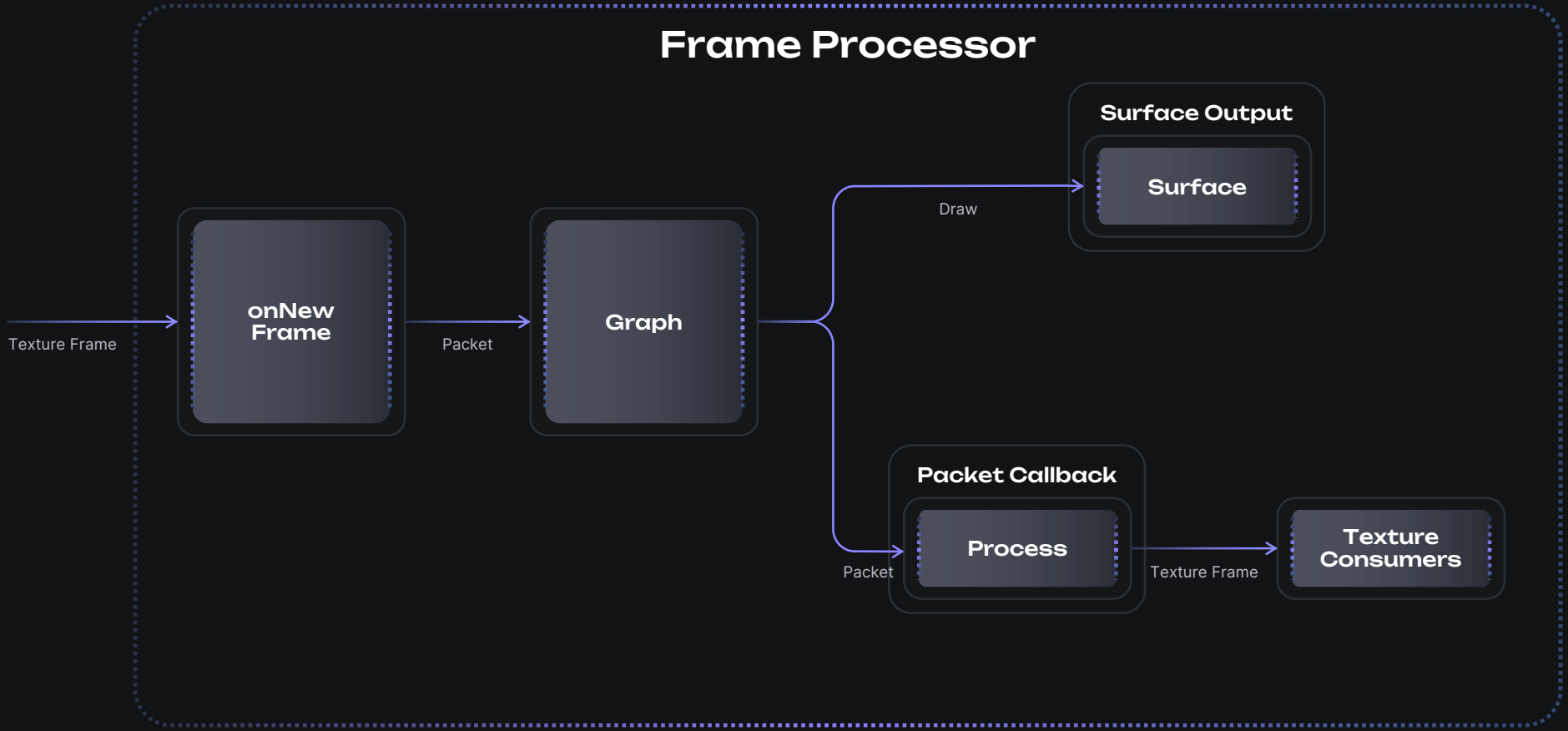



Frame Processor
Texture Frame
onNew Frame
Packet
Graph
Packet
Packet Callback
Process
Texture Frame
Texture Consumers

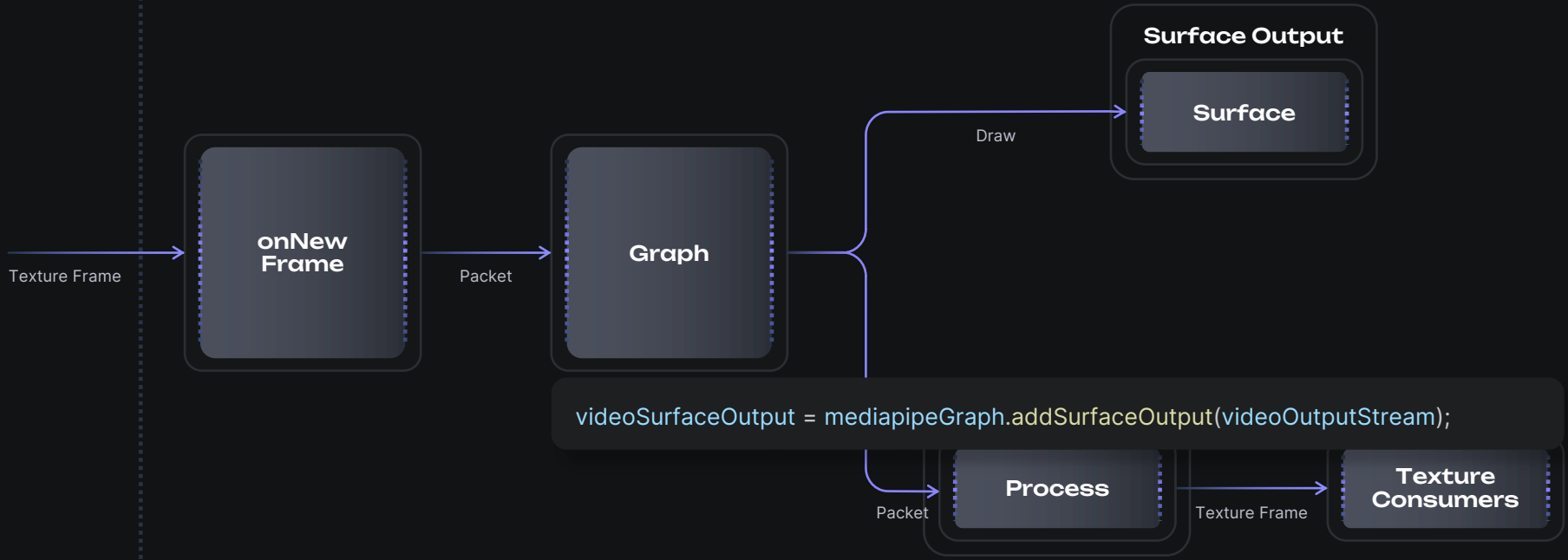
Frame Processor

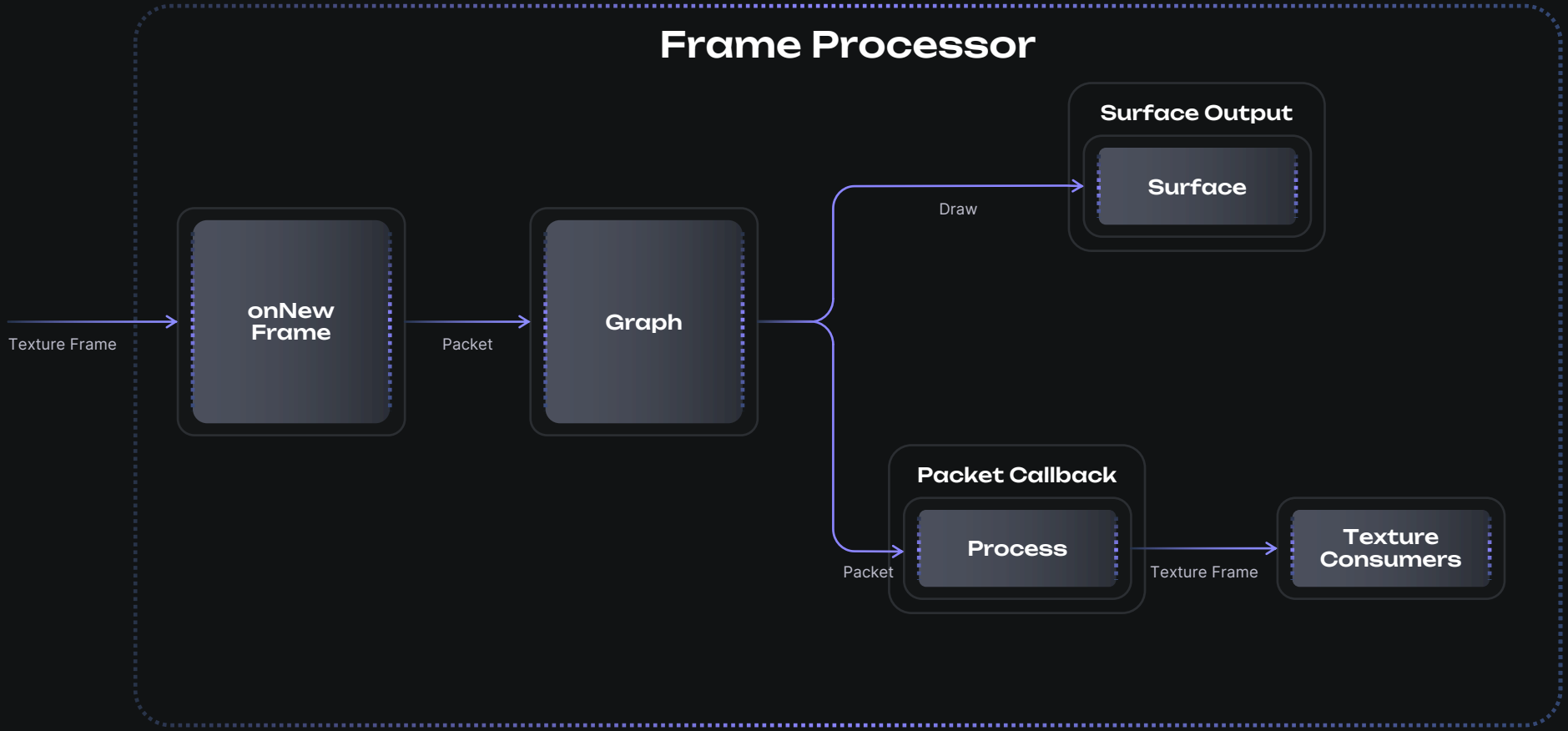


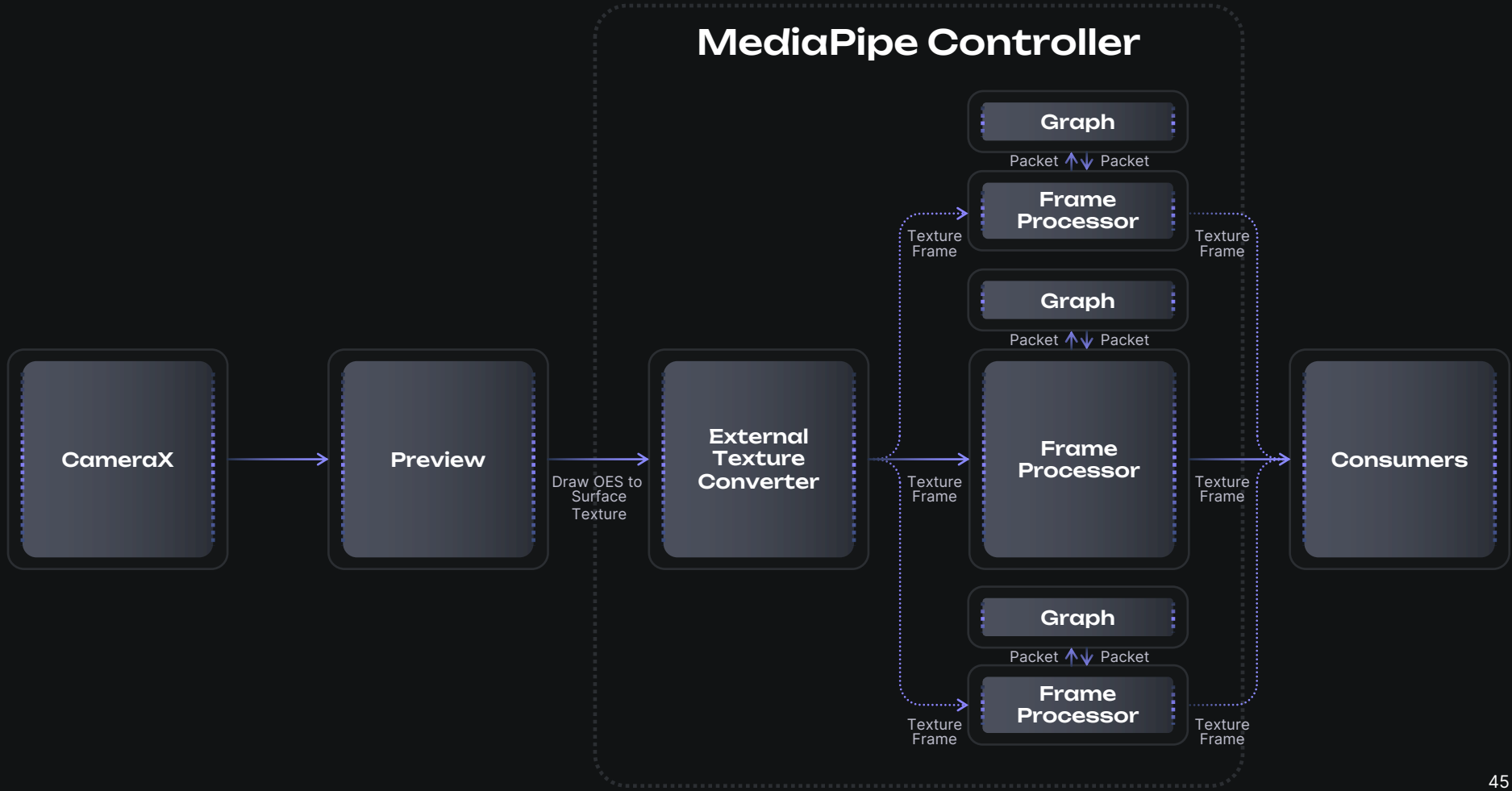

Frame Processor
Texture Frame
onNew Frame
Packet
Graph
Packet
Packet Callback
Process
Texture Frame
Texture Consumers

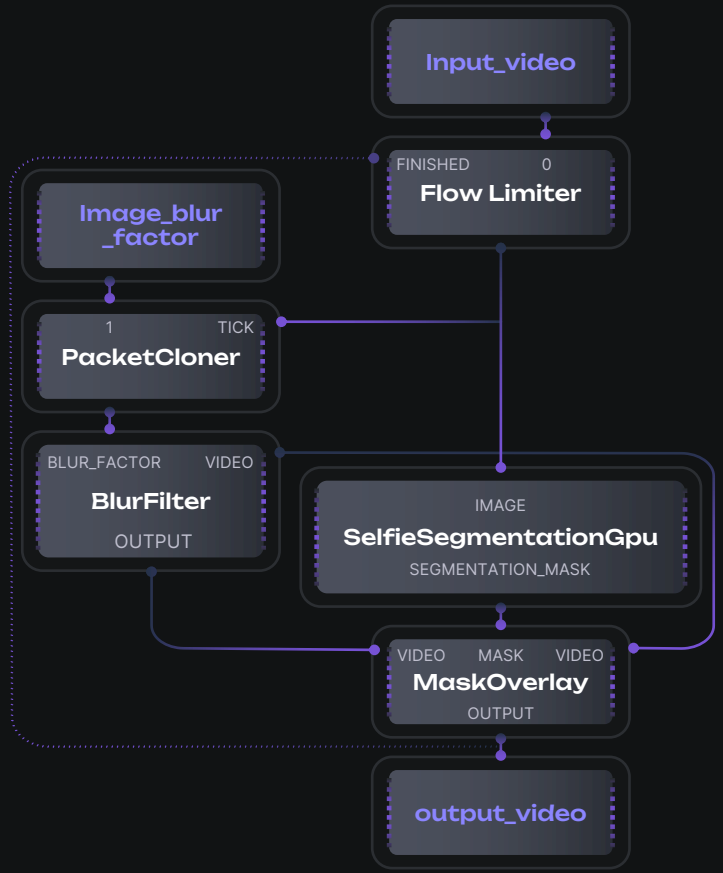

Frame Processor
Surface Output
Surface
Draw
Texture Frame
onNew Frame
Packet
Graph
Packet Callback
Process
Packet
Texture Frame
Texture Consumers

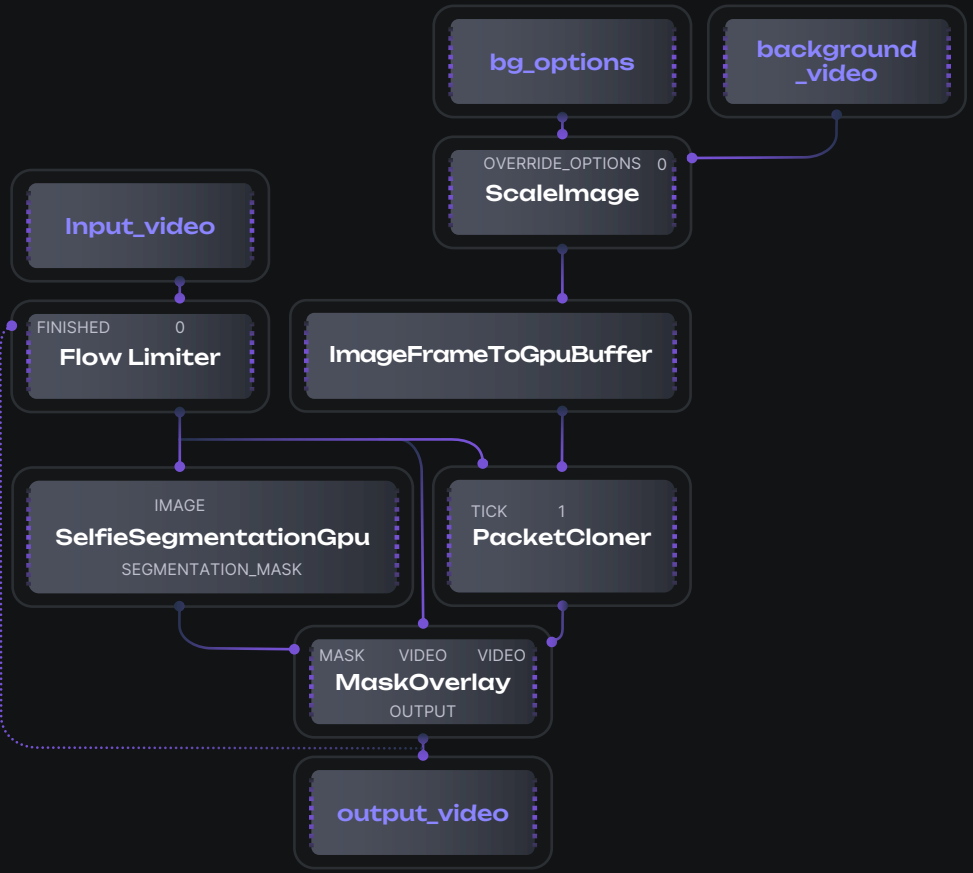
Frame Processor

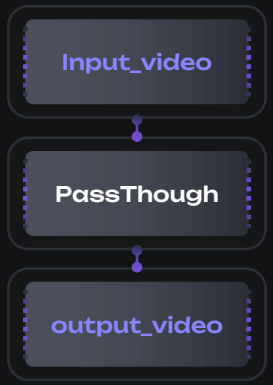



Frame Processor
Surface Output
Surface
Draw
Texture Frame
onNew Frame
Packet
Graph
Packet Callback
Process
Packet
Texture Frame
Texture Consumers


MediaPipe Controller
CameraX
Preview
Draw OES to Surface Texture
External Texture Converter
Graph
Packet
Packet
Frame Processor
Texture Frame
Texture Frame
Graph
Packet
Packet
Frame Processor
Texture Frame
Texture Frame
Graph
Packet
Packet
Frame Processor
Texture Frame
Texture Frame
Consumers

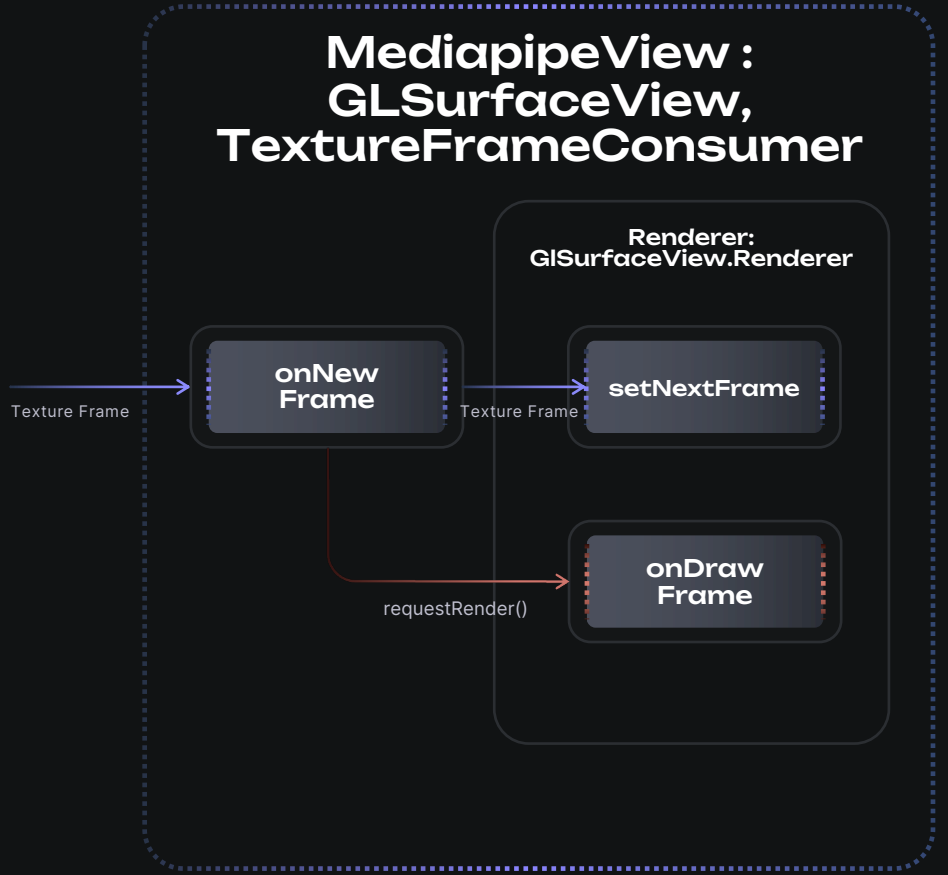

Input_video
FINISHED
0
Flow Limiter
Image_blur
_factor
1
TICK
PacketCloner
BLUR_FACTOR
VIDEO
BlurFilter
OUTPUT
IMAGE
SelfieSegmentationGpu
SEGMENTATION_MASK
VIDEO
MASK
VIDEO
MaskOverlay
OUTPUT
output_video


bg_options
background_video
OVERRIDE_OPTIONS 0
ScaleImage
Input_video
FINISHED 0
Flow Limiter
ImageFrameToGpuBuffer
IMAGE
SelfieSegmentationGpu
SEGMENTATION_MASK
TICK 1
PacketCloner
MASK VIDEO VIDEO
MaskOverlay
OUTPUT
output_video

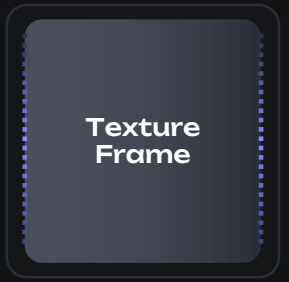

Input_video
PassThough
output_video

Common View

```
class MediapipeVideoView(
    context: Context,
    attrs: AttributeSet? = null,
) :
    GLSurfaceView(context, attrs),
    TextureFrameConsumer {
      private val renderer =
          GlSurfaceViewRenderer()

      init {
          setEGLContextClientVersion(...)
          setEGLContextFactory(...)
          setRenderer(renderer)
          renderMode = RENDERMODE_WHEN_DIRTY
      }

      override fun onNewFrame(frame: TextureFrame) {
          renderer.setFrameSize(frame.width, frame.height)
          renderer.setNextFrame(frame)
          requestRender()
      }
}
```



WebRTC Render



WebRTC Render

WebRTC Render

DJON

```
class RtcRenderer: TextureFrameConsumer {
    private var source: org.webrtc.VideoSource?

    override fun onNewFrame(frame: TextureFrame) {
        var videoFrame: VideoFrame? = null
        try {
            val texture = org.webrtc.TextureBufferImpl(
                frame.width,
                frame.height,
                VideoFrame.TextureBuffer.Type.RGB,
                frame.textureName,
                ...
            )
            videoFrame = org.webrtc.VideoFrame(texture,...)
            source?.capturerObserver?.onFrameCaptured(videoFrame)
        } catch (e: Exception) {
            // Handle error
        } finally {
            frame.release()
            videoFrame?.release()
        }
    }

}
```

WebRTC Render

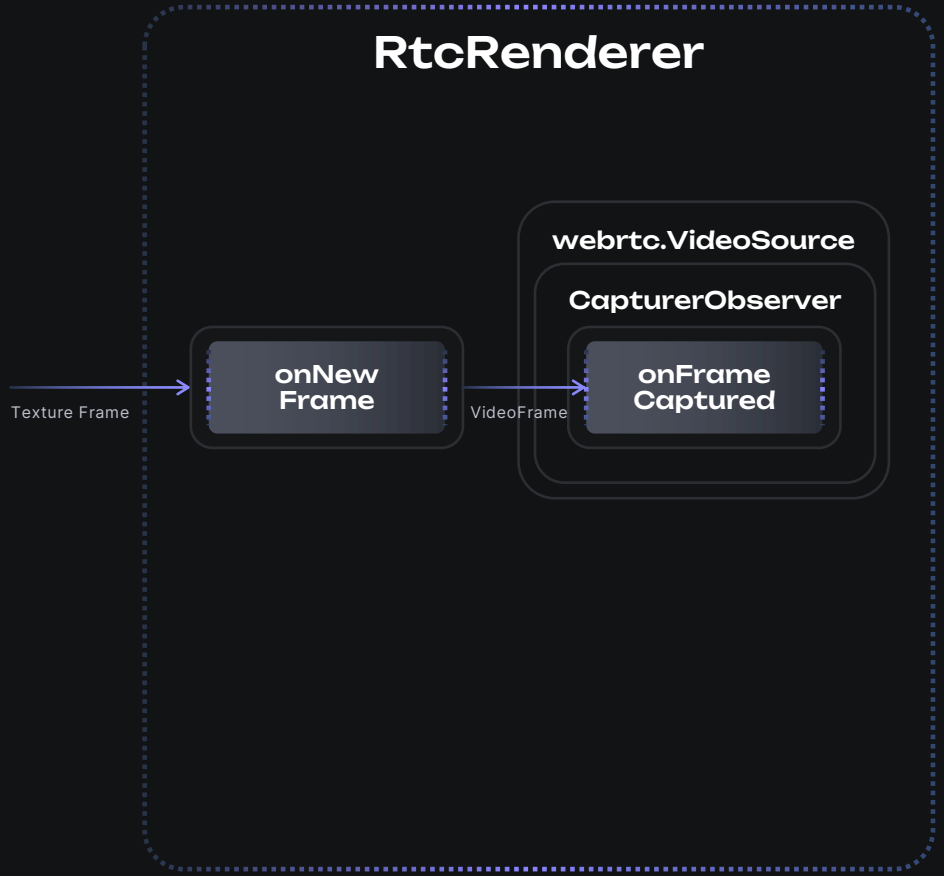
```
class RtcRenderer: TextureFrameConsumer {
    private var source: org.webrtc.VideoSource?

    override fun onNewFrame(frame: TextureFrame) {
        var videoFrame: VideoFrame? = null
        try {
            val texture = org.webrtc.TextureBufferImpl(
                frame.width,
                frame.height,
                VideoFrame.TextureBuffer.Type.RGB,
                frame.textureName,
                ...
            )
            videoFrame = org.webrtc.VideoFrame(texture,...)
            source?.capturerObserver?.onFrameCaptured(videoFrame)
        } catch (e: Exception) {
            // Handle error
        } finally {
            frame.release()
            videoFrame?.release()
        }
    }

}
```

WebRTC Render

```
class RtcRenderer: TextureFrameConsumer {
    private var source: org.webrtc.VideoSource?

    override fun onNewFrame(frame: TextureFrame) {
        var videoFrame: VideoFrame? = null
        try {
            val texture = org.webrtc.TextureBufferImpl(
                frame.width,
                frame.height,
                VideoFrame.TextureBuffer.Type.RGB,
                frame.textureName,
                ...
            )
            videoFrame = org.webrtc.VideoFrame(texture,...)
            source?.capturerObserver?.onFrameCaptured(videoFrame)
        } catch (e: Exception) {
            // Handle error
        } finally {
            frame.release()
            videoFrame?.release()
        }
    }

}
```

WebRTC Render

```
class RtcRenderer: TextureFrameConsumer {
    private var source: org.webrtc.VideoSource?

    override fun onNewFrame(frame: TextureFrame) {
        var videoFrame: VideoFrame? = null
        try {
            val texture = org.webrtc.TextureBufferImpl(
                frame.width,
                frame.height,
                VideoFrame.TextureBuffer.Type.RGB,
                frame.textureName,
                ...
            )
            videoFrame = org.webrtc.VideoFrame(texture,...)
            source?.capturerObserver?.onFrameCaptured(videoFrame)
        } catch (e: Exception) {
            // Handle error
        } finally {
            frame.release()
            videoFrame?.release()
        }
    }

}
```



DION

Blur Calculator

Mask Overlay

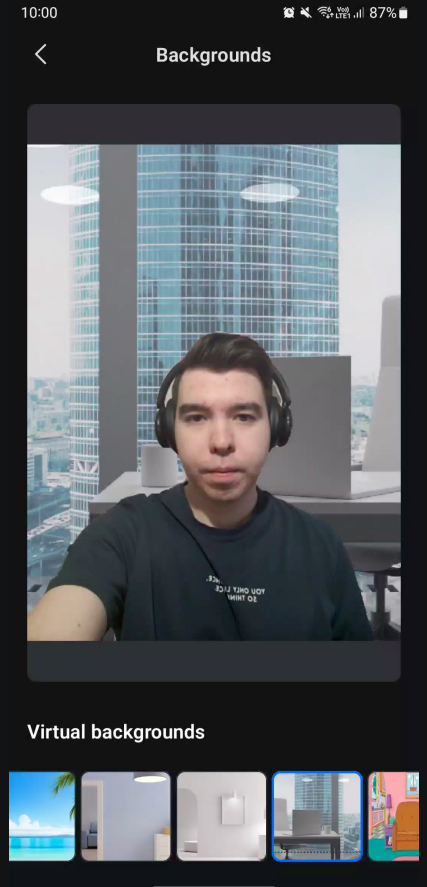
Scale image

JNI

Custom graphs

Results


10:00
87%
Backgrounds
Virtual backgrounds
None
Blur


10:00
87%
Backgrounds
Virtual backgrounds


10:00
87%
Backgrounds
Virtual backgrounds
None
Blur


10:00
87%
Backgrounds
Virtual backgrounds

Results

DAZN

Production release

Improvement in memory usage

APK Size

from 205mb to 62mb

installed size

from 236mb to 143mb

Improvement in CPU usage

Performance improvement

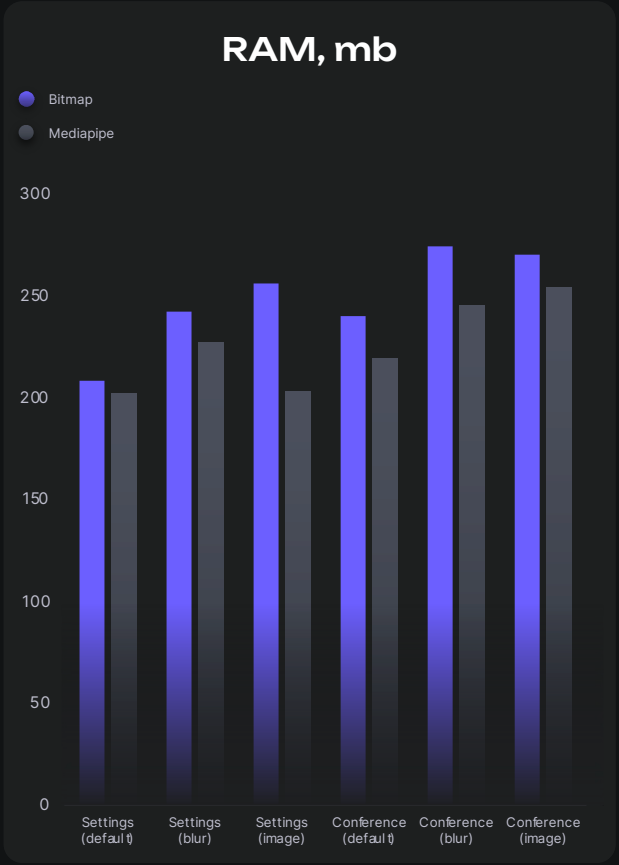
due to less usage of GC

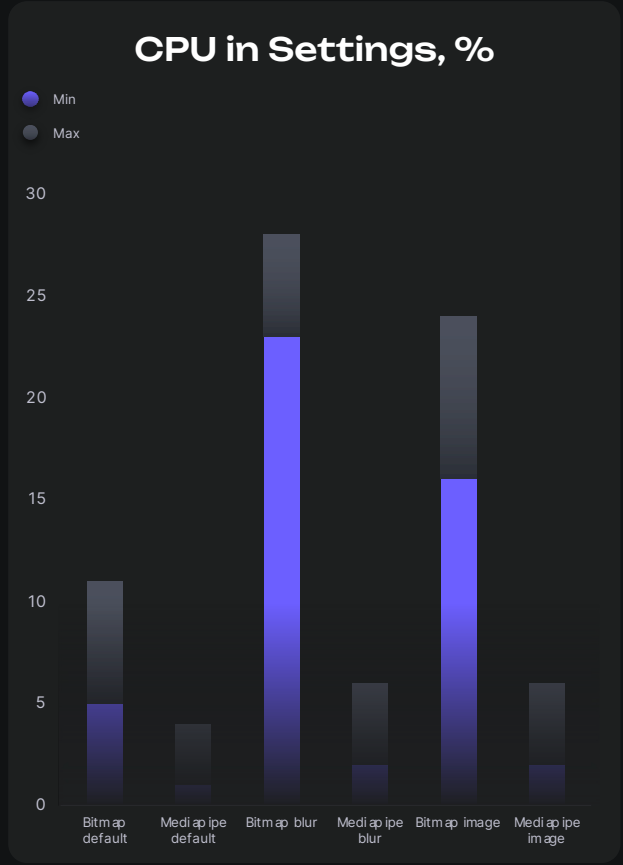
More control

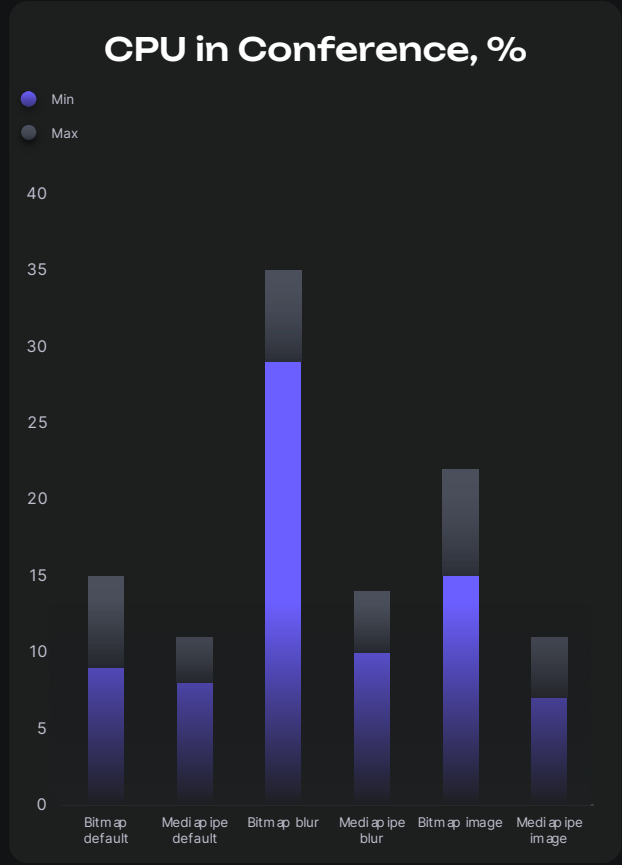
over the image processing process

Results

DIXON







Tips and tricks

Clangd

Hedron's Compile Commands Extractor for Bazel

Efficient Gaussian blur

Blur Kernel Generator

Reduce MediaPipe size

MaskOverlay fix

DION

Спасибо за внимание

support@diongo.ru