ETA: Extensible Toolkit for Video Analytics

PSCR Public Safety Broadband Stakeholder Meeting

Jason Corso, PhD
jason@voxel51.com

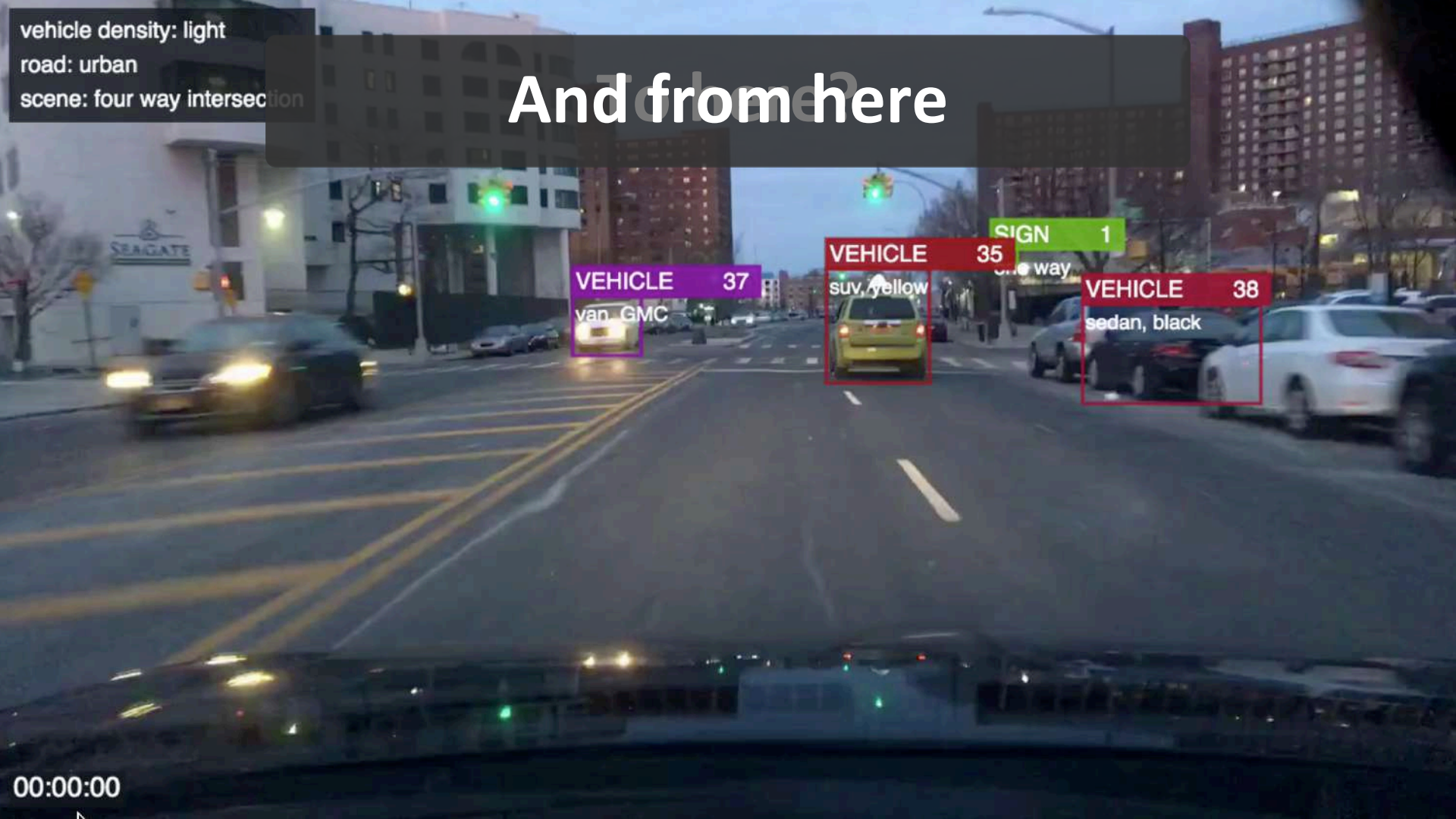
Brian Moore, PhD
brian@voxel51.com

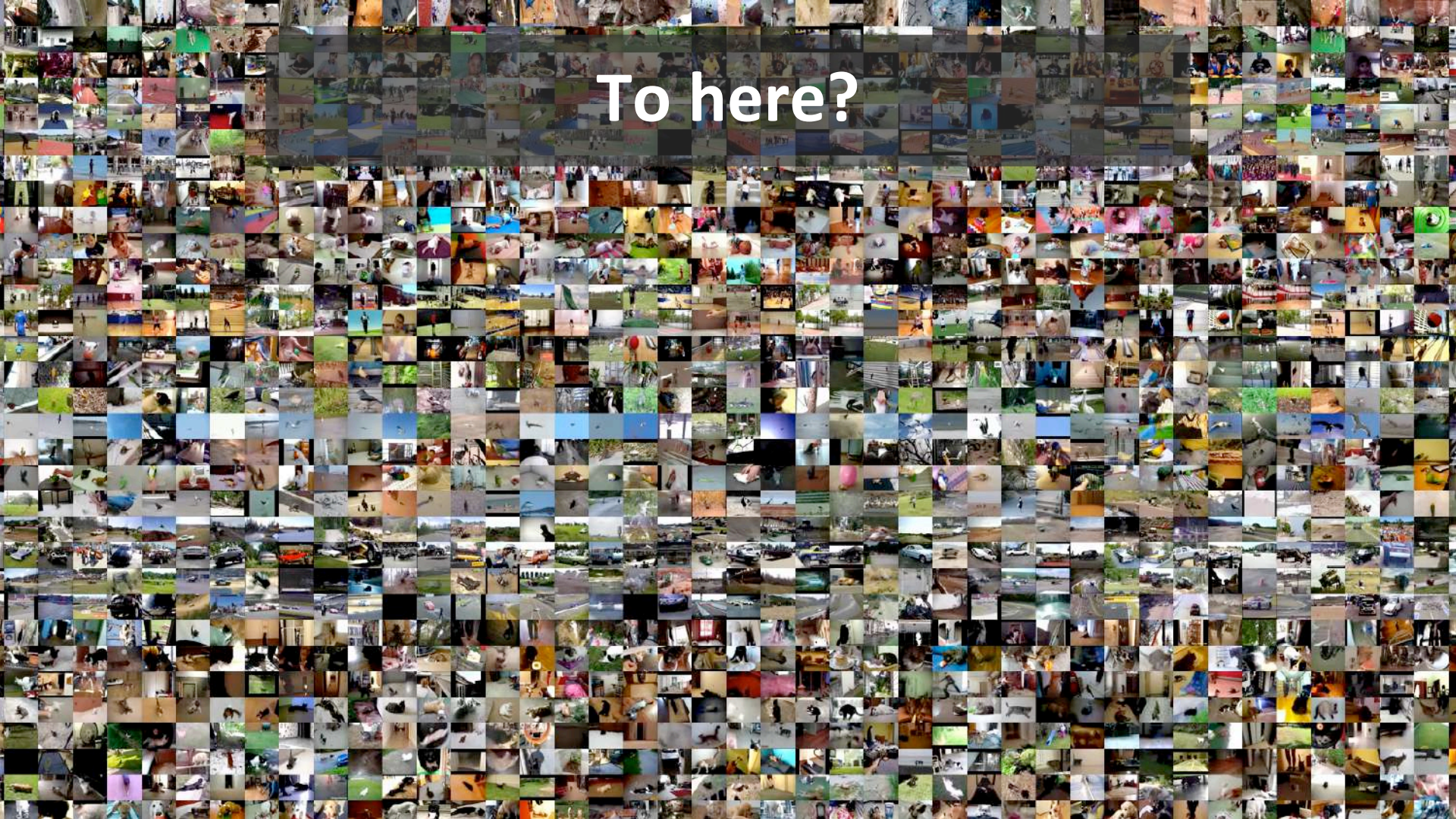
DISCLAIMER

This presentation was produced by guest speaker(s) and presented at the National Institute of Standards and Technology's 2019 Public Safety Broadband Stakeholder Meeting. The contents of this presentation do not necessarily reflect the views or policies of the National Institute of Standards and Technology or the U.S. Government.

Posted with permission

How do we get from here


vehicle density: light
road: urban
scene: four way intersection
And from here
VEHICLE 37
van, GMC
VEHICLE 35
suv, yellow
SIGN 1
VEHICLE 38
sedan, black
00:00:00


To here?

By creating an

open-platform for

video analytics

NIST Grant: The ETA Project

The **Extensible Toolkit for Analytics (ETA)** is our open-source library for video analytics.

Our Technical Goal

- *Establish an extensible video analytics infrastructure that will support next-generation capabilities in public safety applications.*

Our Programmatic Goal

- *Catalyze a community of innovation that brings public safety video analytics up-to-date with cutting edge computer vision research.*

What is Hard About This?

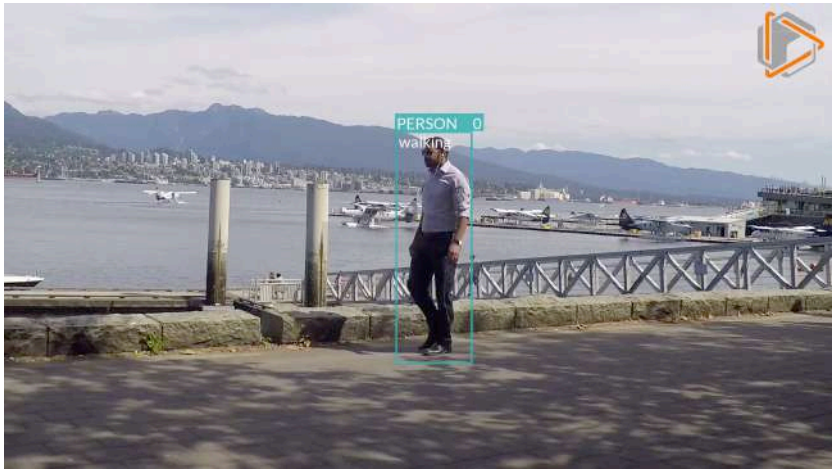
- How to leverage the state of the art in computer vision and machine learning?
- How to spread these capabilities to **broad uses** in public safety?
- How to **integrate** with existing and proprietary stacks?
- How to **scale**?
- How to **incentivize stakeholders**?

Video-First Analytics

Standard Approaches are Frame-Based

Running? Walking? Standing?

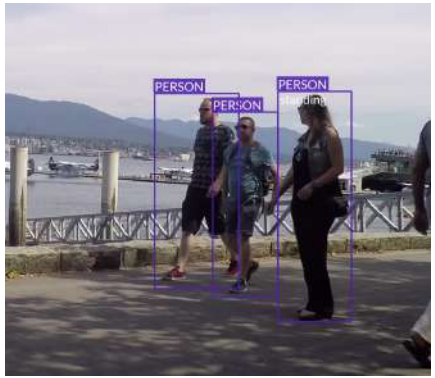
Voxel51 is **Video First**

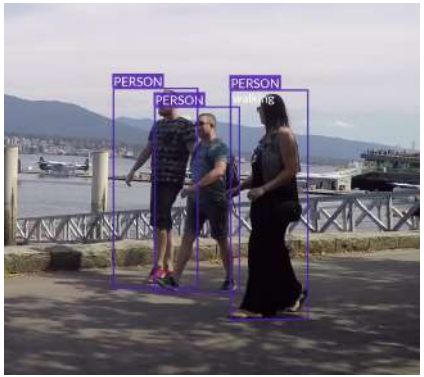


Walking!

Video-First Analytics

Standard Approaches are Frame-Based





Standing? Walking?

Voxel51 is **Video First**

Walking!

Video-First Analytics

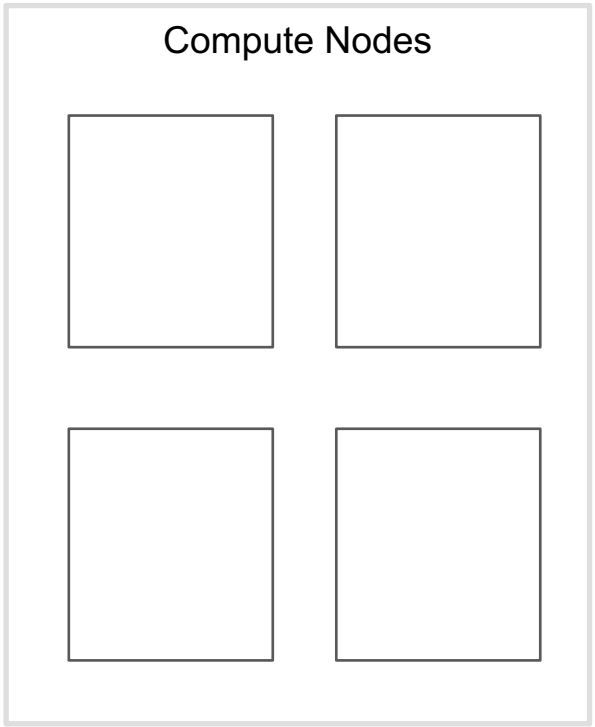
Standard Approaches are Frame-Based

Voxel51 is **Video First**

12.8% improvement

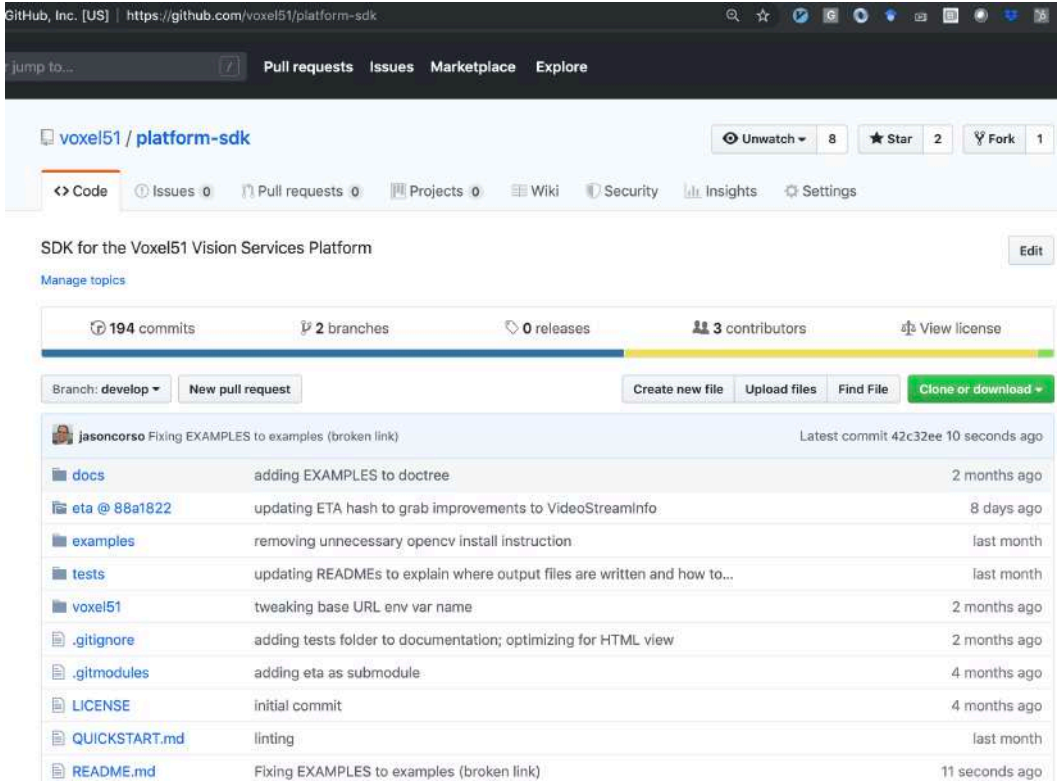
Streaming Analytics

Time



Extensible Analytics

- Platform-SDK
  - Enables third-party party analytic developers to easily deploy and integrate their analytics on our platform.
  - Directly integrates into the overall ETA infrastructure.
- Available on GitHub



Example Third-Party: Privacy-Preserving Analytic

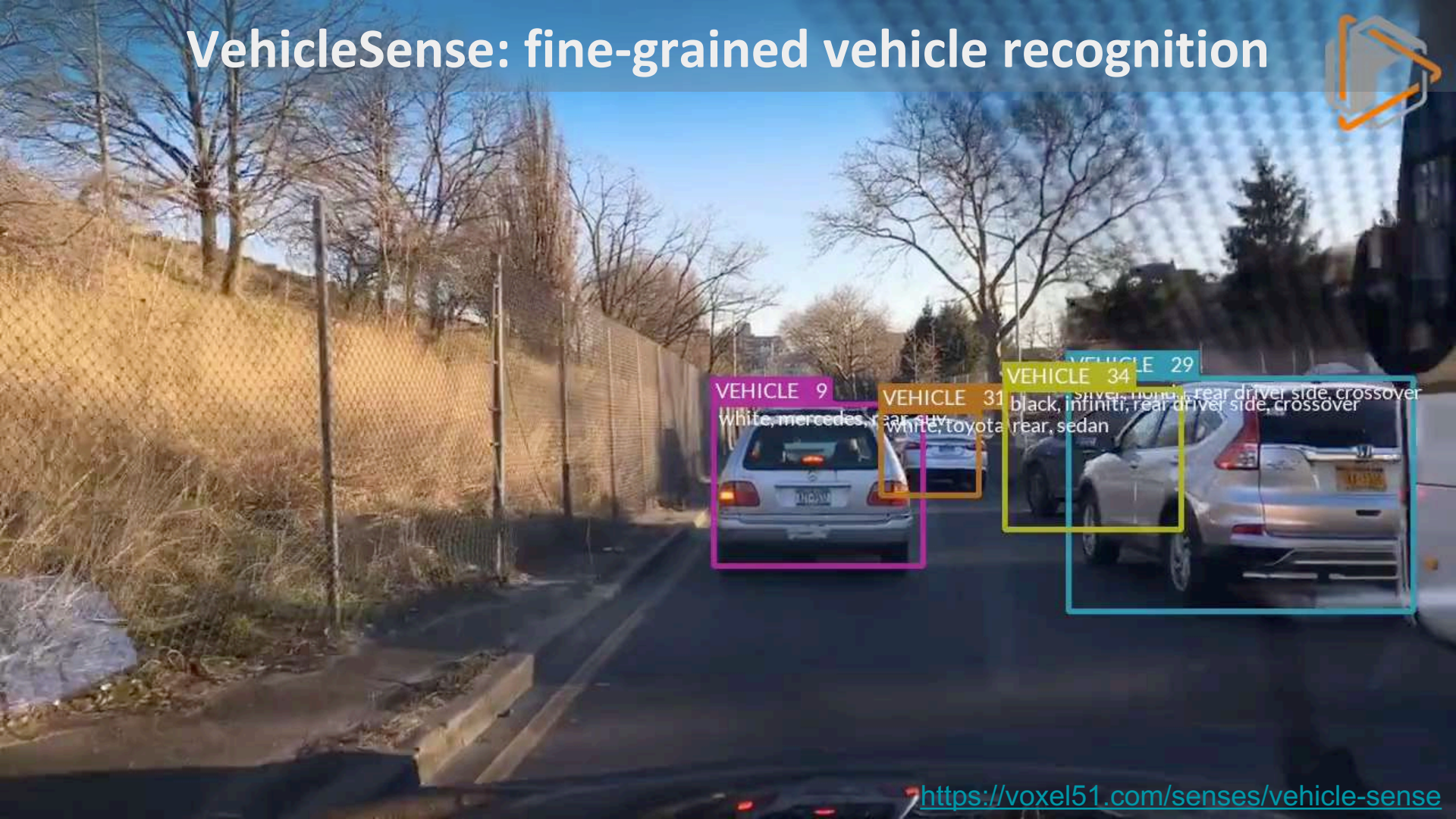
- Handle FOIA requests with automated ease.

EgoVid

Original

Anonymized Faces

VehicleSense: fine-grained vehicle recognition



https://voxel51.com/senses/vehicle-sense

RoadSense: adds road signs and road scenes

https://voxel51.com/senses/road-sense

Ground Truth: Explosion
Prediction: Explosion

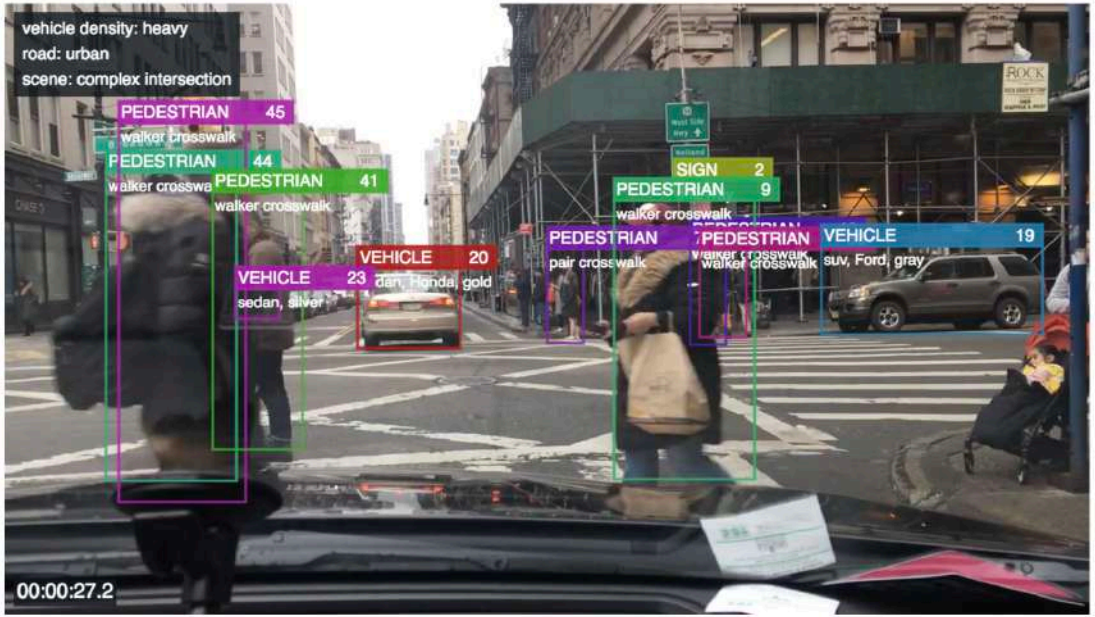
Ground Truth: theft
Prediction: theft

Video 20ddf373

Starts at 00:00:22.8

Duration: 4.1s

Reset Clip



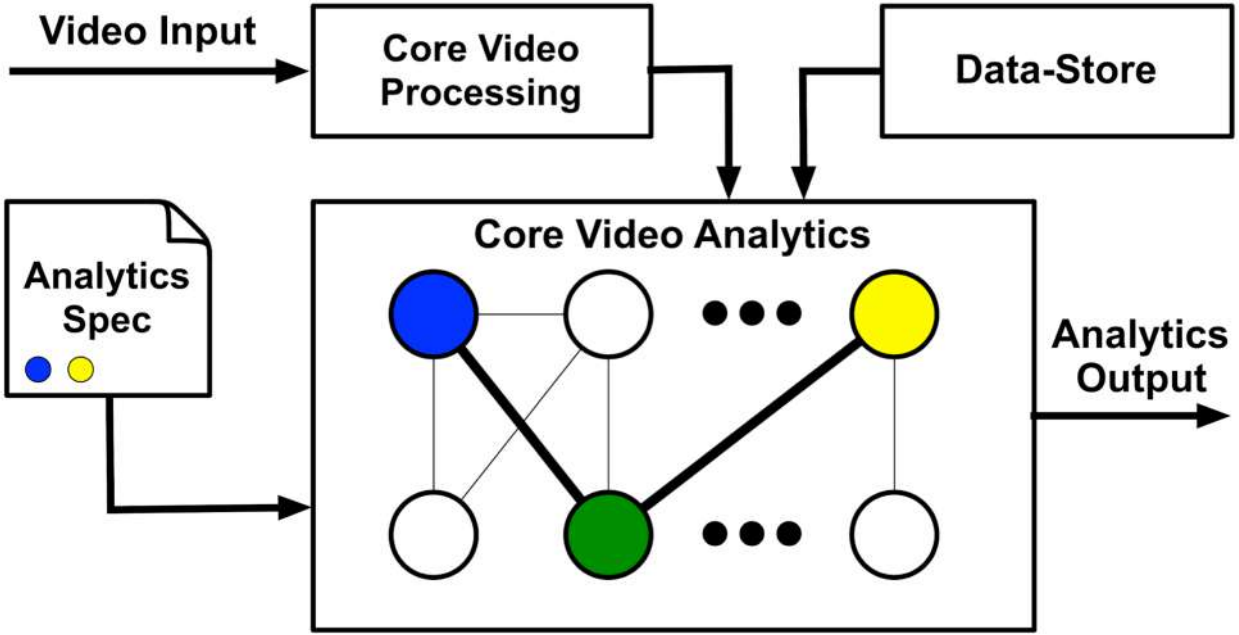
Added by: support@voxel51.com

Key: 20ddf373

Recorded: Unknown

Uploaded: 2019-01-07T20:59:43

ETA System Design



ETA System I/O

Video Input

- Users connect data asynchronously to the Voxel51 Platform

Data-Store

- Stores user-uploaded data, internal metadata, models, parameter files, and analytics outputs
- Storage is implemented as a collection of scalable buckets in **Google Cloud** or **AWS**

Analytics Output

- Returned in a simple, flexible **JSON format**
- Users can download analytics asynchronously from the Voxel51 Platform

ETA System Analytics

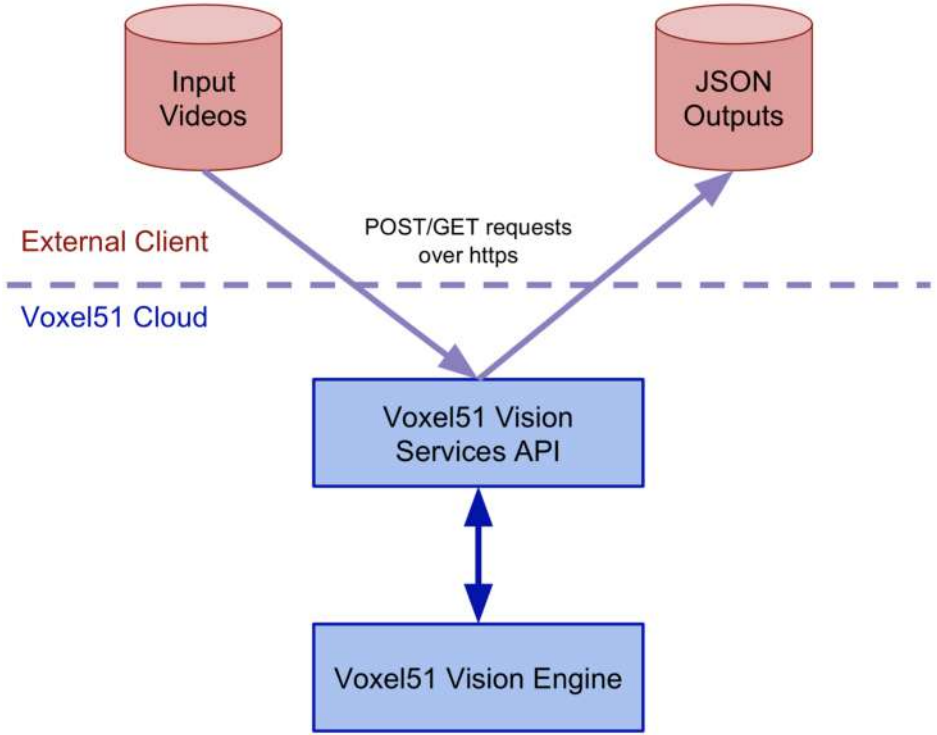
Core Video Processing

- Low-level video operations such as sampling, resizing, optical flow, and segmentation
- **Preprocessing step** for all input video
- Generates an internal representation of the video that is amenable for subsequent analysis

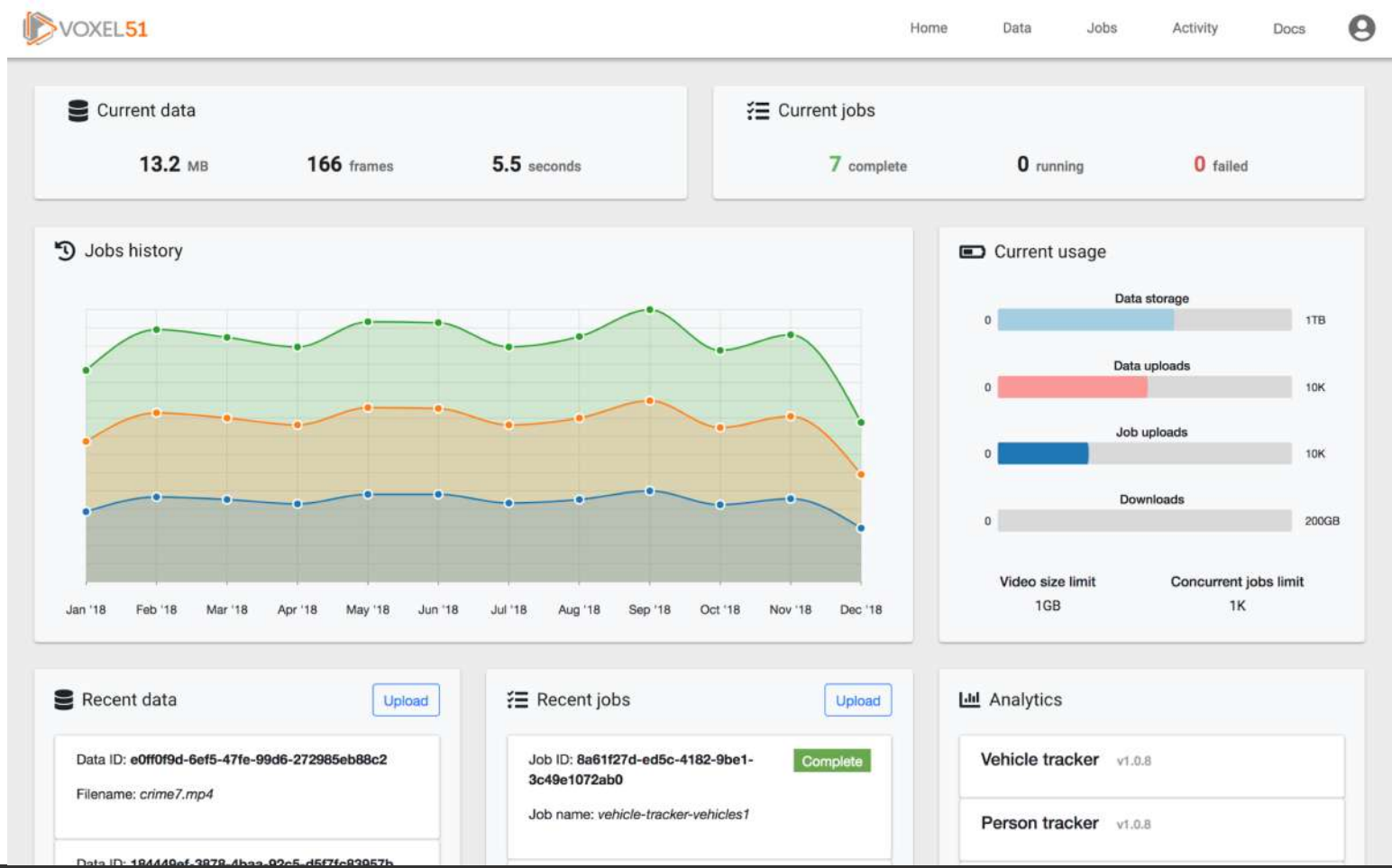
Core Video Analytics

- Implements **core computer vision analytics** such as detection, tracking, classification, and recognition
- Relies heavily on open-source libraries (OpenCV, TensorFlow, Keras, etc.) and publicly available research implementations of modern vision algorithms

API Design

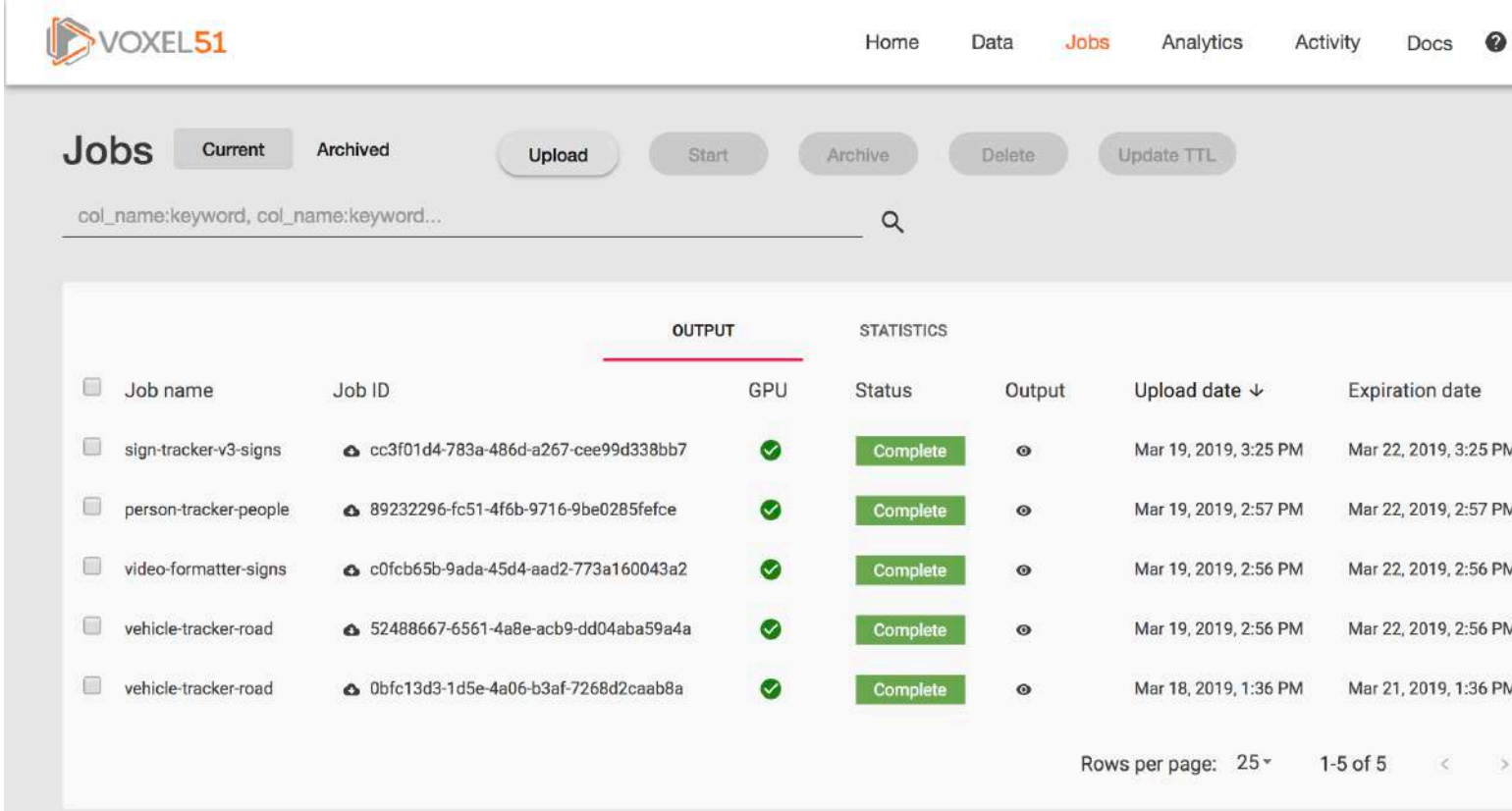

Input Videos
JSON Outputs
External Client
POST/GET requests over https
Voxel51 Cloud
Voxel51 Vision Services API
Voxel51 Vision Engine

Platform Console



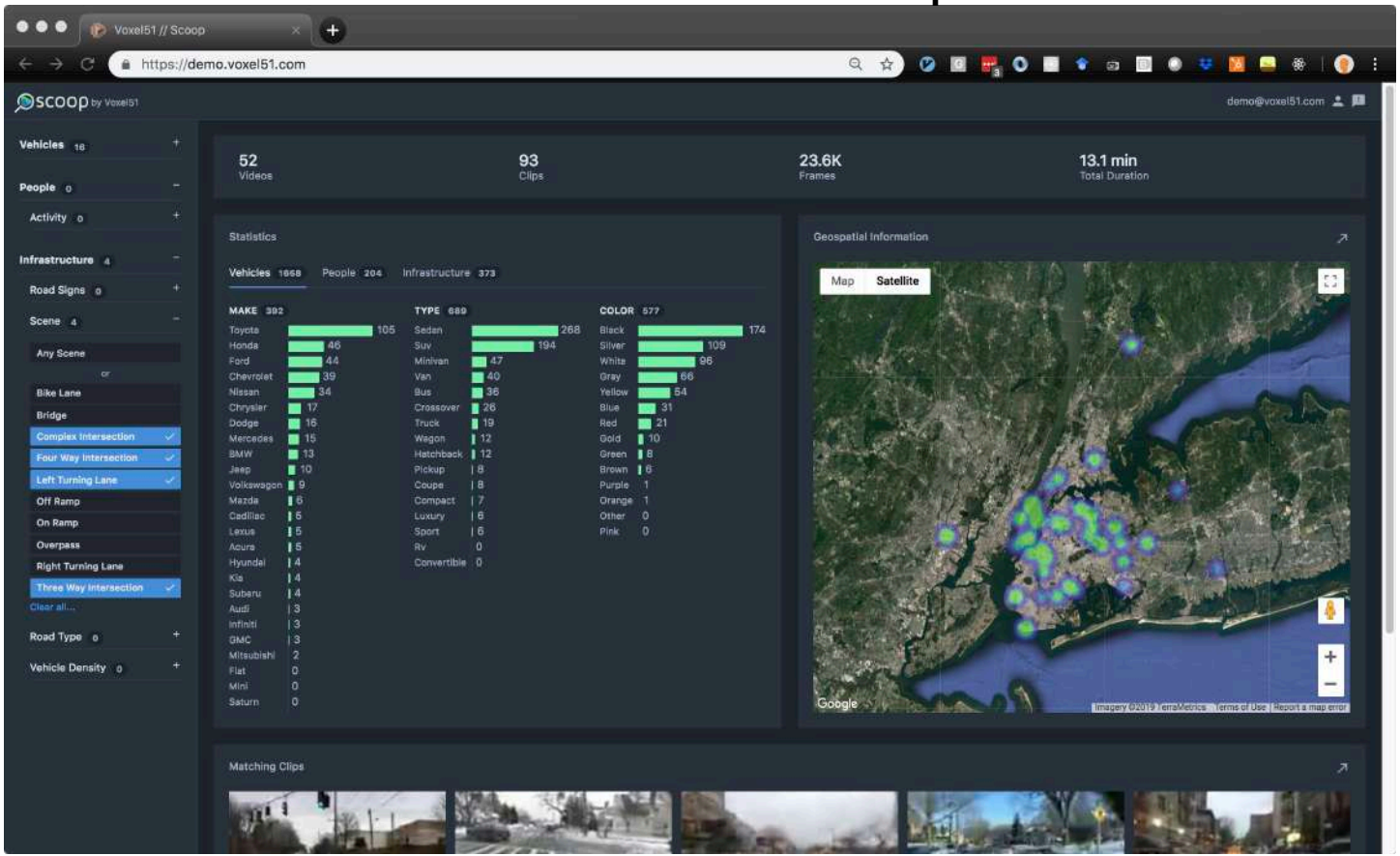
Platform Console

Jobs Board



Scoop pulls this all together

https://voxel51.com/demo



Faceted Search

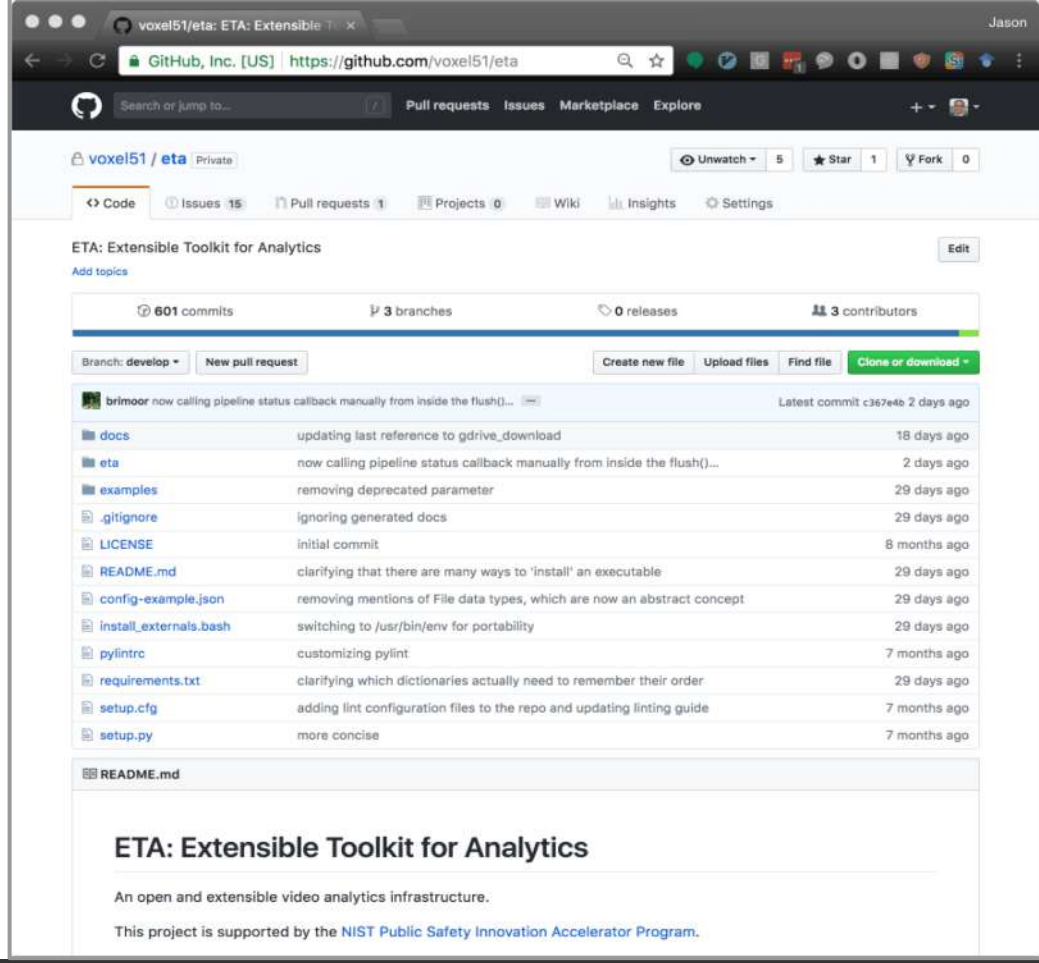
- Direct access to ontology
  - Search meaningful
  - Create complex logical queries
- Labels schema
  - Schema enforced by system
  - User-uploadable
  - Automatically generated (later)
- Backend builds an index of content and returns slice of dataset in real-time
- Let's you drill into specific concerns

Insights

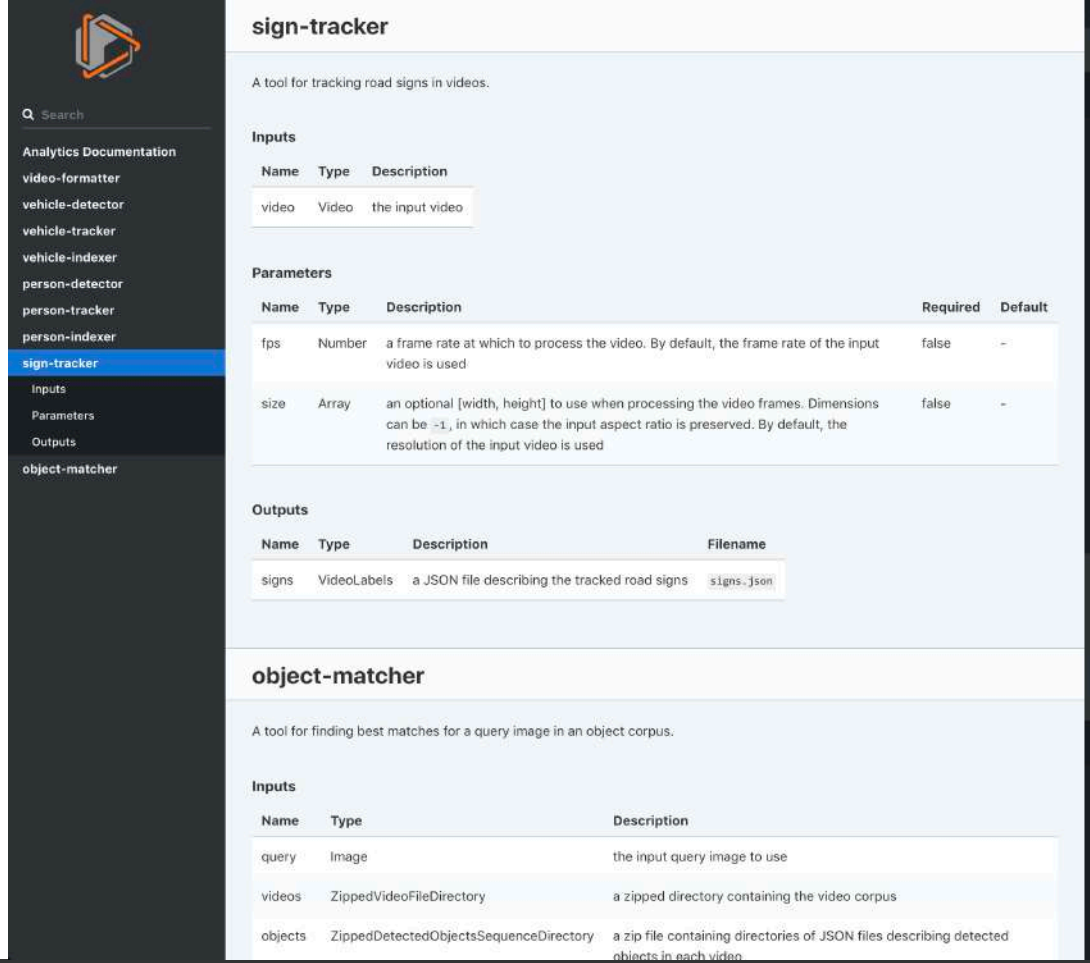
- Backend index provides faceted-search results
- Summarized
  - Occurrence statistics across ontology
  - Co-occurrence statistics with facet
  - Spatial information
- All individual resources can be studied independently as well

Open-Source

- ETA – Vision Engine Code
- API-PY – Python API Client Library
- API-JS – Javascript API Client Lib.
- Platform-SDK – Wrap your custom vision methods to be deployed on the platform at scale.
- Coming Soon!
  - Player51 – State of the art Javascript video player
- All at https://github.com/voxel51



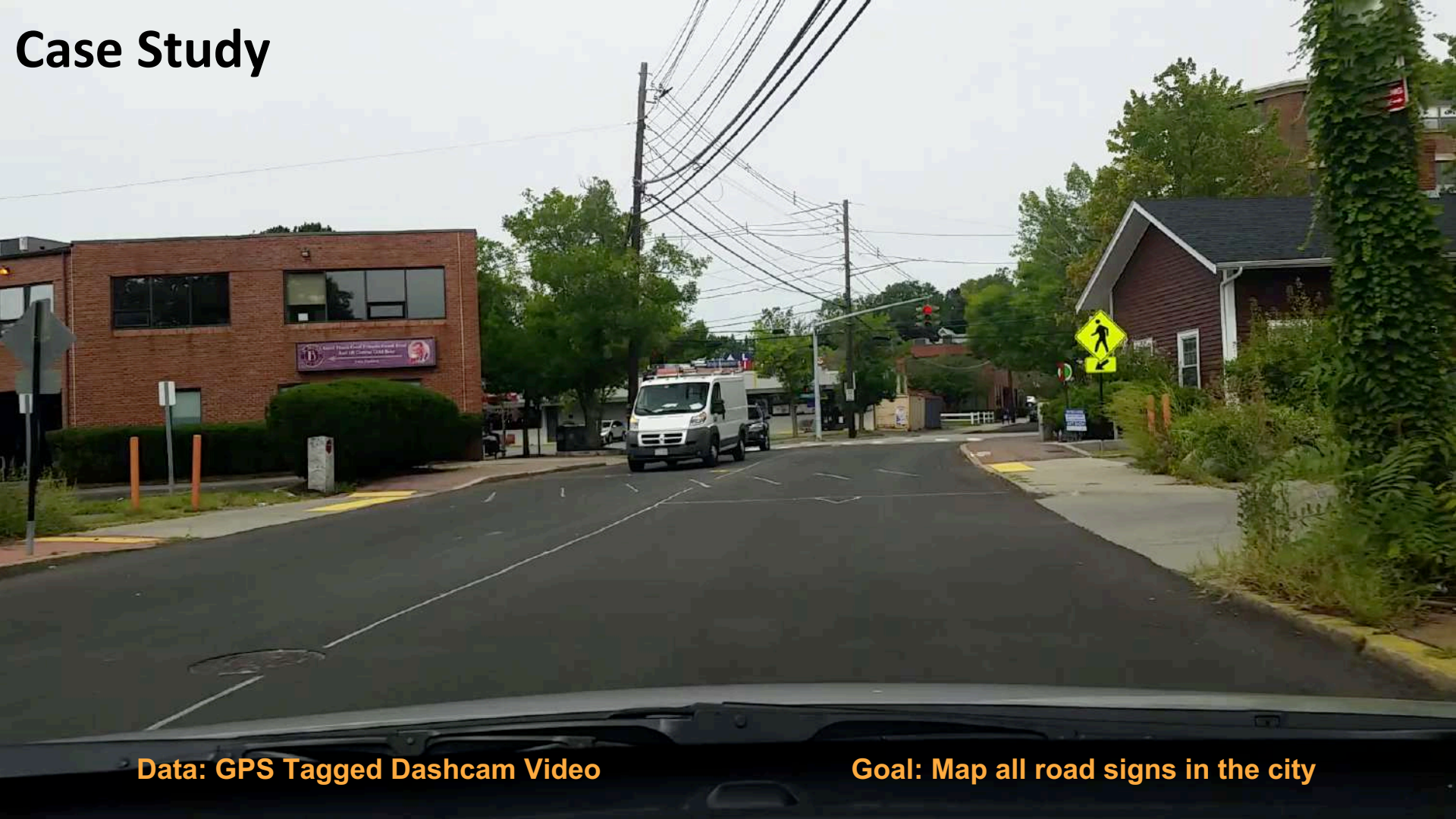
Platform Docs



Case Study: Mapping Smart Cities

https://voxel51.com

Case Study



Data: GPS Tagged Dashcam Video

Goal: Map all road signs in the city

Existing Frame-based Methods

Many spurious detections

Unusable for mapping

Voxel51 Video-Based Model

More accurate

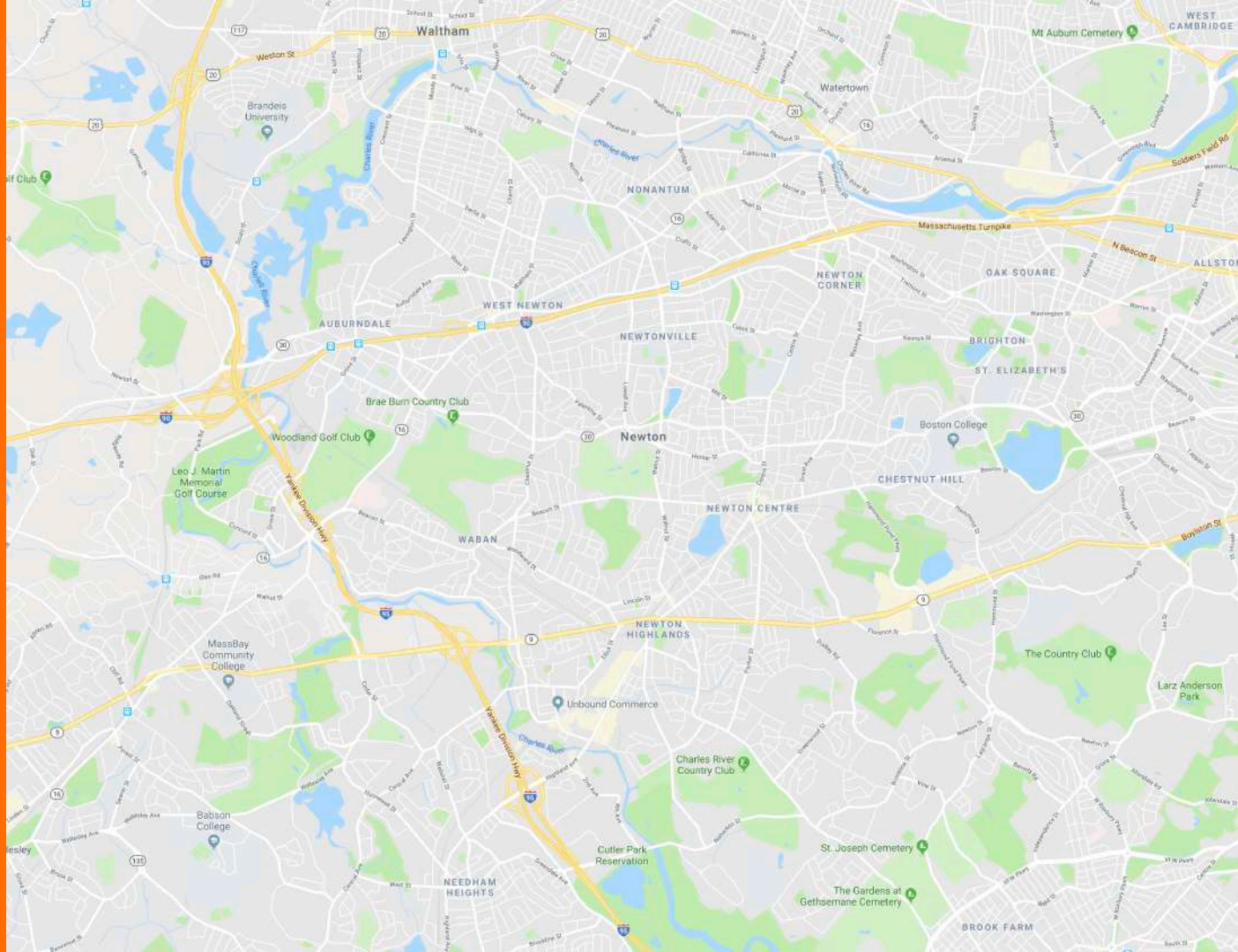
Enables mapping

Newton, MA

2.6 Days of Video

Few Hundred Miles

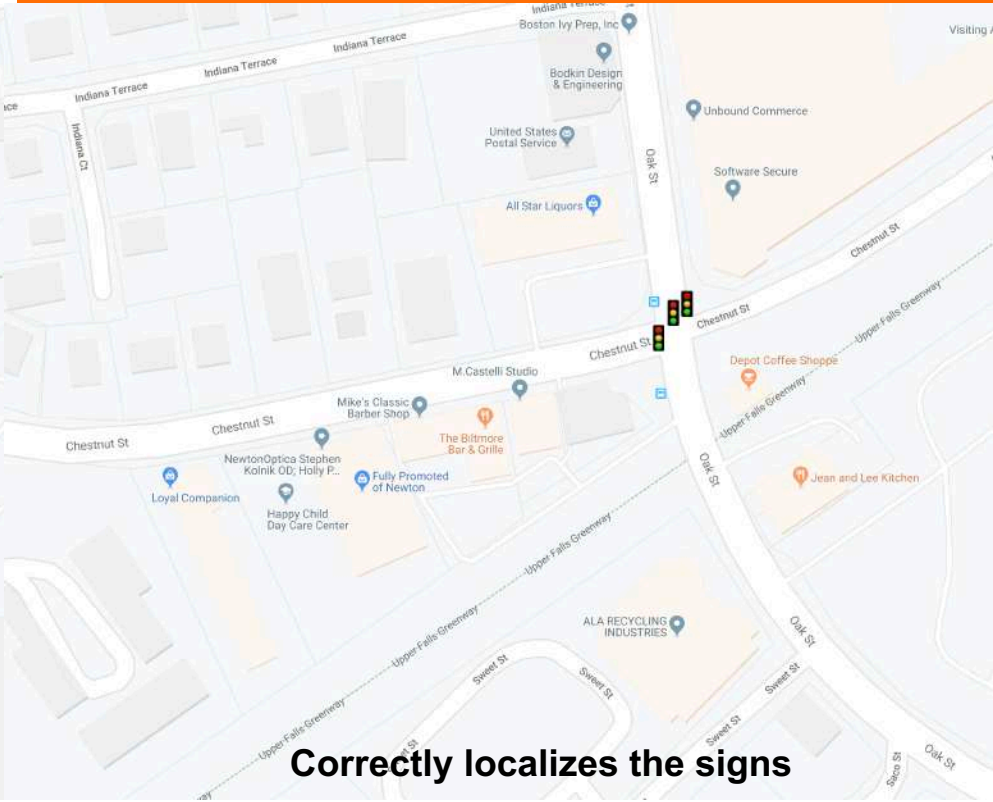
10 Hours of Compute



Frame-Based Models

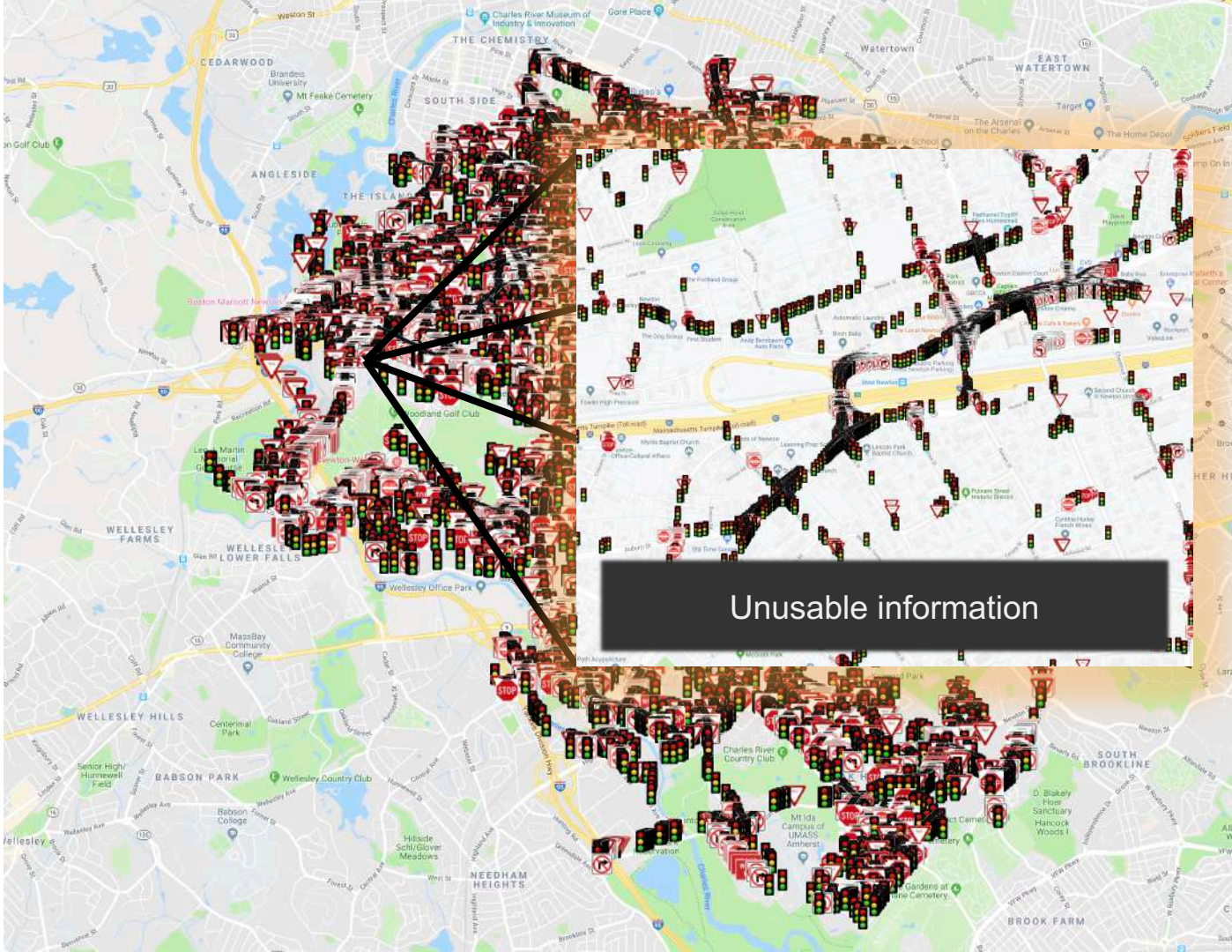
Voxel51 Video-Based Model

Unable to determine number and location of signs

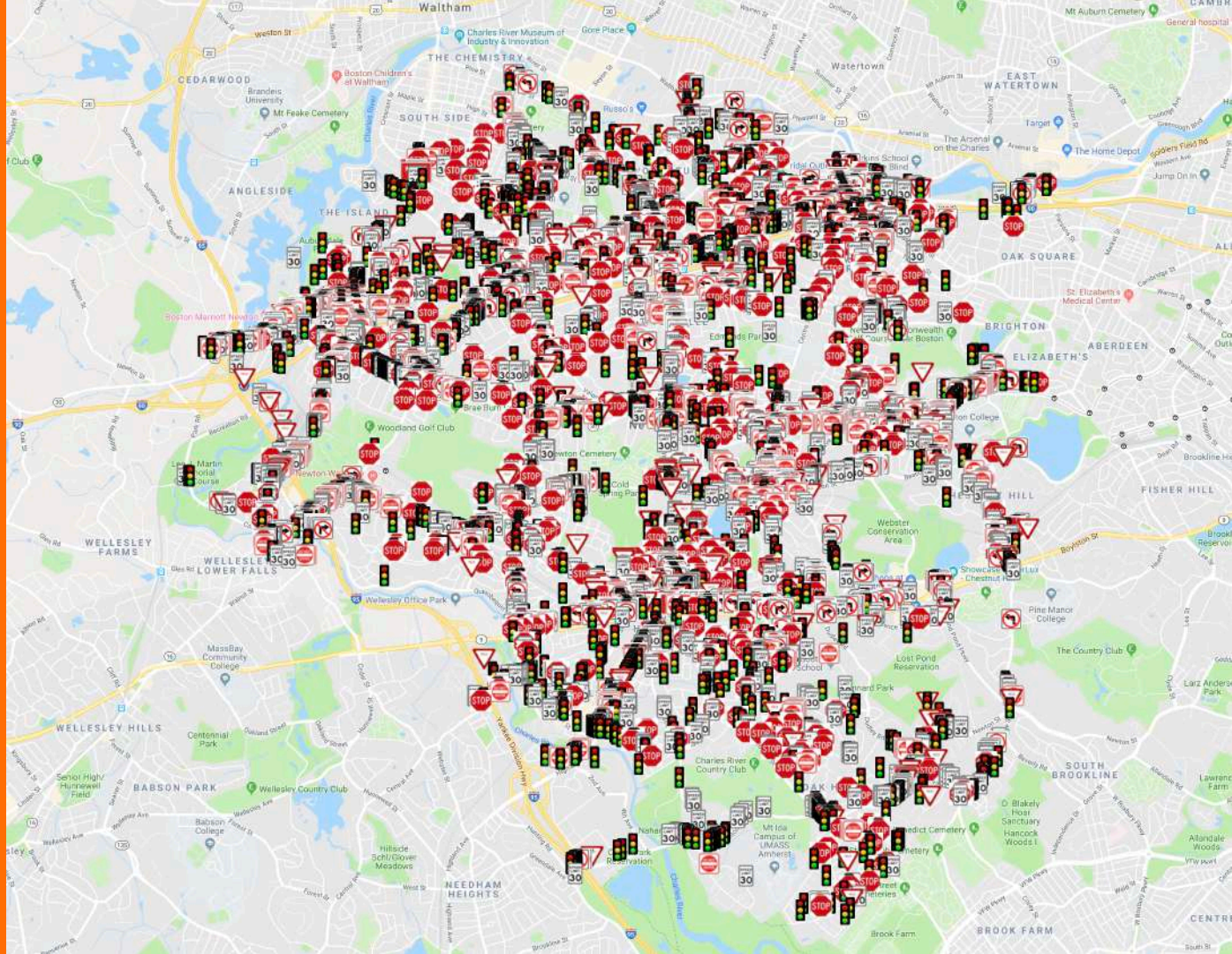


Correctly localizes the signs

Frame-Based Road Sign Map



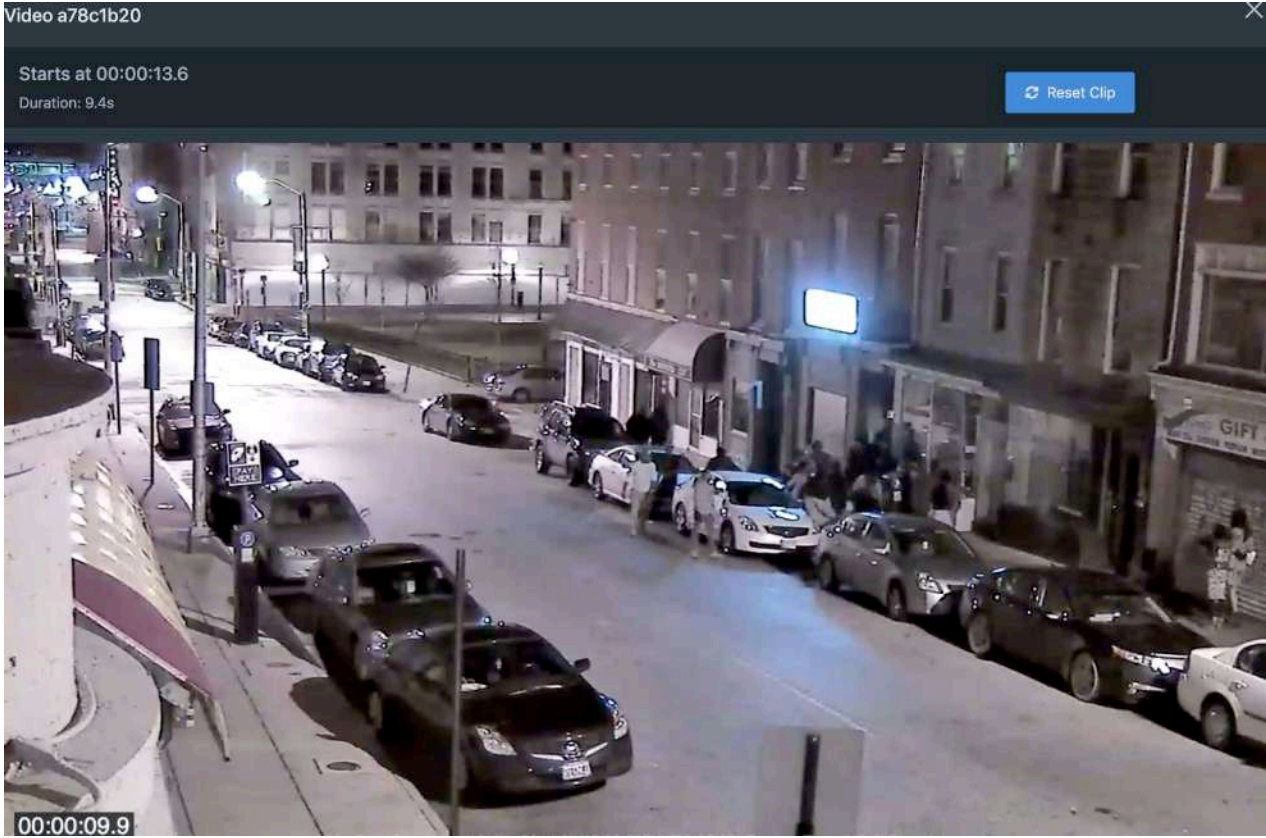
Voxel51
Road Sign Map



Case Study Teaser: Baltimore CitiWatch

https://voxel51.com

Fight Detected



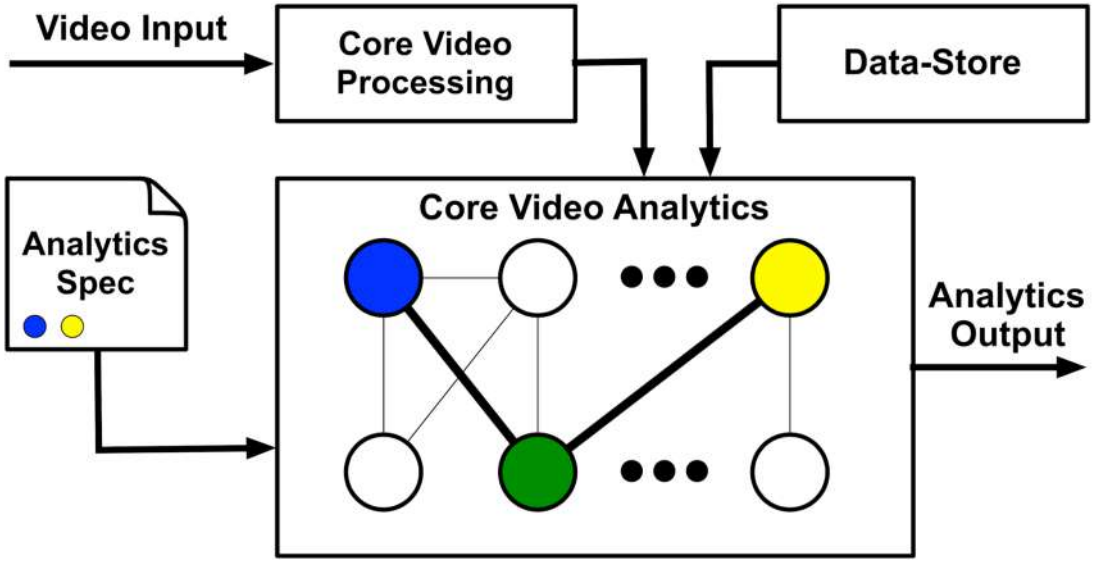
solutions@voxel51.com
https://voxel51.com/demo


Come back for the
Next
Session
2:25 PM

Backup

Deployment Strategy

The ETA Library

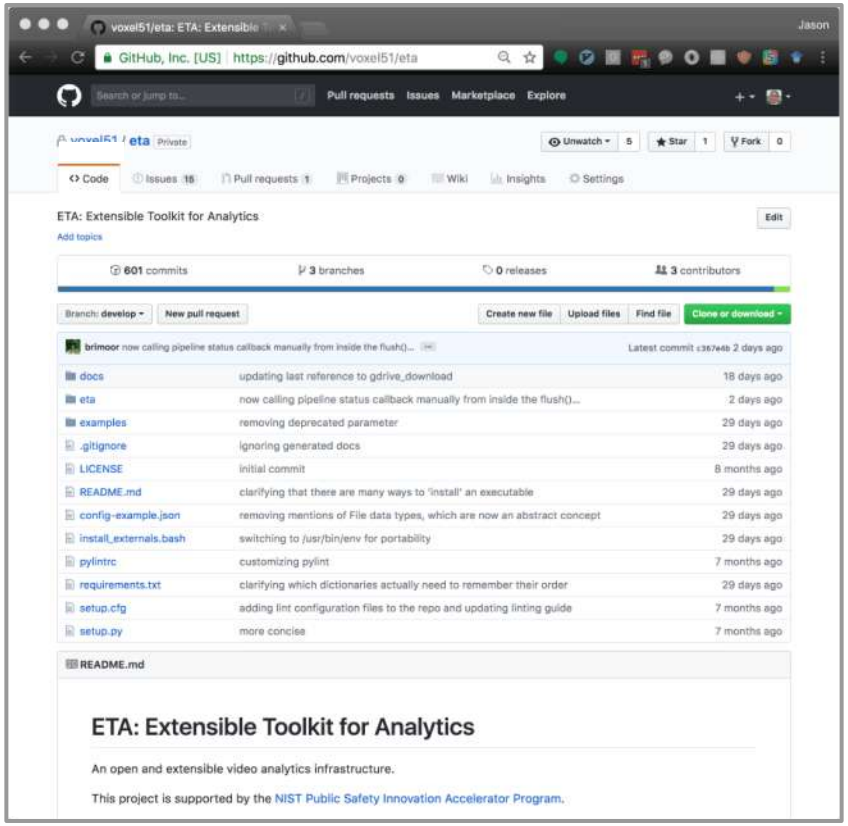


ETA Library

- **Modular** software infrastructure
- Core infrastructure implemented in Python to maximize accessibility to the vision community
- Modules are **generic** and can be implemented in **arbitrary languages** (e.g. C++) and even hosted remotely in third-party clouds
- Modules are connected to form pipelines that are run to compute the requested analytics
- Analytics are stored in a simple, flexible **JSON format**

The ETA System is a dynamically configurable graph whose nodes are modules and whose edges represent data flowing through the system

Open-Source on GitHub!



ETA Modules

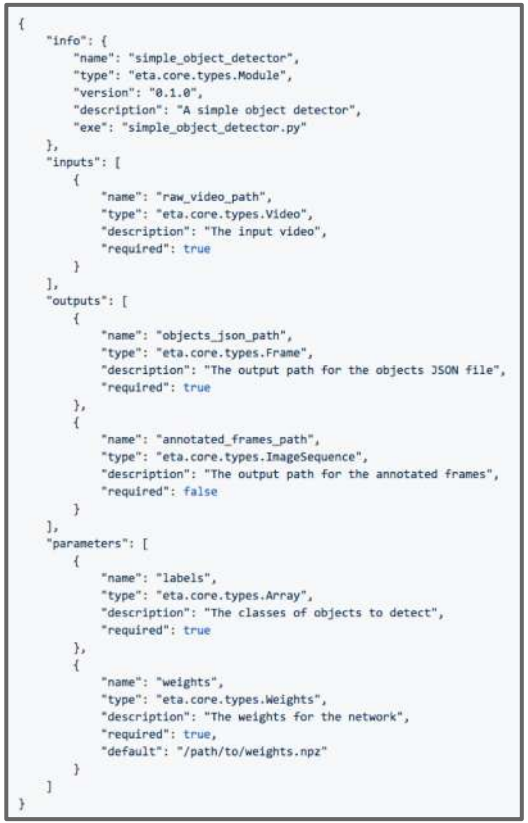
- ETA modules are simply executables that take JSON files as input and write their outputs to disk

- Input **configuration JSON** files specify:
  - The locations of the *input* data on disk to process
  - The *parameter values to use*
  - The locations on disk to write the *output* data

- The ETA library defines a rich **type system** that all modules use to declare their I/O types
  - Allows modules to communicate seamlessly

Module Metadata Files

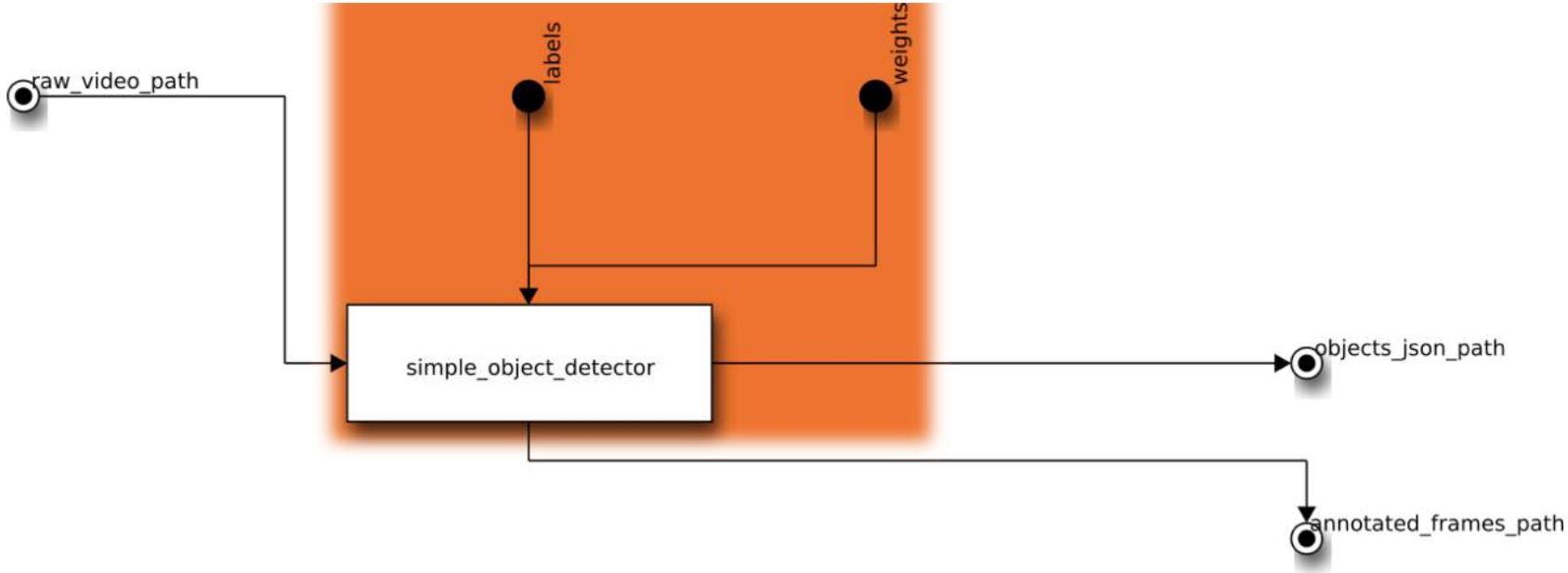
- Each module provides a **metadata JSON** file
  - Enumerates the module's inputs, outputs, and parameters
  - Defines the type of each field
- Complete description of a module's operation
- Enables modules to be implemented in any language
- **Generated automatically** from source for modules built on ETA



```
{
    "info": {
        "name": "simple_object_detector",
        "type": "eta.core.types.Module",
        "version": "0.1.0",
        "description": "A simple object detector",
        "exe": "simple_object_detector.py"
    },
    "inputs": [
        {
            "name": "raw_video_path",
            "type": "eta.core.types.Video",
            "description": "The input video",
            "required": true
        }
    ],
    "outputs": [
        {
            "name": "objects_json_path",
            "type": "eta.core.types.Frame",
            "description": "The output path for the objects JSON file",
            "required": true
        },
        {
            "name": "annotated_frames_path",
            "type": "eta.core.types.ImageSequence",
            "description": "The output path for the annotated frames",
            "required": false
        }
    ],
    "parameters": [
        {
            "name": "labels",
            "type": "eta.core.types.Array",
            "description": "The classes of objects to detect",
            "required": true
        },
        {
            "name": "weights",
            "type": "eta.core.types.Weights",
            "description": "The weights for the network",
            "required": true,
            "default": "/path/to/weights.npz"
        }
    ]
}
```

An ETA Module Visualized

- Block diagram representation of a simple object detector:



Developing Modules in ETA

- Powerful yet simple **Python framework** for building modules
- Built on popular open-source libraries (OpenCV, TensorFlow)
- Extensive core video library
  - Reading, writing, iterating over video frames
  - Supports all common video codecs
- Robust interface for serializing data to disk

Writing Modules in ETA

ETA Module Definition

```
import eta.core.module as etam

class ExampleModuleConfig(etam.BaseModuleConfig):
    '''An example config class.'''

    def __init__(self, d):
        super(ExampleModuleConfig, self).__init__(d)
        self.data = self.parse_object_array(d, "data", DataConfig)
        self.parameters = self.parse_object(d, "parameters", ParametersConfig)


class DataConfig(Config):
    '''Data configuration settings.'''

    def __init__(self, d):
        self.input_path = self.parse_string(d, "input_path")
        self.output_path = self.parse_string(d, "output_path")


class ParametersConfig(Config):
    '''Parameter configuration settings.'''

    def __init__(self, d):
        self.parameter = self.parse_number(d, "parameter", default=1.0)
```

Example JSON Config

```
{
    "data": [
        {
            "input_path": "/path/to/input.mp4",
            "output_path": "/path/to/output.mp4"
        }
    ],
    "parameters": {}
}
```

Core Video Analytics: Now

Current Core Video Analytics

- Deep (VGG-16) and standard (SURF, ORB, weighted histogram, etc.) embeddings
- Deep (C3D) and standard (per-frame-voting) video embedding
- Object detection
- Object tracking
- Per-frame and clip-level event detection
- Visual image similarity search

Current Road-Related Analytics

- Vehicle and road sign detection
- Fine-grained vehicle recognition (type, make, model, color)

Current Surveillance-Related Analytics

- Action recognition
- Target reacquisition

Core Video Analytics: Soon!

Upcoming Road-Related Analytics

- *Improved* fine-grained vehicle recognition
  - Using object tracking to increase accuracy
  - Integrating license plate recognition (LPR)
- Road scene understanding (lanes, markings, intersections)

Upcoming Surveillance-Related Analytics

- Action recognition models trained on a database of criminal activity video
- Assessing building exterior quality from aerial/street images
- Target search via linguistic description

Related Efforts

- Support for streaming video

We are willing and able to train custom models on your data!

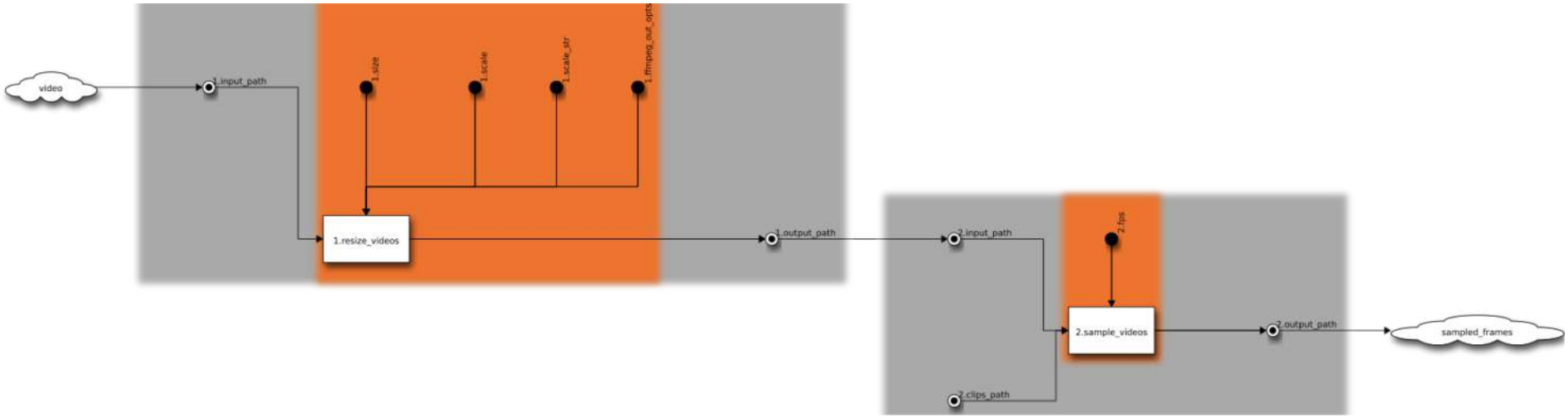
Pipeline Metadata Files

- Pipelines are defined by simple **metadata JSON** files
- Describes the **module connections** that define the execution graph
- Sets parameters for the constituent modules
- Exposes tunable parameters to the user

```
{
    "info": {
        "name": "video_formatter",
        "type": "eta.core.types.Pipeline",
        "version": "0.1.0",
        "description": "A pipeline for formatting video files"
    },
    "inputs": ["video"],
    "outputs": ["sampled_frames"],
    "modules": [
        {
            "name": "resize_videos",
            "tunable_parameters": ["size", "scale"],
            "set_parameters": {}
        },
        {
            "name": "sample_videos",
            "tunable_parameters": ["fps"],
            "set_parameters": {}
        }
    ],
    "connections": [
        {
            "source": "INPUT.video",
            "sink": "resize_videos.input_path"
        },
        {
            "source": "resize_videos.output_path",
            "sink": "sample_videos.input_path"
        },
        {
            "source": "sample_videos.output_path",
            "sink": "OUTPUT.sampled_frames"
        }
    ]
}
```

An ETA Pipeline Visualized

- Block diagram representation of a simple pipeline:



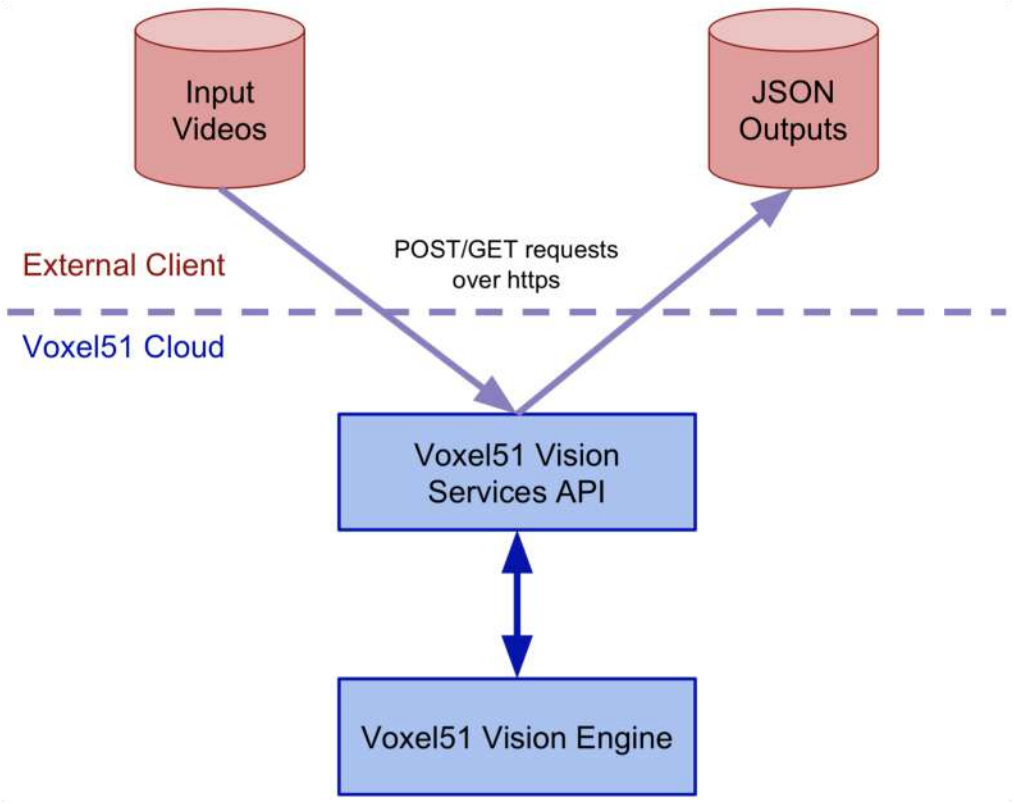
The Voxel51 Vision Services API

Vision Services API

- Canonical **cloud-based interface** through which users interact with ETA
- Accessible via standard HTTPS requests
- Also provide **client libraries** to access the API programmatically from Python and JavaScript
- Lives in **Google Cloud**, with plans to provide a parallel deployment in **Amazon Web Services**
- Full documentation at https://voxel51.com/docs/api

The Vision Services API is currently in private beta. Visit our website at https://voxel51.com to request an invitation to use our platform!

Vision Services API Design



Voxel51 Vision Engine

- Scalable cluster of **Docker containers**, each of which contains a copy of the ETA Library
- Deployed via Kubernetes, a open-source platform for managing containerized workloads
- Cluster **scales automatically** to allocate resources on-demand in response to user job requests
- Runs jobs **asynchronously** as resources become available

Typical User Workflow

- **Register** for our API and obtain an authentication token (one-time only)
- **Upload data** to the cloud via the API
- Schedule analytics to be run on your data by issuing **job requests** via the API

(Tasks run asynchronously in the Voxel51 Cloud)

- Periodically ping the API to **check the status** of your jobs
- When tasks finish, issue an API request to **download the outputs**, which are concisely represented in a flexible JSON format

Example Applications

Target Reacquisition

- User provides a video corpus (e.g. surveillance camera network) and **example image(s)** of an entity of interest
- System queries the corpus and returns best-matches for the entity across the corpus

Linguistic Description Retrieval

- User provides a video corpus and a **natural language description** of an entity of interest
  - *Ex: "red pickup with a dented front bumper"*
- System queries the corpus and returns best-matches

Target reacquisition is currently available on our beta platform!

Target Reacquisition

- Toy example of a target reacquisition query:

Visual Query

Video Corpus

Linguistic Description Query

- Toy example of a linguistic description query:

Linguistic Query

“blue sedan with tape holding on the front-bumper”

Video Corpus

API Usage: Data Upload

- Uploading data in Python:

```
from voxel51.api import API

api = API()

# Upload surveillance corpus
for video in surveillance_corpus:
    response = api.upload_data(video, dataset="surveillance-20180320")
dataset_id = response["dataset_id"]

# Upload query image
response = api.upload_data("suspect-vehicle.png")
data_id = response["data_id"]
```

API Usage: Job Request

- Example request for a target reacquisition query:
- Query image and video corpus were previously uploaded to the Voxel51 Cloud
- Data referenced by IDs in the job request

```
{
    "name": "target-reacquisition",
    "algorithm": "similarity-search",
    "data": {
        "query-image": {
            "data_id": "<data-id>",
        },
        "video-corpus": {
            "dataset_id": "<dataset-id>"
        }
    },
    "parameters": {
        "top_k": 10
    }
}
```

API Usage: Job Request

- Example request for a linguistic retrieval query:

```
{
    "name": "semantic-language-query",
    "algorithm": "natural-language-search",
    "data": {
        "query-string": "blue sedan with tape holding on the front-bumper",
        "video-corpus": {
            "signed_url": "https://storage.googleapis.com/XXXXXX"
        }
    },
    "parameters": {
        "top_k": 10
    }
}
```

Note that data can be accessed from an external cloud via signed URLs

API Usage: Running a Job

- Running a job and checking its status:

```
# Upload job request
response = api.create_job("job.json")
job_id = response["job_id"]
```

```
# Start job
api.start_job(job_id)
```

```
# Check job status
response = api.get_job_status(job_id)
```

```
HTTP/1.1 201 OK
{
    "job_id": "<job-id>",
    "status": "COMPLETE",
    "start_time": "2018-03-21 08:30:00",
    "completion_time": "2018-03-21 08:32:30",
    "message": "Job complete"
}
```

API Usage: Job Output

- Example output for a target reacquisition query:

```
# Download job output
api.download_job_output(job_id, "output.json")
```


```
{
    "top_matches": [
        {
            "confidence": 0.850,
            "video": {
                "data_id": "<data-id>",
            },
            "frame": 150,
            "bounding_box": {
                "top_left": {"x": 0.125, "y": 0.100},
                "bottom_right": {"x": 0.250, "y": 0.300}
            }
        },
        ...
    ]
}
```