Implementing AI-powered Semantic Character Recognition in Motor Racing

GTC GPU TECHNOLOGY CONFERENCE

SILICON VALLEY
MARCH 2020

Jesús Hormigo
David Albarracín

Typical race start in television

Can you identify who is who?

A race start with augmented context

Isn’t this more understandable now?

How could this be done?

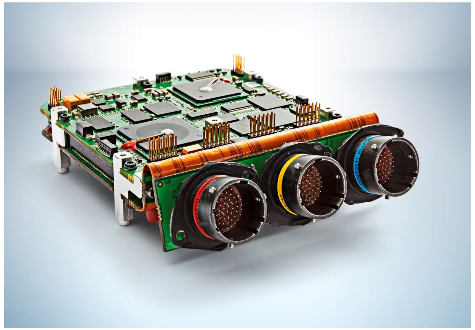
1.- Manually

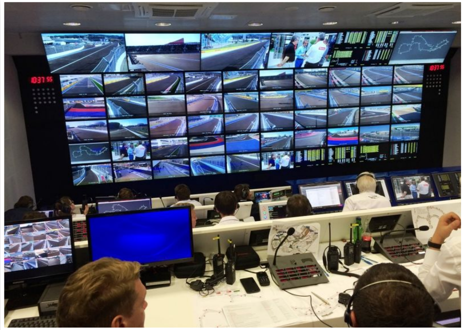
A well trained human could potentially point and click to manually set the tags on every scene.

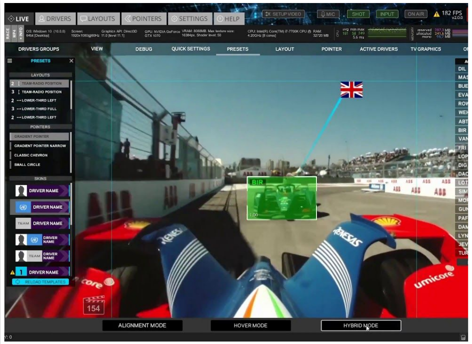
The challenge is that during live production in TV broadcasting, where shots are typically between 2-3 secs long, TV operators can’t be fast enough to point and click, nor to add by hand any tags in a consistent way.

How could this be done?

2.- With advanced technical solutions involving GPS







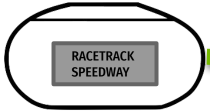
How could this be done?

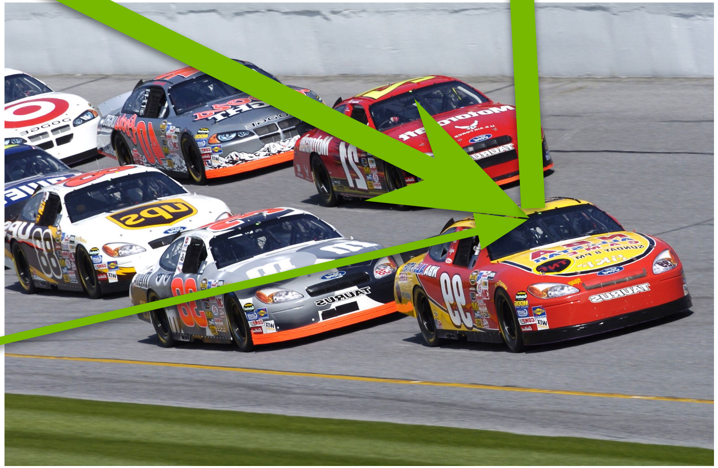
1 GLOBAL POSITIONING SATELLITES
Track the positions of the racers via a small antenna mounted in each car's roof.

2 UNBOARD COMPUTER
Collects telemetry data (such as speed, throttle, brake, gears, g-force, A.P.M., etc) and streams it five times per second through the car's antennae to a series of base stations.

3 BASE STATION ANTENNAS
Relay the data from the cars to the production tracks. They are mounted on top of the grandstands and on camera platforms on the back stretch. There are usually between four and eight, depending on track geometry.

Base Station locations





4 CAMERAS
Track and record positions of cars to create images that are sent to the production trucks through cables. There are six cameras positioned at strategic areas around the track to provide the best possible angles for using the system pointer graphics feature.

3 BASE STATION ANTENNAS
House data-gathering computers, rendering machines, network hubs, video feeds, monitors and audio equipment that an eight-to-ten man crew converts into real time graphics using software.

5 END RESULTS
The final graphics are sent out for television, internet and wireless devices

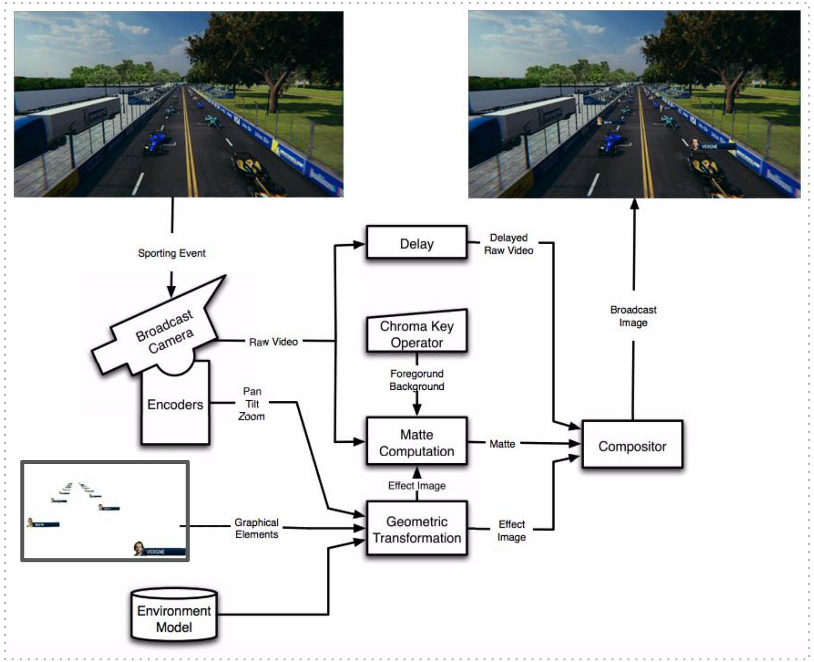
* Example of the pointer feature that follows the car in real time.

* Example of the telemetry information displayed along bottom.

How could this be done?

However, this solution have some cons:

- Complex technical implementation
- High operational costs
- Expensive electronics including RTK GPS (with a reference base station) for every car
- Kilometers (or miles :-]) of cable wired to each base station throughout the whole circuit
- Cameras with special “base” positioned at strategic areas around the track to provide the best possible angles and not valid with every shot
- Onboard cameras are not possible to be tracked even with the above methodology



Our Approach - AI to the rescue

3.- With Neural Networks

We developed an autonomous system that **trains itself** on synthetic data based on a FBX model and **infers in real time**.

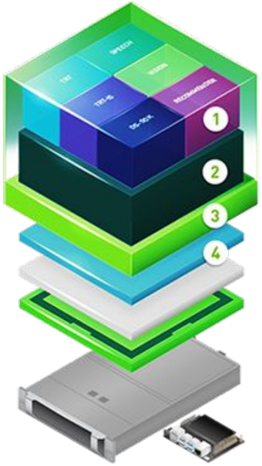
This way we can augment the context of scenes automatically and at a very low cost of implementation and operation.

Our Approach - Automated training and RT Inference

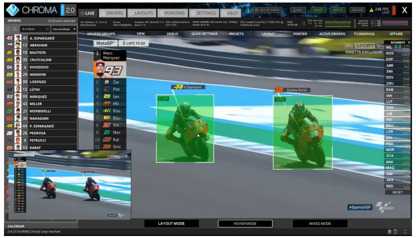
Automated Dataset Generation

AI Training

Deployment



Real Time Inference



The challenges

- ➢ Lighting conditions are variable during the day
- ➢ In some series the car models have exactly the same shape - and only have changing the liveries
- ➢ Inference has to be in REAL TIME as few frames later is too late to be used
- ➢ Matching specific standards in connectors and delivery of picture in HD

Let's deep dive into the details

- The Neural network architecture
- Synthetic dataset generation
- Training the neural network
- The inference
- The API
- The operator's User Interface
- The results
- Next generation neural network

The Neural Network

TensorFlow

Keras

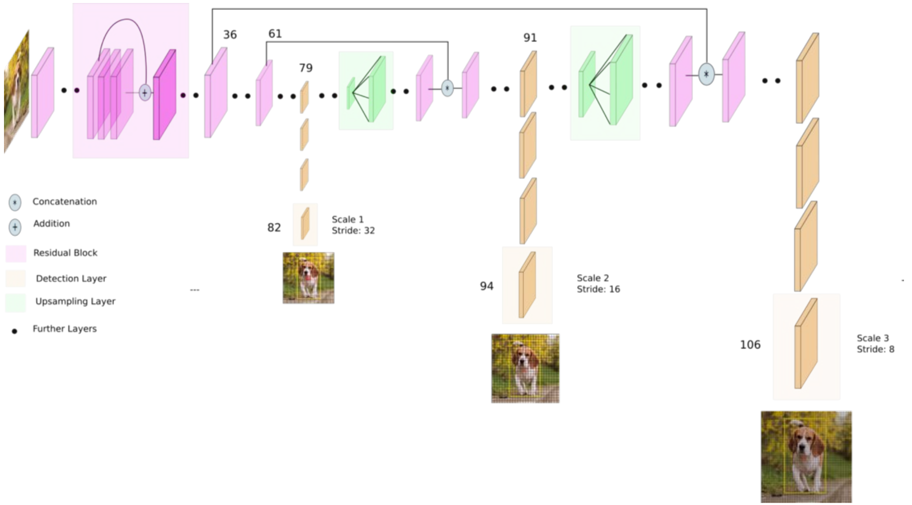
jupyter

NVIDIA
CUDA

python

Google
Cloud
TPU

TensorRT



The Training with real images

Real data is expensive to produce, sometimes dangerous and biased. There is no availability of real images before the first race and is a work intensive task with not enough footage to train the neural network and generate a good model.

In addition to this, almost no team races during testing with the final liveries.

Traditional workflow requires:

- Tagging precision
- Time to deliver
- Dataset preparation
- Data augmentation

The Training - 100% Synthetic training

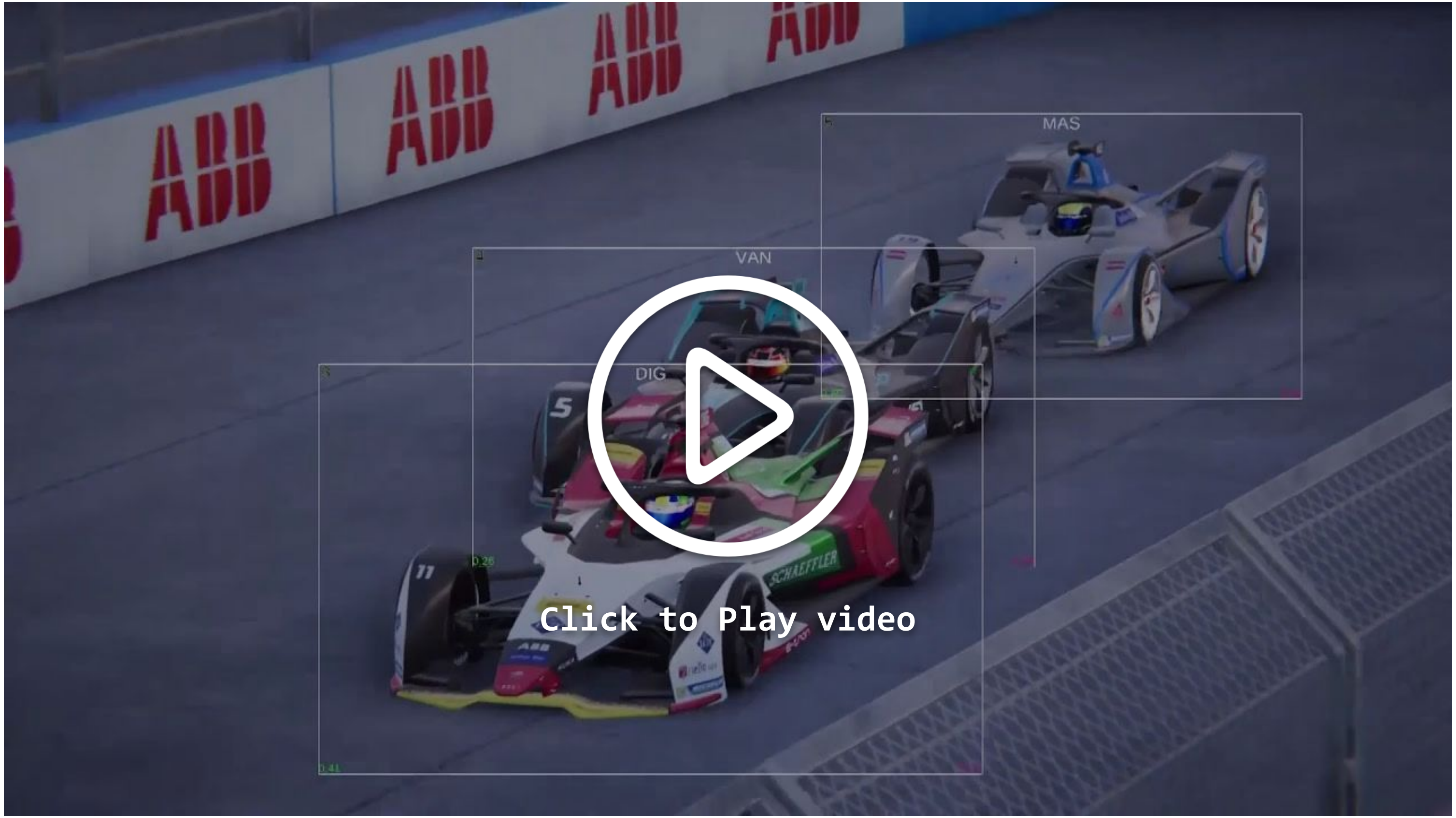
Synthetic labels are automatic and accurate and extremely useful for training, in addition to validation

Our workflow consists of:

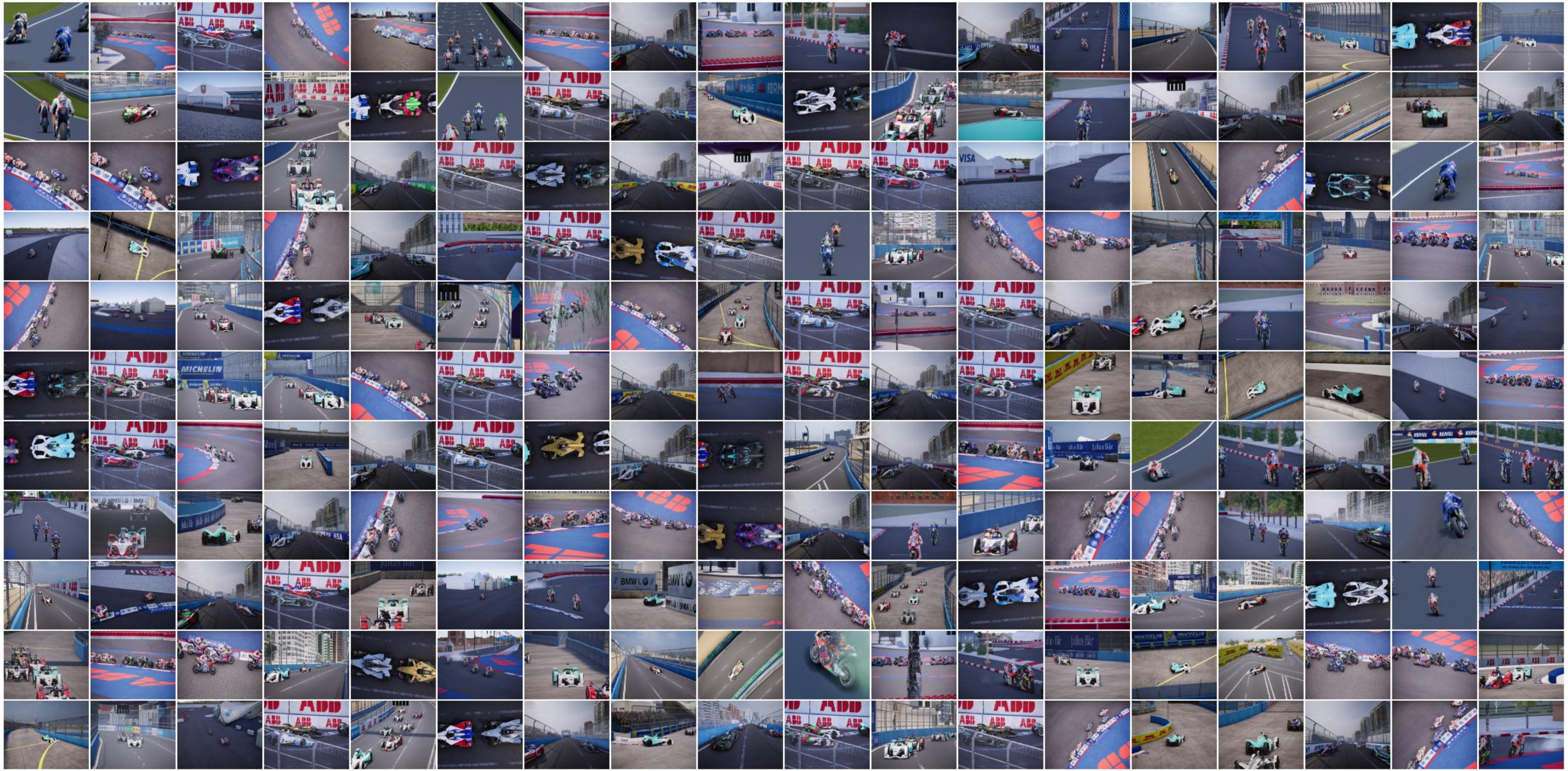
- Semi-automated tagging of real images
- Generation of synthetic images with latest changes
- Fine tuned training pipeline
  - Freeze most convolutional blocks and train only front-end from scratch
  - Train whole network loading pre-trained weights
- Distribution of dataset: training - validation
- Mixing synthetic and real images for better usability
  - More detections pass threshold
  - Rise of accuracy metric around 15%

Lighting conditions affects detection

The Training - Synthetic image generation



The Training - Our Synthetic DataSet



The Training - DataSet building

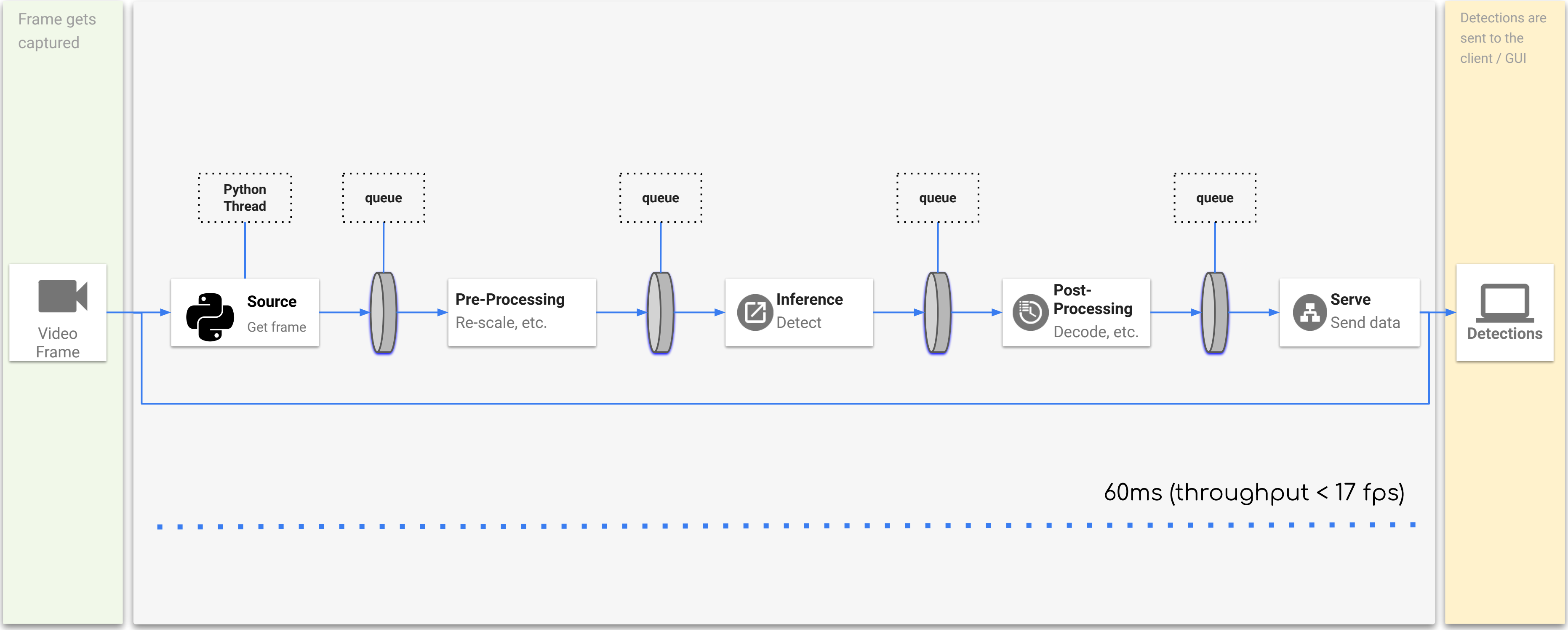
```python
def load_bboxes(annot_data, annot_format, filter_conf, invertYAxis):

    # Return list
    bounding_boxes = []

    if annot_format.startswith('Synth1819'):

        for bbox in annot_data['BoundingBoxList']:

            # Retrieving data
            driver = bbox['Driver']
            car_bb = bbox['Car_percent_in_bb']
            oth_bb = bbox['Other_cars_percent_in_bb']
            x_fix = bbox['Fixed_bb']['x']
            y_fix = bbox['Fixed_bb']['y']
            w_fix = bbox['Fixed_bb']['width']
            h_fix = bbox['Fixed_bb']['height']

            # Getting combinated area reduction coefficient
            comb_coef, bbox_factor = combined_area_reduction_coef(bbox, filter_conf)

            # Parsing bbox params to VOC format
            bounding_box = {
                'label': driver,
                'xmin': x_fix,
                'ymin': y_fix,
                'xmax': (x_fix + w_fix),
                'ymax': (y_fix + h_fix),
                'car_percent_in_bb': car_bb,
                'other_cars_percent_in_bb': oth_bb,
                'combined_area_reduction_coef': comb_coef,
                'bbox_factor': bbox_factor
            }

            if 'CarVisibleRatio' in bbox:
                bounding_box['CarVisibleRatio'] = bbox['CarVisibleRatio']

            bounding_boxes.append(bounding_box)

    ...
```

```python
def filtering(imageshape, bboxes, filter_conf):

    # Load parameters and prepare data

    ...

    # Return list
    filtered_bboxes = []

    # For each bbox run filtering
    for idx, bbox in enumerate(bboxes):

        # Filter 1: Other Cars In Bounding Box
        # If bbox comes with this additional data, meaning, if the
        # bbox was generated synthetically, then we can filter
        if ('car_percent_in_bb' in bbox.keys() and
            'other_cars_percent_in_bb' in bbox.keys() and
            'combined_area_reduction_coef' in bbox.keys()):

            # Getting data from bbox
            car_in_bb = bbox['car_percent_in_bb']
            oth_in_bb = bbox['other_cars_percent_in_bb']
            comb_area_coef = bbox['combined_area_reduction_coef']

            car_check = car_in_bb > min_car_in_bb
            oth_check = oth_in_bb < max_other_cars_in_bb
            comb_a_check = comb_area_coef > min_comb_area_coef

            if not (car_check and oth_check and comb_a_check):
                continue

        ...

        # We passed!
        filtered_bboxes.append(bbox)

    return filtered_bboxes
```

The Inference during Live Broadcasting

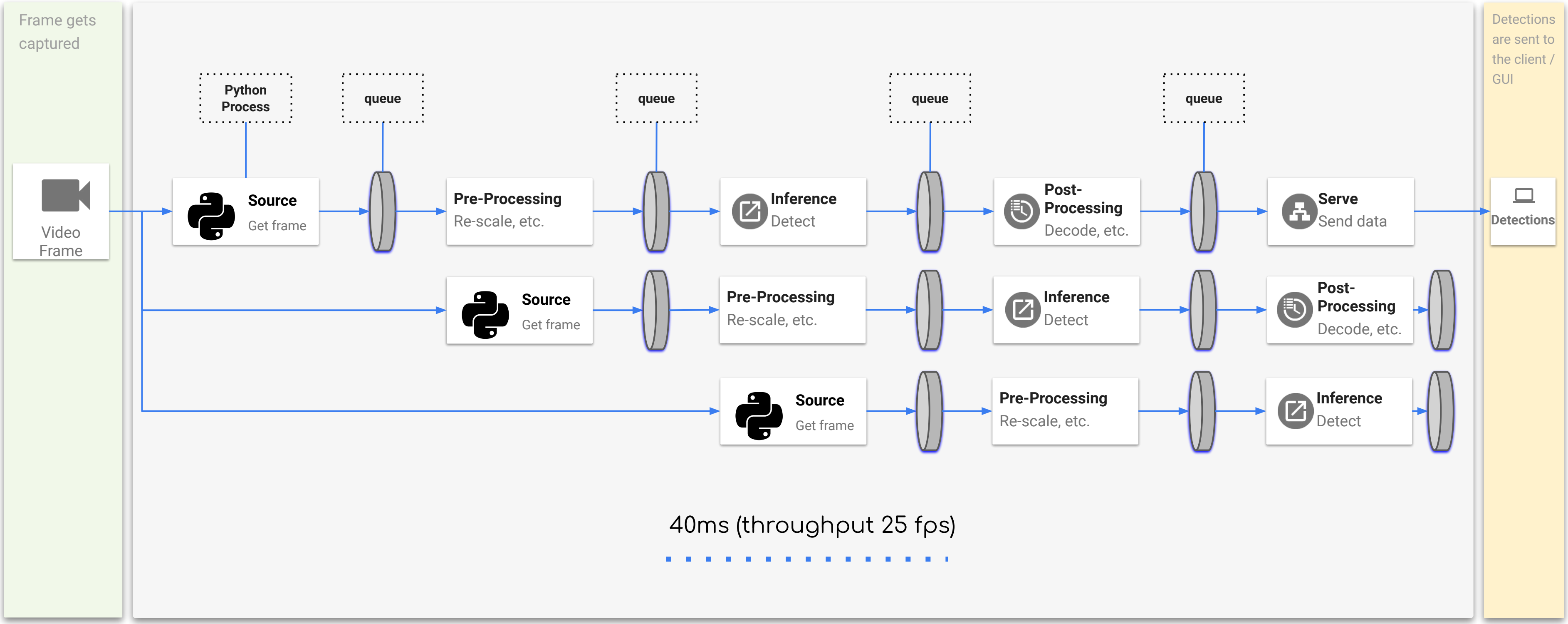
The Inference (going optimum)

· Python · Multi-thread · Initial delay: 320ms (8 frames) · Throughput: 17 fps



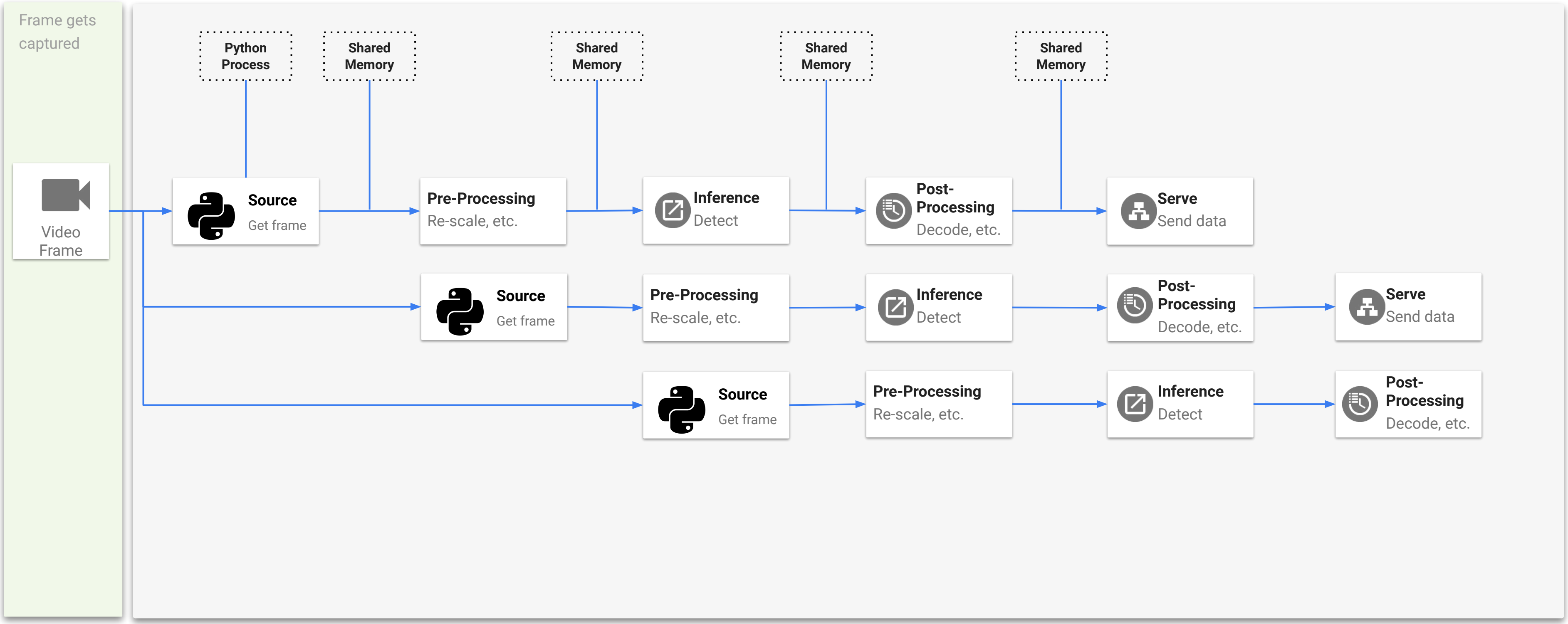
The Inference (going optimum)

· Python · Multiprocessing · Actual parallelism · Big overhead in communications



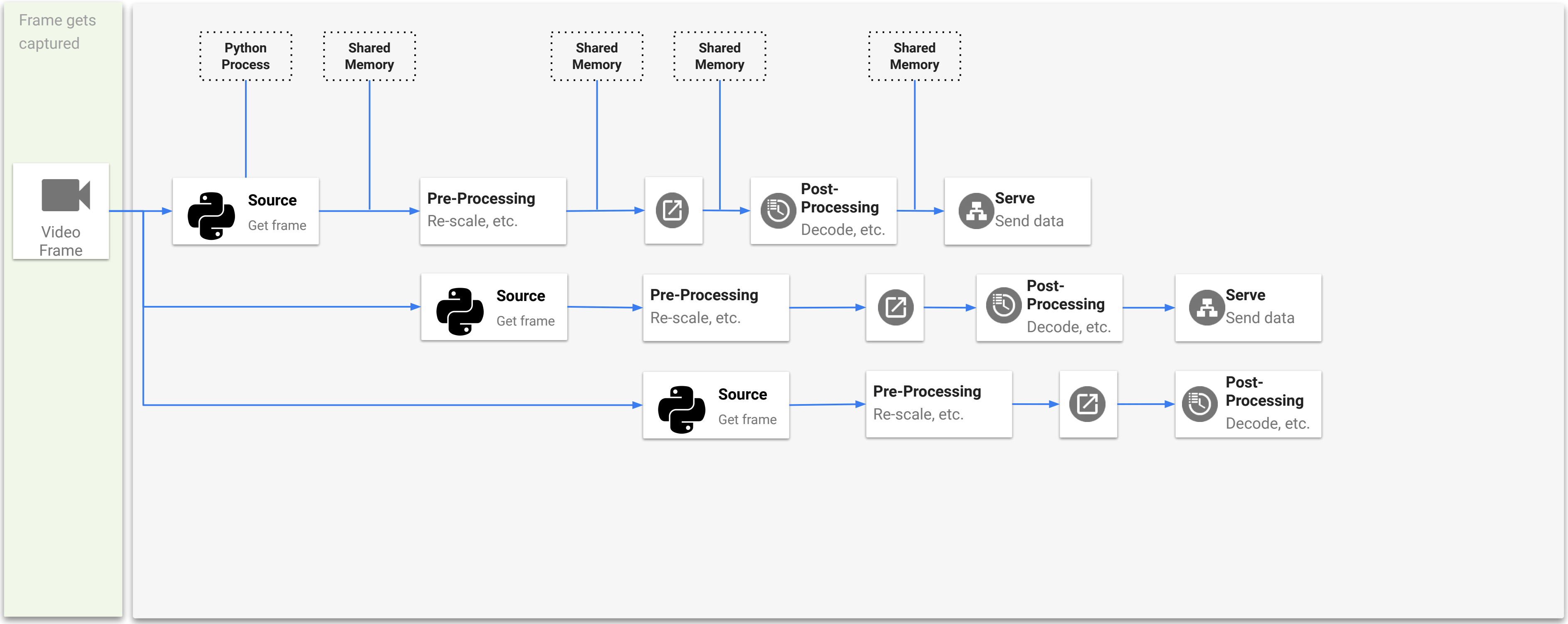
The Inference (going optimum)

· Python · Multiprocessing · Shared memory buffers



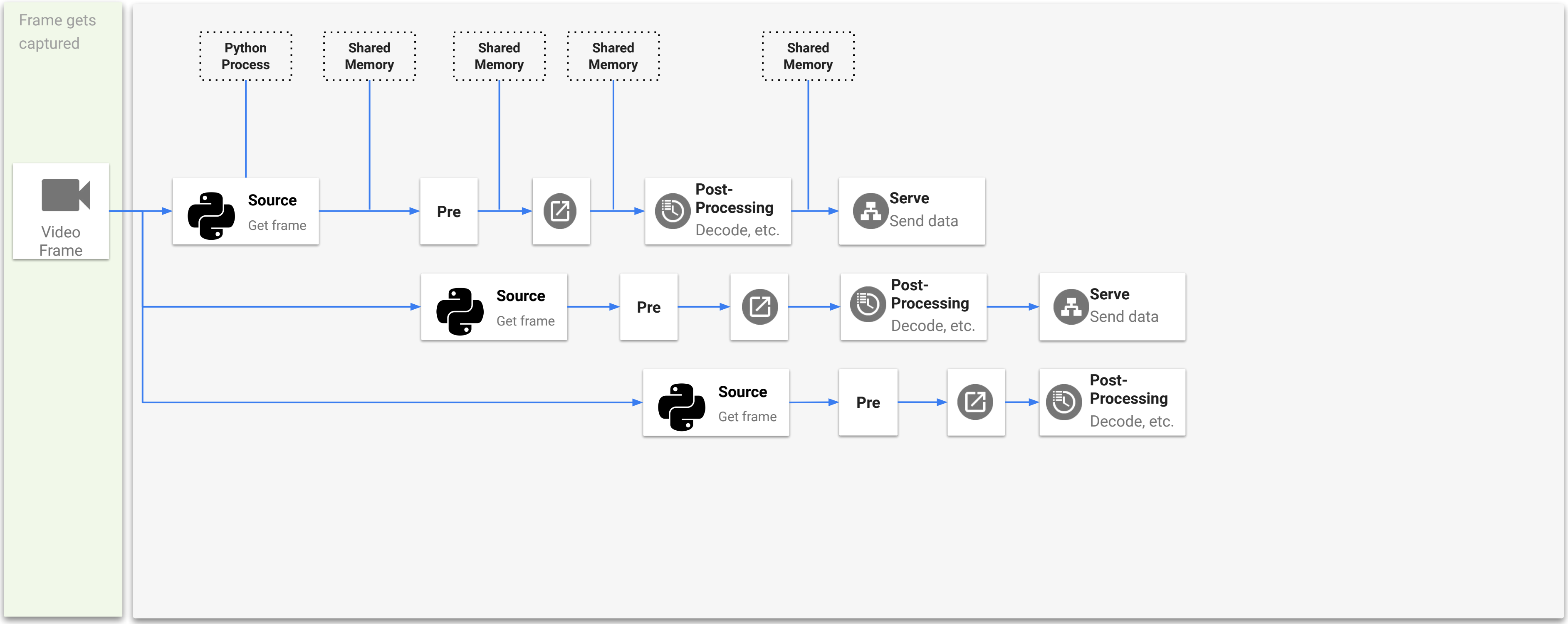
The Inference (going optimum)

Inference: C++ bindings & TensorRT



The Inference (going optimum)

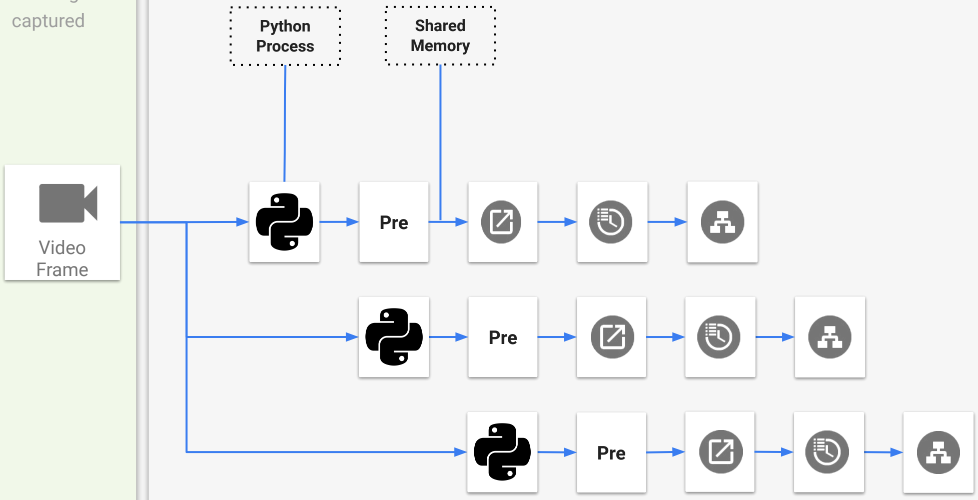
Pre-Processing: C++ bindings & optimize operations



The Inference (going optimum)

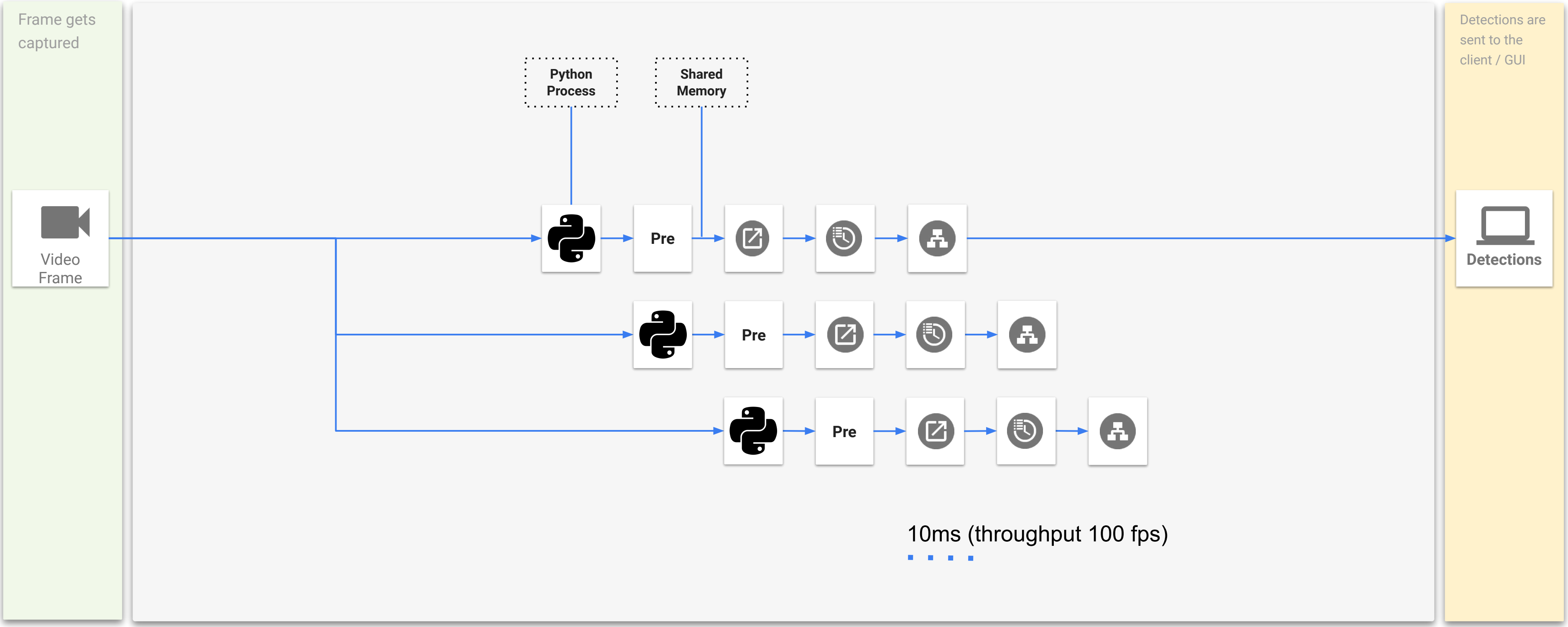
Post-Processing: C++ bindings & vectorize

Frame gets captured



The Inference (going optimum)

· Initial delay: 30ms (< 1 frame) · Throughput: 100 fps

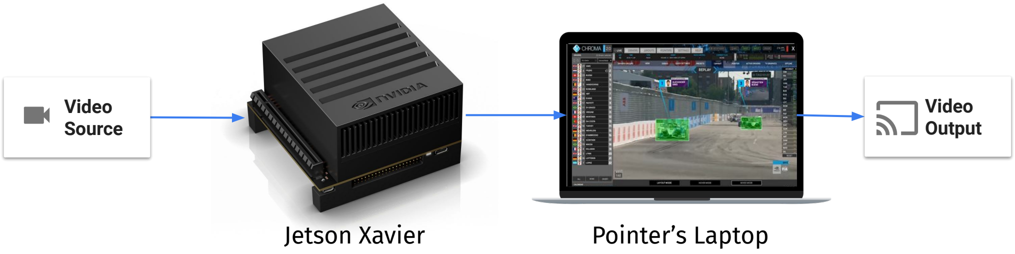


The Inference (going optimum)

```python
def inference(conf, lock, barrier,...):

    """Reads from inference queue, do inference, queue into postprocessing queue"""

    # Synchronism at the beginning all processes should be initialized
    # before going into the main loop
    lock.acquire()

    # Three possible engines: keras, TensorFlow (executing a frozen graph
    # from keras directly in tf), TensorRT
    if conf['inferencer']['engine'] == 'TRT': # TensorRT
        ...

        lib = load_lib(conf['TRT']['lib'])
        manager = RawNNRT(lib,...)

        def predict_tensorrt(model, batch_input):
            model.run_network(batch_input)

        predict = predict_tensorrt

    elif conf['inferencer']['engine'] == 'TF': # TensorFlow
        ...

    else: # keras
        ...

    with manager as model:
        ...

        # Wait forever for new queue elements
        while True:
            ...

            batch_output = predict(model, batch_input)
    ...
```

```python
# based on nvidia's reimplementation of pjreddie's darknet

from ctypes import *
...

def load_lib(libpath="libnn_raw_wrapper.so"):
    return CDLL(libpath, RTLD_GLOBAL)

class RawNNRT:
    """
    Stateful wrapper on the RT C++ inferencer so it can be used in a "with" statement
    """
    def __init__(self, lib, ... ):
        self.lib = lib
        ...

    def __enter__(self):
        """initialize the system at the beginning of the "with" statement"""

    def __exit__(self, t, v, tb):
        """destroy the system when getting out of the "with" statement"""

    def run_network(self, img):
        """img must be an array of float32, in format (batchSize, width, height,
        nchannels), all of these being members of self.raw[0]"""

        self.raw[0].base.input_data = img.ctypes.data_as(c_float_p)
        if self.callback_buffer_free:
            self.start_infer_raw(self.raw)
            self.callback_buffer_free()
            self.end_infer_raw(self.raw)
        else:
            self.infer_raw(self.raw)
        if self.buffer_owner:
            self.copy_output(self.raw_results)
            results = [[x[0].transpose((1,2,0))] for x in self.raw_results]
            return results

...
```

The Inference - The results



Reduced Form Factor Setup

Highly portable (for hostile environments), real time: **25 FPS**
Inference delay: **3 frames**
Output: fill & key signals

Pointers v3 Setup

Very powerful: **100 FPS**
Inference Delay: **0 frames**
Output: mixed video (clean + graphics) or fill & key signals

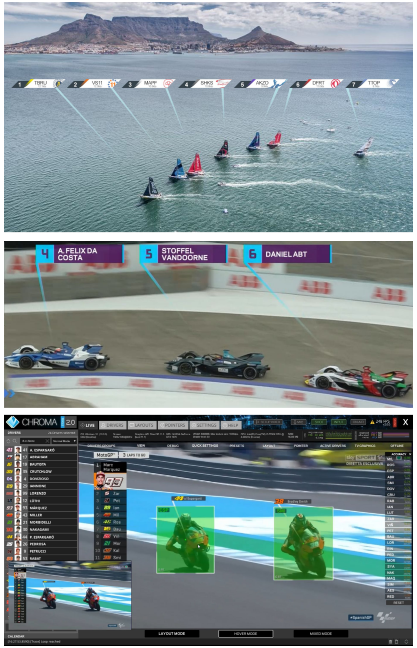
The API interface - Human configuration

Inference server

User Interface

SDI

Output graphics




```
...
2020-03-04 14:02:30,269 core send_data_to_client [INFO] Client accept timed out
2020-03-04 14:02:30,269 inferencer server [INFO] Frame 2057 throughput[FPS] 62.50  delay[s] 0.02  srcL 0.015588  srcW 0.015588
    preL 0.015588  preW 0.000000  infL 0.015588  infW 0.007581  posL 0.015588  posW 0.000000  DAM(1.0), SHT_CHG = False
2020-03-04 14:02:30,301 core send_data_to_client [INFO] Client accept timed out
2020-03-04 14:02:30,301 inferencer server [INFO] Frame 2058 throughput[FPS] 31.98  delay[s] 0.03  srcL 0.015717  srcW 0.015717
    preL 0.015717  preW 0.000000  infL 0.031271  infW 0.007588  posL 0.031271  posW 0.000000  BIR(0.994), SHT_CHG = True
2020-03-04 14:02:30,316 core send_data_to_client [INFO] Client accept timed out
2020-03-04 14:02:30,316 inferencer server [INFO] Frame 2059 throughput[FPS] 62.50  delay[s] 0.03  srcL 0.015554  srcW 0.015554
    preL 0.015554  preW 0.000000  infL 0.015621  infW 0.007574  posL 0.015621  posW 0.000000  BIR(0.99), SHT_CHG = False
2020-03-04 14:02:30,332 core send_data_to_client [INFO] Client accept timed out
2020-03-04 14:02:30,332 inferencer server [INFO] Frame 2060 throughput[FPS] 62.50  delay[s] 0.03  srcL 0.015621  srcW 0.015621
    preL 0.015621  preW 0.000000  infL 0.015621  infW 0.007584  posL 0.015621  posW 0.000000  BIR(0.984), SHT_CHG = False
...
```

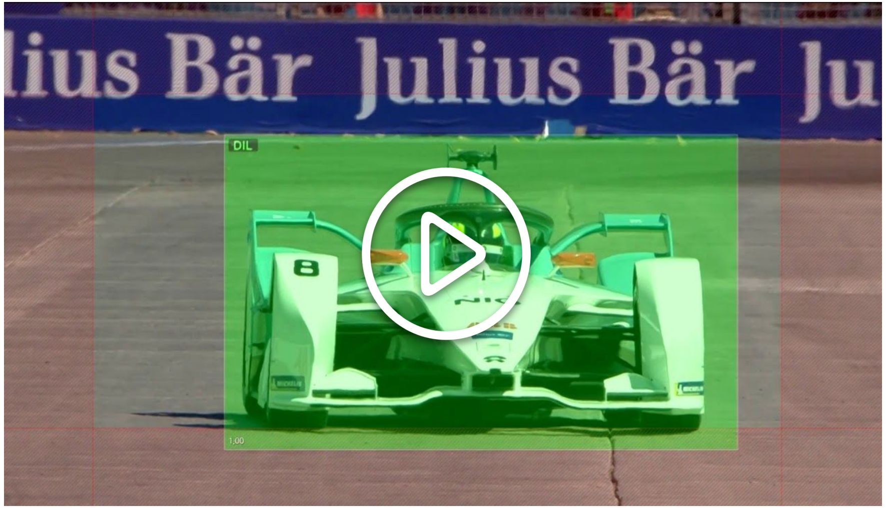
The User Interface

Made with unity

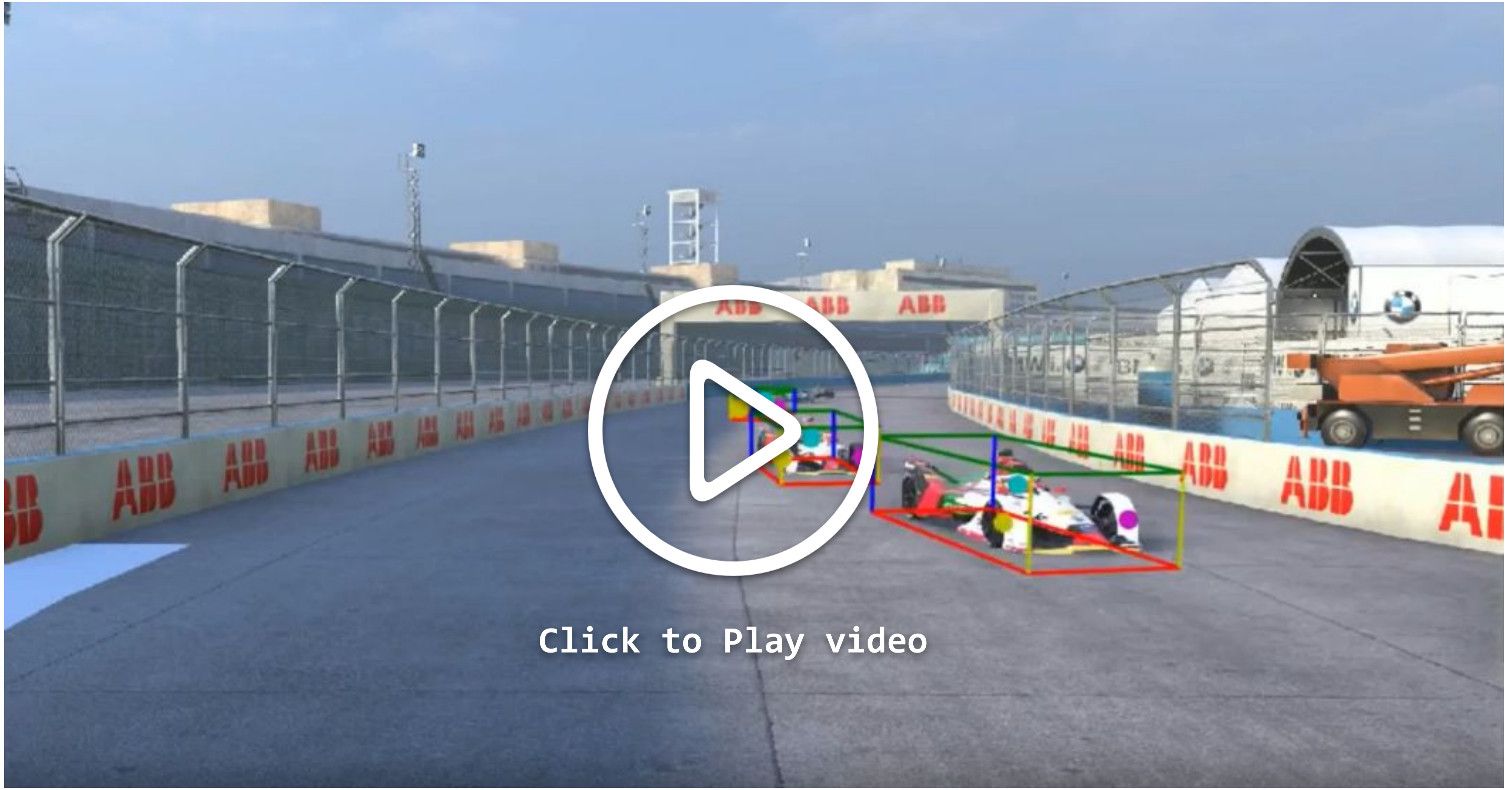
The Results

Tags on TV

Chroma Tools



Next generation of synth training and inference



Absolute Positional Tracking: object geometry and components of the car

Thank you!

JESÚS HORMIGO
CTO

jesus@virtuallylive.com
@jesushormigo

DAVID ALBARRACÍN
Lead Research Engineer

dalbarracin@virtuallylive.com

GTC GPU TECHNOLOGY CONFERENCE
SILICON VALLEY
MARCH 2020