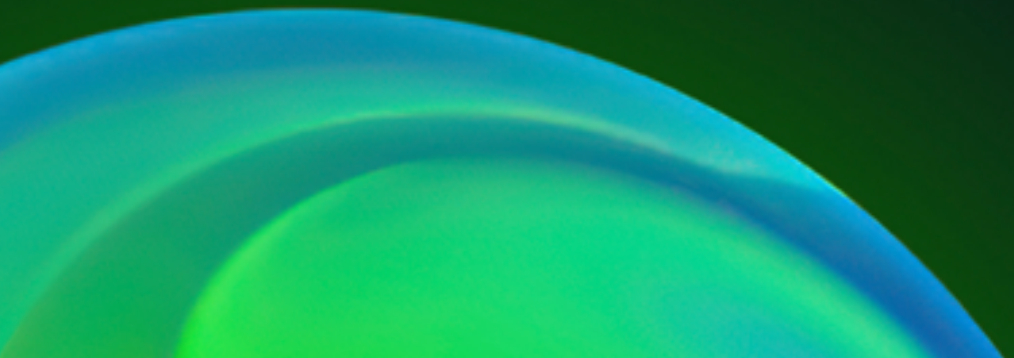
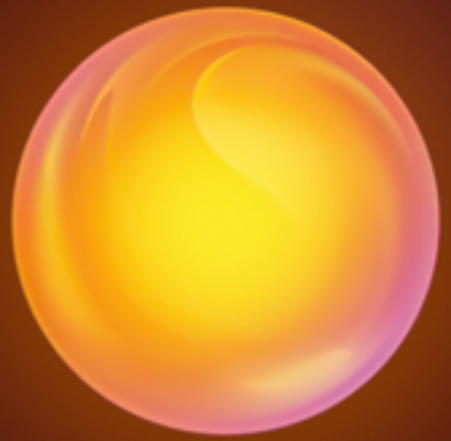
ML-обработка медиапотоков в браузере на примере видеоконференций SberJazz

Роман Лебедев, SberDevices
Дмитрий Балиев, SberDevices





Докладчики

Роман Лебедев

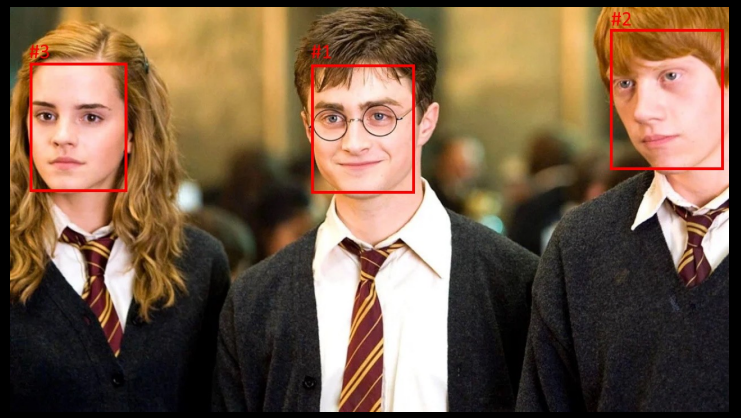
Computer Vision Platform
lebedev_rv

Дмитрий Балиев

Machine Learning RnD
balievdmitri

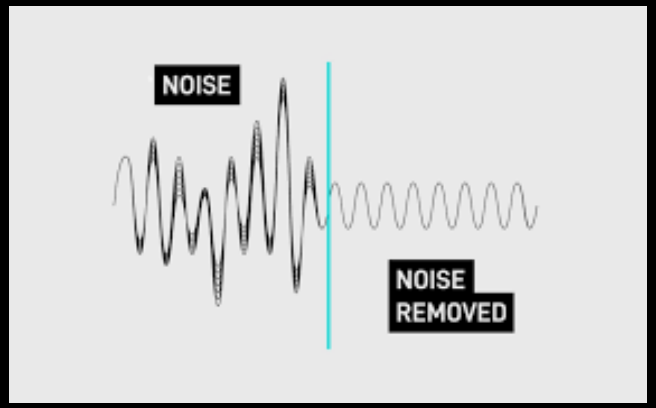
SBER DEVICES

Что можно делать с помощью ML?

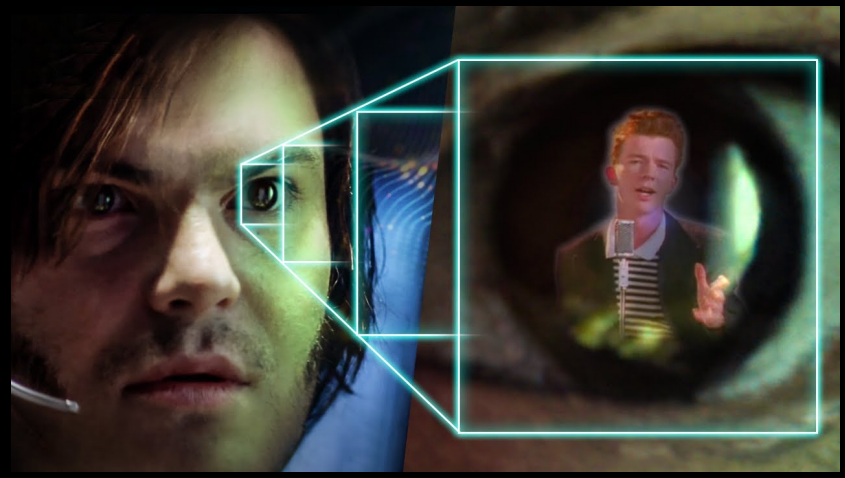


Находить лица

Заменять фон



Убирать шум с записи микрофона



Улучшать картинку

SBER DEVICES

SberJazz

SberJazz — платформа для видеоконференций

до 200 чел.
конференции

до 2000 чел.
вебинары

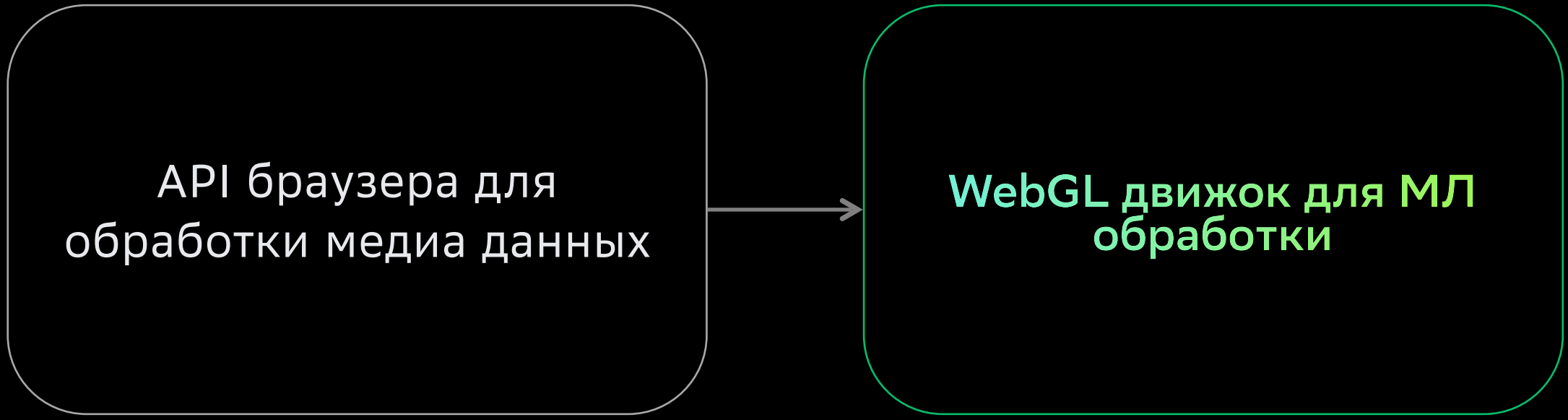
Привычный
и удобный интерфейс

Работает с любым
оборудованием

ML-алгоритмы:
- шумоподавление;
- замена фона;
- транскрибация речи.

Бесшовное
переключение
между устройствами

Агенда



API браузера для обработки медиа данных

Как получить данные?

Захватываем камеру и микрофон

```
const mediaStream = await navigator.mediaDevices.getUserMedia({
    audio: true,
    video: true,
});
```

MediaStream API

Выделяем дорожки данных

```
1   const [videoTrack] = mediaStream.getVideoTracks();
2   const [audioTrack] = mediaStream.getAudioTracks();
```

MediaStreamTrack

Обрабатываем поток данных

Streams API

Обрабатываем поток данных

WebCodecs API, MediaStreamTrackProcessor, MediaStreamTrackGenerator

Работаем с видео

```
const processor = new MediaStreamTrackProcessor({ track: videoTrack });
const transformer = new TransformStream(transformEffect);
const generator = new MediaStreamTrackGenerator({ kind: 'video' });

const readableStream = processor.readable;
const writableStream = generator.writable;

readableStream
    .pipeThrough(transformer)
    .pipeTo(writableStream);

const resultMediaStream = new MediaStream([generator]);
```

TransformStream

Работаем с видео

```
const processor = new MediaStreamTrackProcessor({ track: videoTrack });
const transformer = new TransformStream(transformEffect);
const generator = new MediaStreamTrackGenerator({ kind: 'video' });

const readableStream = processor.readable;
const writableStream = generator.writable;

readableStream
    .pipeThrough(transformer)
    .pipeTo(writableStream);

const resultMediaStream = new MediaStream([generator]);
```

Работаем с видео

```
const processor = new MediaStreamTrackProcessor({ track: videoTrack });
const transformer = new TransformStream(transformEffect);
const generator = new MediaStreamTrackGenerator({ kind: 'video' });

const readableStream = processor.readable;
const writableStream = generator.writable;

readableStream
    .pipeThrough(transformer)
    .pipeTo(writableStream);

const resultMediaStream = new MediaStream([generator]);
```

Работаем с видео

```
const processor = new MediaStreamTrackProcessor({ track: videoTrack });
const transformer = new TransformStream(transformEffect);
const generator = new MediaStreamTrackGenerator({ kind: 'video' });

const readableStream = processor.readable;
const writableStream = generator.writable;

readableStream
    .pipeThrough(transformer)
    .pipeTo(writableStream);

const resultMediaStream = new MediaStream([generator]);
```

Работаем с видео

```
const applyEffect = async (frame) ⇒ {
    const buffer = new Uint8Array(frame.allocationSize());
    const layout = await frame.copyTo(buffer);
    // process
    return new VideoFrame(buffer, frame);
};

const transformEffect = {
    async transform(frame, controller) {
        const newFrame = await applyEffect(frame);
        frame.close();
        controller.enqueue(newFrame);
    },
};
```

VideoFrame

Работаем с видео

```
const applyEffect = async (frame) ⇒ {
    const buffer = new Uint8Array(frame.allocationSize());
    const layout = await frame.copyTo(buffer);
    // process
    return new VideoFrame(buffer, frame);
};

const transformEffect = {
    async transform(frame, controller) {
        const newFrame = await applyEffect(frame);
        frame.close();
        controller.enqueue(newFrame);
    },
};
```

VideoFrame

Работаем с видео

```
const applyEffect = async (frame) ⇒ {
    const buffer = new Uint8Array(frame.allocationSize());
    const layout = await frame.copyTo(buffer);
    // process
    return new VideoFrame(buffer, frame);
};

const transformEffect = {
    async transform(frame, controller) {
        const newFrame = await applyEffect(frame);
        frame.close();
        controller.enqueue(newFrame);
    },
};
```

VideoFrame

Работаем с видео

```
const applyEffect = async (frame) ⇒ {
    const buffer = new Uint8Array(frame.allocationSize());
    const layout = await frame.copyTo(buffer);
    // process
    return new VideoFrame(buffer, frame);
};

const transformEffect = {
    async transform(frame, controller) {
        const newFrame = await applyEffect(frame);
        frame.close();
        controller.enqueue(newFrame);
    },
};
```

VideoFrame

Работаем с видео

```
const applyEffect = async (frame) ⇒ {
    const buffer = new Uint8Array(frame.allocationSize());
    const layout = await frame.copyTo(buffer);
    // process
    return new VideoFrame(buffer, frame);
};

const transformEffect = {
    async transform(frame, controller) {
        const newFrame = await applyEffect(frame);
        frame.close();
        controller.enqueue(newFrame);
    },
};
```

VideoFrame

Работаем с видео

```
1   const applyEffect = async (frame) ⇒ {
2       const buffer = new Uint8Array(frame.allocationSize());
3       const layout = await frame.copyTo(buffer);
4       // process
5       return new VideoFrame(buffer, frame);
6   };
7
8   const transformEffect = {
9       async transform(frame, controller) {
10          const newFrame = await applyEffect(frame);
11          frame.close();
12          controller.enqueue(newFrame);
13      },
14  };
```

VideoFrame

Работаем с аудио

```
const processor = new MediaStreamTrackProcessor({ track: audioTrack });
const transformer = new TransformStream(transformEffect);
const generator = new MediaStreamTrackGenerator({ kind: 'audio' });

const readableStream = processor.readable;
const writableStream = generator.writable;

readableStream
    .pipeThrough(transformer)
    .pipeTo(writableStream);

const resultMediaStream = new MediaStream([generator]);
```

AudioData

SBER DEVICES

Успех?

Поддержка WebCodecs в браузерах



Только видео 16.4+

caniuse web codecs

Альтернативы для видео?

<video> и <canvas>

```
1   const inputVideo = document.createElement('video');
2   inputVideo.srcObject = mediaStream;
3   inputVideo.play();
4   const { width, height } = track.getSettings();
5
6   const inputCanvas = document.createElement('canvas');
7   inputCanvas.width = width;
8   inputCanvas.height = height;
9   const inputCtx = inputCanvas.getContext(‘2d’, { willReadFrequently: true });
10
11  const outputCanvas = document.createElement('canvas');
12  outputCanvas.width = width;
13  outputCanvas.height = height;
14  const outputCtx = outputCanvas.getContext(‘2d’);
15
16  const resultMediaStream = outputCanvas.captureStream(targetFPS);
```

Canvas API

<video> и <canvas>

```
const inputVideo = document.createElement('video');
inputVideo.srcObject = mediaStream;
inputVideo.play();
const { width, height } = track.getSettings();

const inputCanvas = document.createElement('canvas');
inputCanvas.width = width;
inputCanvas.height = height;
const inputCtx = inputCanvas.getContext('2d', { willReadFrequently: true });

const outputCanvas = document.createElement('canvas');
outputCanvas.width = width;
outputCanvas.height = height;
const outputCtx = outputCanvas.getContext('2d');

const resultMediaStream = outputCanvas.captureStream(targetFPS);
```

Canvas API

Захватываем входной кадр

```
1   const processFrame = () ⇒ {
2       const startTimeMs = performance.now();
3
4       inputCtx.drawImage(inputVideo, 0, 0, width, height);
5       const imageData = inputCtx.getImageData(0, 0, width, height);
6
7       const resultImageData = proccessImageData(imageData);
8       outputCtx.putImageData(resultImageData, 0, 0);
9
10      const endTimeMs = performance.now();
11
12      setTimeout(processFrame, timePerFrameMs - (endTimeMs - startTimeMs));
13  }
14  processFrame();
```

Canvas API

Получаем новый кадр

```
1   const processFrame = () ⇒ {
2       const startTimeMs = performance.now();
3
4       inputCtx.drawImage(inputVideo, 0, 0, width, height);
5       const imageData = inputCtx.getImageData(0, 0, width, height);
6
7       const resultImageData = proccessImageData(imageData);
8       outputCtx.putImageData(resultImageData, 0, 0);
9
10      const endTimeMs = performance.now();
11
12      setTimeout(processFrame, timePerFrameMs - (endTimeMs - startTimeMs));
13  }
14  processFrame();
```

Canvas API

Перебираем кадры

```
1   const processFrame = () ⇒ {
2       const startTimeMs = performance.now();
3
4       inputCtx.drawImage(inputVideo, 0, 0, width, height);
5       const imageData = inputCtx.getImageData(0, 0, width, height);
6
7       const resultImageData = proccessImageData(imageData);
8       outputCtx.putImageData(resultImageData, 0, 0);
9
10      const endTimeMs = performance.now();
11
12      setTimeout(processFrame, timePerFrameMs - (endTimeMs - startTimeMs));
13  }
14  processFrame();
```

setTimeout(), requestAnimationFrame()

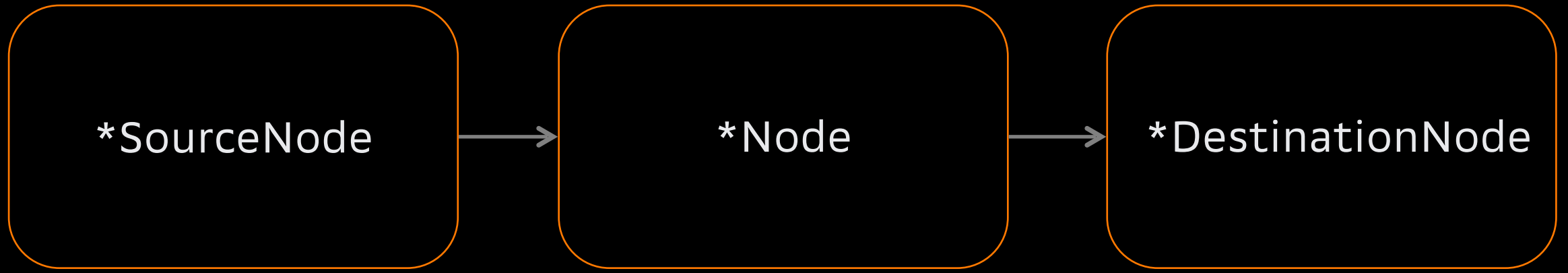
Работаем с кадром

```
const proccessImageData = (imageData) ⇒ {
    for (let i = 0; i < imageData.data.length; i += 4) {
        const r = imageData.data[i];
        const g = imageData.data[i + 1];
        const b = imageData.data[i + 2];
        const a = imageData.data[i + 3];
    }
    return imageData;
};
```

ImageData

Альтернативы для аудио?

Web Audio API



Web Audio API

Web Audio API

Работаем с Web Audio

```
const ctx = new AudioContext({ sampleRate: 48000 });

const source = ctx.createMediaStreamSource(mediaStream);

await ctx.audioWorklet.addModule("worklet-processor.js");
const processor = new AudioWorkletNode(ctx, "processor");

const destination = ctx.createMediaStreamDestination();

source
    .connect(processor)
    .connect(destination);

const resultStream = destination.stream;
```

Web Audio API, AudioWorkletProcessor

Работаем с Web Audio

```
const ctx = new AudioContext({ sampleRate: 48000 });

const source = ctx.createMediaStreamSource(mediaStream);

await ctx.audioWorklet.addModule("worklet-processor.js");
const processor = new AudioWorkletNode(ctx, "processor");

const destination = ctx.createMediaStreamDestination();

source
    .connect(processor)
    .connect(destination);

const resultStream = destination.stream;
```

Web Audio API, AudioWorkletProcessor

Работаем с Web Audio

```
const ctx = new AudioContext({ sampleRate: 48000 });

const source = ctx.createMediaStreamSource(mediaStream);

await ctx.audioWorklet.addModule("worklet-processor.js");
const processor = new AudioWorkletNode(ctx, "processor");

const destination = ctx.createMediaStreamDestination();

source
    .connect(processor)
    .connect(destination);

const resultStream = destination.stream;
```

Web Audio API, AudioWorkletProcessor

Работаем с Web Audio

```
const ctx = new AudioContext({ sampleRate: 48000 });

const source = ctx.createMediaStreamSource(mediaStream);

await ctx.audioWorklet.addModule("worklet-processor.js");
const processor = new AudioWorkletNode(ctx, "processor");

const destination = ctx.createMediaStreamDestination();

source
    .connect(processor)
    .connect(destination);

const resultStream = destination.stream;
```

Web Audio API, AudioWorkletProcessor

Работаем с Web Audio

```
const ctx = new AudioContext({ sampleRate: 48000 });

const source = ctx.createMediaStreamSource(mediaStream);

await ctx.audioWorklet.addModule("worklet-processor.js");
const processor = new AudioWorkletNode(ctx, "processor");

const destination = ctx.createMediaStreamDestination();

source
    .connect(processor)
    .connect(destination);

const resultStream = destination.stream;
```

Web Audio API, AudioWorkletProcessor

Работаем с Web Audio

```
const ctx = new AudioContext({ sampleRate: 48000 });

const source = ctx.createMediaStreamSource(mediaStream);

await ctx.audioWorklet.addModule("worklet-processor.js");
const processor = new AudioWorkletNode(ctx, "processor");

const destination = ctx.createMediaStreamDestination();

source
    .connect(processor)
    .connect(destination);

const resultStream = destination.stream;
```

Web Audio API, AudioWorkletProcessor

SBER DEVICES

Определяем AudioWorkletProcessor

```
1   class IdWorkletProcessor extends AudioWorkletProcessor {
2       process(inputs, outputs, parameters) {
3           for (let i = 0; i < inputs.length; i++) {
4               const input = inputs[i];
5               const output = outputs[i];
6
7               for (let channel = 0; channel < input.length; channel++)
8                   output[channel].set(input[channel]);
9           }
10
11          return true;
12      }
13  }
14
15  registerProcessor("processor", IdWorkletProcessor);
```

AudioWorkletProcessor

Определяем AudioWorkletProcessor

```
class IdWorkletProcessor extends AudioWorkletProcessor {
    process(inputs, outputs, parameters) {
        for (let i = 0; i < inputs.length; i++) {
            const input = inputs[i];
            const output = outputs[i];

            for (let channel = 0; channel < input.length; channel++)
                output[channel].set(input[channel]);
        }

        return true;
    }
}

registerProcessor("processor", IdWorkletProcessor);
```

AudioWorkletProcessor

Теперь успех?

Что по нагрузке?

Web Workers

Web Workers, MessagePort

Transferable objects

- MessagePort
- ArrayBuffer
- ReadableStream, WritableStream, TransformStream
- AudioData, VideoFrame
- OffscreenCanvas
- ImageBitmap

Работаем с Main Thread

```
const processor = new MediaStreamTrackProcessor({ track });
const generator = new MediaStreamTrackGenerator({ kind });

const readableStream = processor.readable;
const writableStream = generator.writable;

const worker = new Worker("transformer.worker.js");

worker.postMessage({ readableStream, writableStream },
    [readableStream, writableStream]
);
```

Работаем с Main Thread

```
const processor = new MediaStreamTrackProcessor({ track });
const generator = new MediaStreamTrackGenerator({ kind });

const readableStream = processor.readable;
const writableStream = generator.writable;

const worker = new Worker("transformer.worker.js");

worker.postMessage({ readableStream, writableStream },
    [readableStream, writableStream]
);
```

Работаем с Main Thread

```
const processor = new MediaStreamTrackProcessor({ track });
const generator = new MediaStreamTrackGenerator({ kind });

const readableStream = processor.readable;
const writableStream = generator.writable;

const worker = new Worker("transformer.worker.js");

worker.postMessage({ readableStream, writableStream },
    [readableStream, writableStream]
);
```

Работаем с Main Thread

```
const processor = new MediaStreamTrackProcessor({ track });
const generator = new MediaStreamTrackGenerator({ kind });

const readableStream = processor.readable;
const writableStream = generator.writable;

const worker = new Worker("transformer.worker.js");

worker.postMessage({ readableStream, writableStream },
    [readableStream, writableStream]
);
```

Работаем с Worker Thread

```
onmessage = (event) ⇒ {
    const { readableStream, writableStream } = event.data;

    const transformer = new TransformStream(transformEffect);

    readableStream
        .pipeThrough(transformer)
        .pipeTo(writableStream);
};
```

Работаем с Worker Thread

```
onmessage = (event) ⇒ {
    const { readableStream, writableStream } = event.data;

    const transformer = new TransformStream(transformEffect);

    readableStream
        .pipeThrough(transformer)
        .pipeTo(writableStream);
};
```

Работаем с Worker Thread

```
onmessage = (event) ⇒ {
    const { readableStream, writableStream } = event.data;

    const transformer = new TransformStream(transformEffect);

    readableStream
        .pipeThrough(transformer)
        .pipeTo(writableStream);
};
```

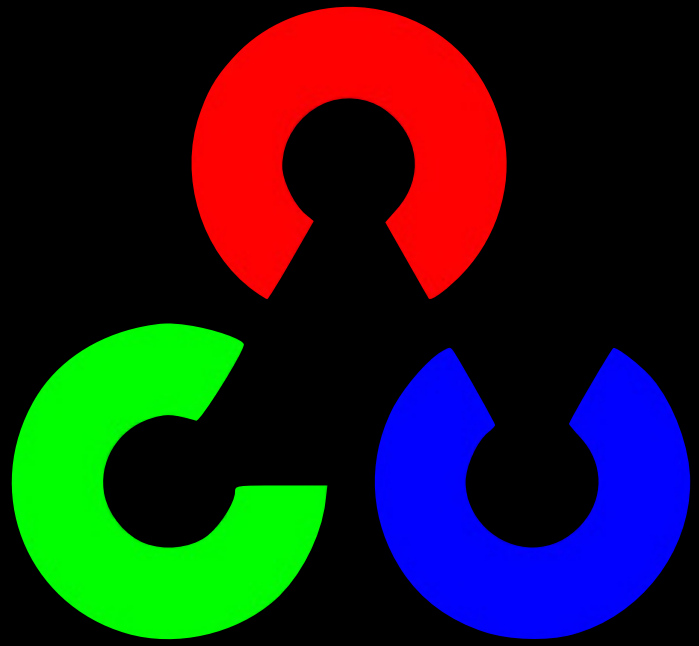
Что с вычислениями?

Backend исполнения

WebAssembly, WebGL, WebGPU

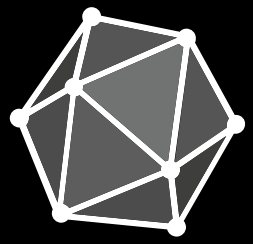
Как быстро потрогать?

OpenCV



OpenCV.js Tutorials

MediaPipe



ONNX

TensorFlow

ONNX Runtime Web, Tensorflow.js

Какие выводы?

Хотим WebCodecs,
но пока не везде,
помним про WebAudio
и <video> с <canvas>

Выносим тяжелые
вычисления в Worker

Пробуем готовые
решения для NN

Используем
WASM или WebGL

А что у нас?

SBER DEVICES

Велосипед!

О чем я расскажу

Как принять решение,
нужен ли вам велосипед?

Если нужен, то как его
сделать?

Мотивация

А зачем вообще делать свой движок?

Мотивация

А зачем вообще делать свой движок?

1. Оптимизация
2. Универсальность
3. Потому что можем

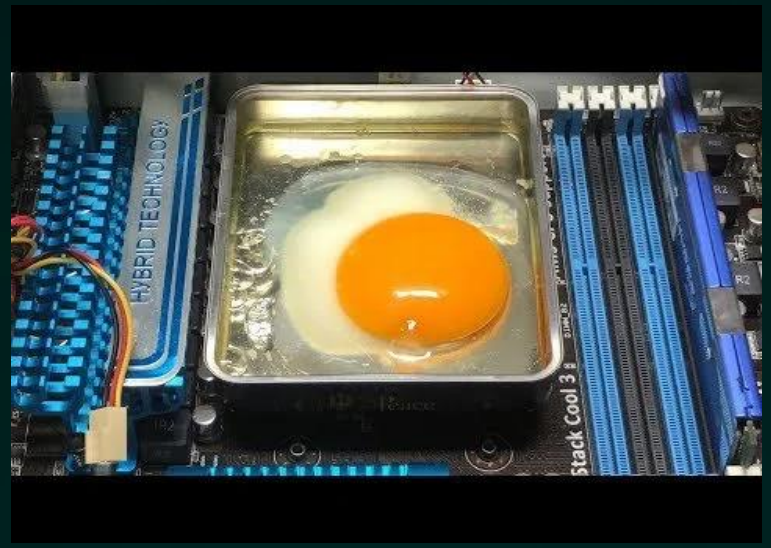
Оптимизация

GPU эффективен

- свертки;
- upscale;
- blur;
- alpha-blending.

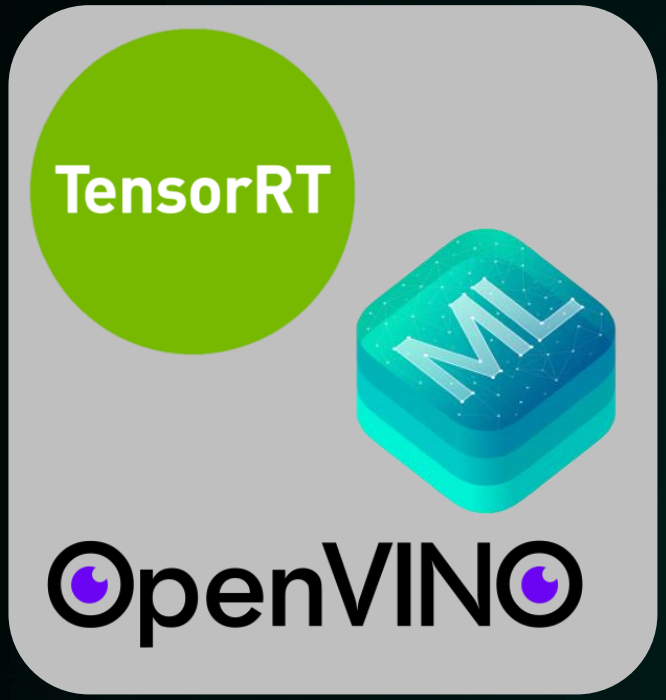
Контроль

- над памятью;
- над низкоуровневой математикой.



Оптимизировать можно разные задачи

Универсальность



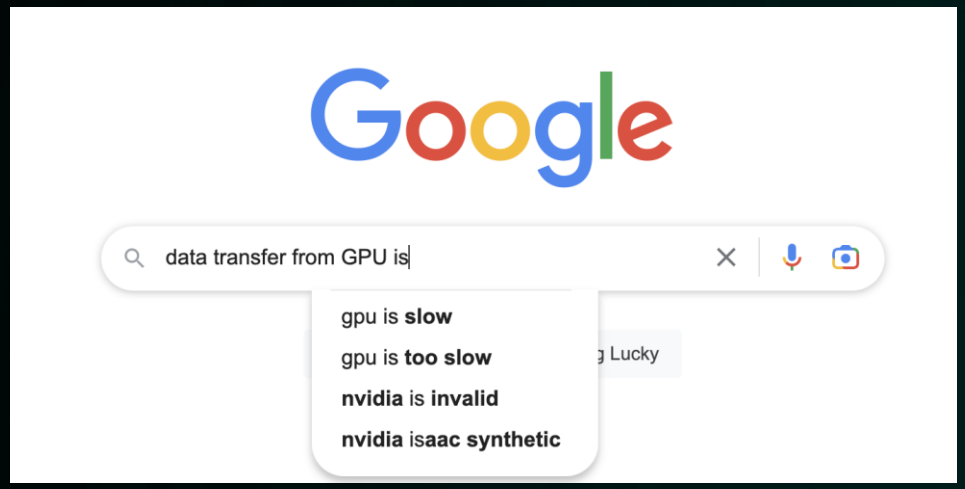
Оптимальная производительность

Обработку нужно писать отдельно

Работает везде

Обработку можно писать на WebGL

Универсальность



Передачи данных между CPU и GPU стоит избегать

Требования к разработчикам

В поиске эльфа:

- низкоуровневое программирование (C++);
- математика и матричные вычисления;
- понимание работы hardware;
- понимание, как работают ML фреймворки внутри.

5 месяцев

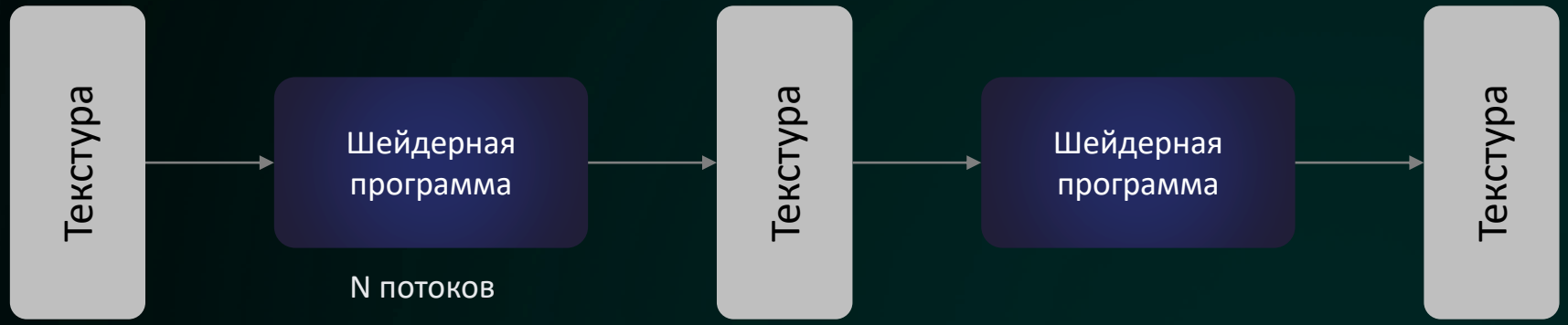
До первого рабочего прототипа

Получаем

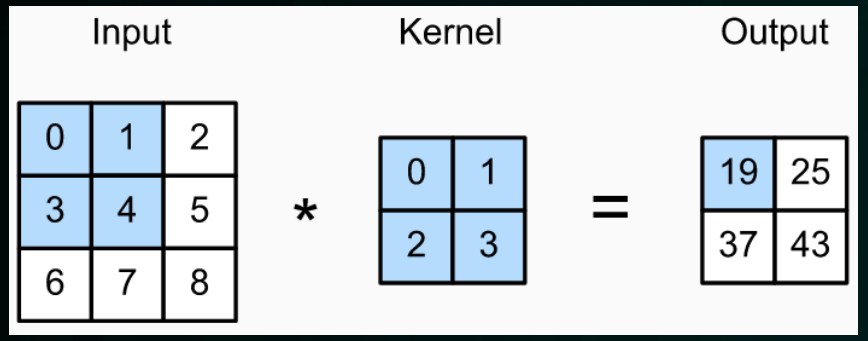
- универсальный код, который можно запустить на любой платформе;
- высокую производительность за счет использования GPU;
- высокие требования к компетенциям разработчиков.

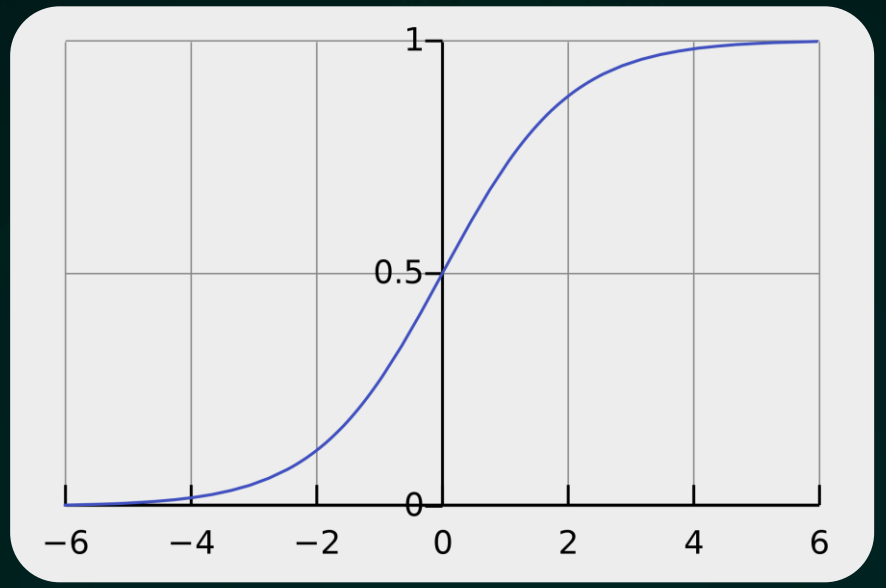
Реализуем?

Архитектурные идеи



Архитектурные идеи





«Толстые» операции:

- convolution;
- pooling;
- upscale;
- gaussian blur.

«Тонкие» операции:

- поэлементные операции;
- функции активации.

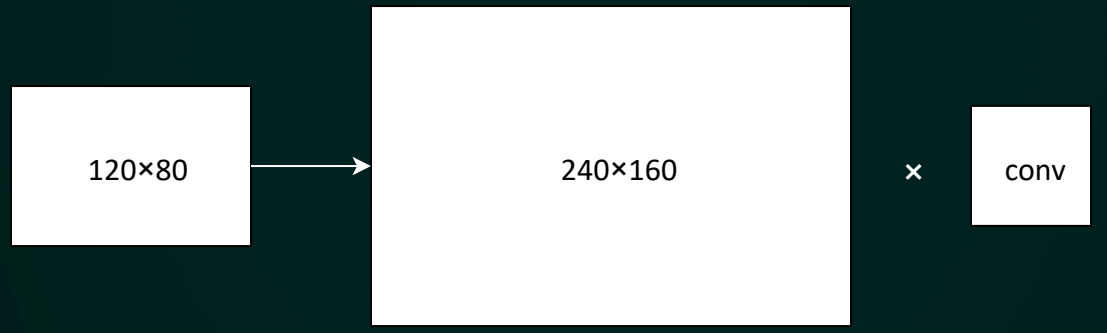
SBER DEVICES

1 шейдерная программа

= 1 «Толстая» операция
+ [0; N] «Тонких» операций

Некоторые операции можно эффективно объединить

Upconv = Upscale + Conv



Что это нам дает?

10%

Прирост производительности от совмещения «Толстых» и «Тонких» операций в одной программе

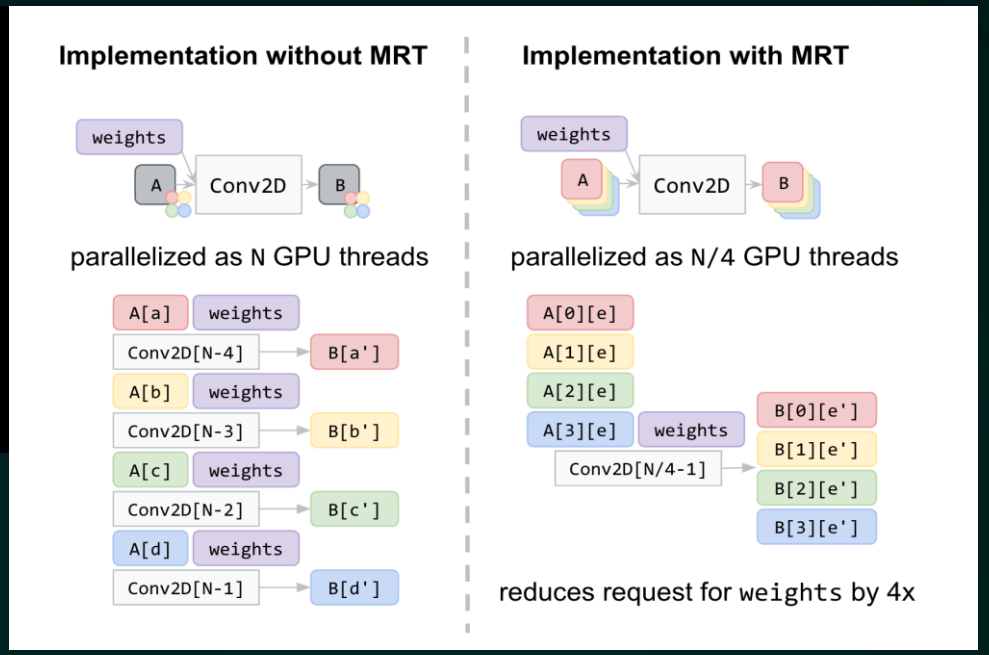
15%

Прирост производительности от совмещения Upscale и свертки в одну операцию

Откуда мы взяли идею Multiple Render Targets

Ключевые идеи:

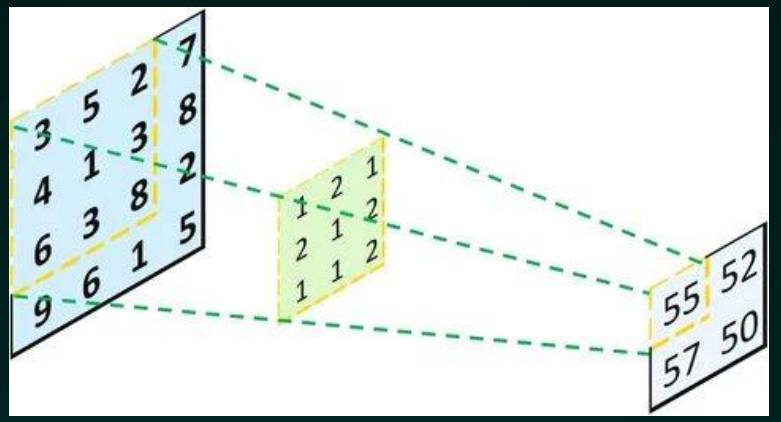
- свертка – memory-bound операция, нужно оптимизировать чтение данных из памяти;
- производительность в WebGL порядка 90% от реализации в OpenGL с Compute Shaders;



https://blog.research.google/2022/08/high-definition-segmentation-in-google.html

Почему MRT работает?

Чтения	Без MRT	MRT 4 пикселя	MRT на 1 пиксель
Ядро свертки	9	9	2,25
Картинка	9	16	4
Всего	18	25	6,25 (35%)



3x

Ожидаемый прирост
производительности
от Multiple Render Target

Тестируем гипотезу

Слой	T → T	T → Q	Q → T	Q → Q
Ker=3x3, size=64x32x40, ch_out=80	1,48	0,61	2,31	0,46
Ker=3x3, size=64x32x80, ch_out=160, stride=2	1,77	0,98	1,72	0,77
Ker=3x3, size=160x160x160, channel-wise	1,86	0,82	3,50	0,51
Ker=5x5, size=64x32x40, ch_out=40	1,91	0,69	2,91	0,60
Upscale=2x2 + Ker=3x3, size=64x32x40, ch_out=20	1,11	0,41	1,49	0,57

Скорость работы отдельных слоев в зависимости от типов текстур на входе и на выходе

2-3x

Прирост быстродействия на «тяжелых» слоях

SBER DEVICES

Результаты

	Без MRT	С MRT
Замена фона только нейросеть	10,3 мс	6,7 мс
Замена фона полный пайплайн	14 мс	9 мс

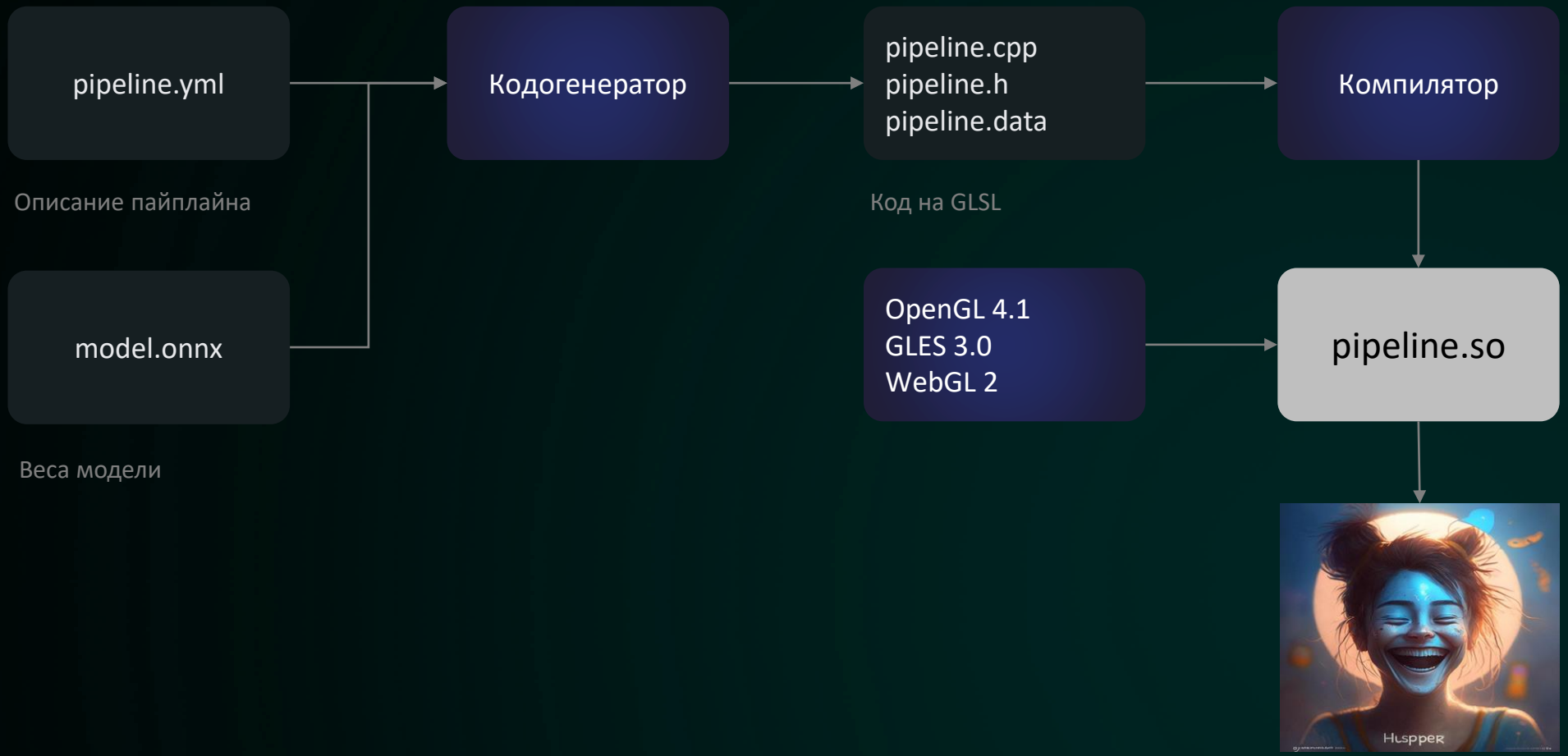
Время работы до и после оптимизации MRT

36%

Реальный прирост
производительности
от Multiple Render Target

SBER DEVICES

Как это работает в целом



Тестируем?

Модели

Соответствие результатов в WebGL и ONNX

Unit тесты

Для отдельных компонент

Side-by-side

Для end-to-end проверки качества

CI/CD

Mesa driver + xvfb-run

О чем поговорили?

Нужен ли вам велосипед?

Как его сделать?

SBER DEVICES


SBER DEVICES