



**VOXEL51**

# **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.**

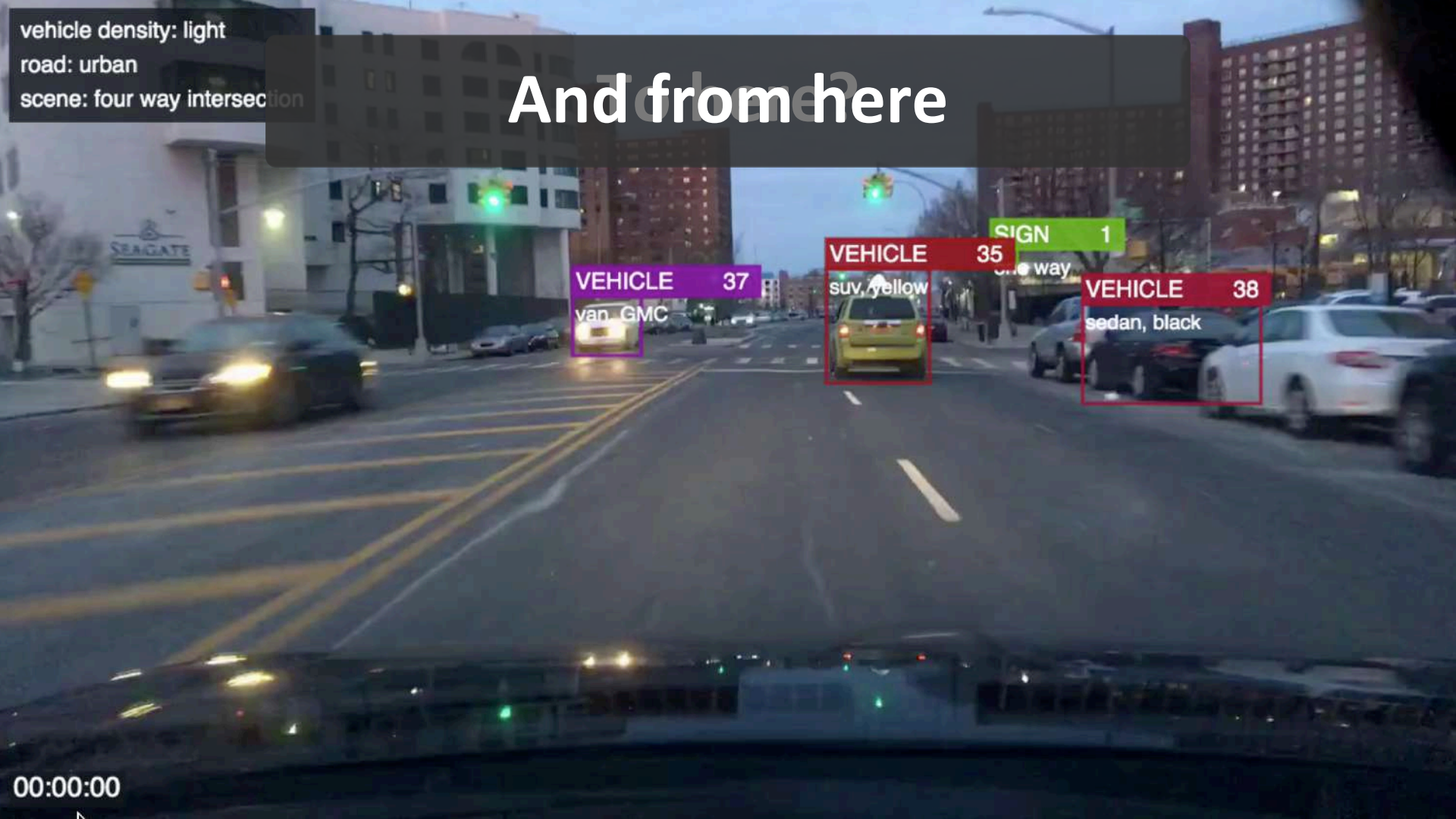
**Posted with permission**

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

VEHICLE 38  
sedan, black

SIGN 1

one way

00:00:00



The background of the entire image is a dense, colorful mosaic of thousands of small, square images. These images appear to be frames from various video recordings, showing a wide variety of scenes: people in different settings, animals, vehicles, and various outdoor and indoor environments. The overall effect is a complex, textured pattern of visual information.

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**?



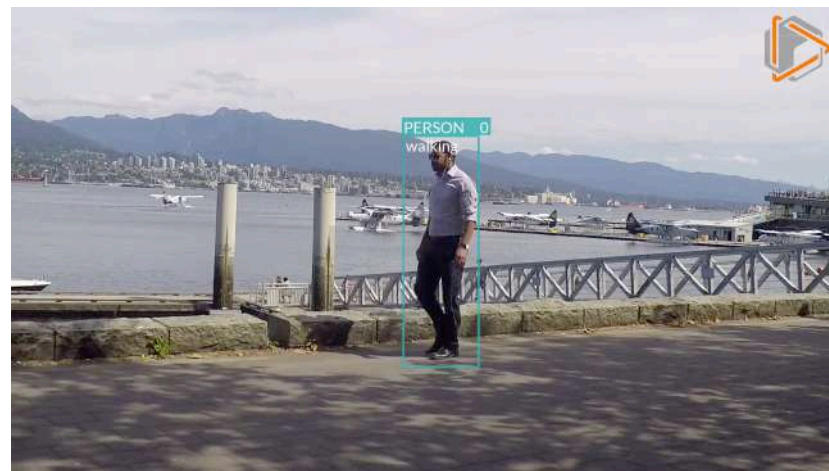
# Video-First Analytics

Standard Approaches are Frame-Based



Running?    Walking?    Standing?

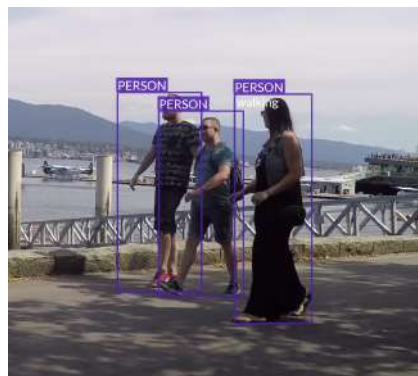
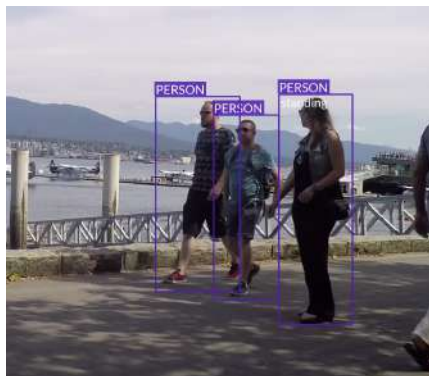
Voxel51 is **Video First**



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**

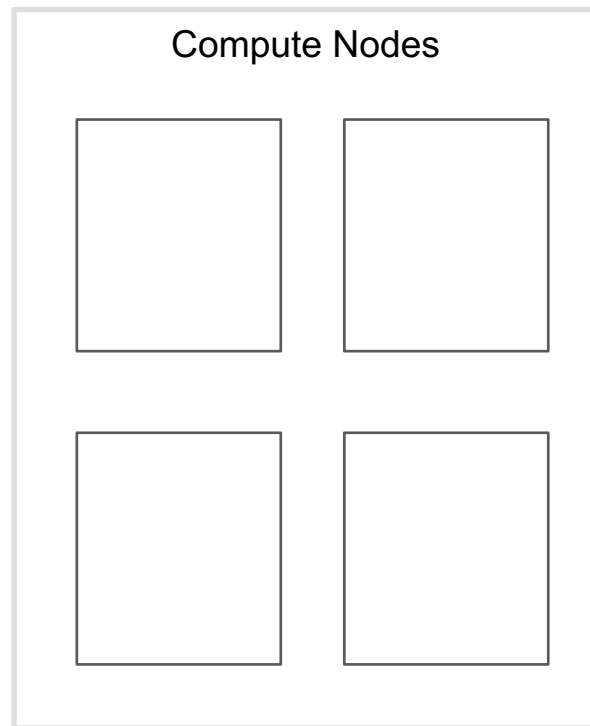


**12.8% improvement**

# Streaming Analytics



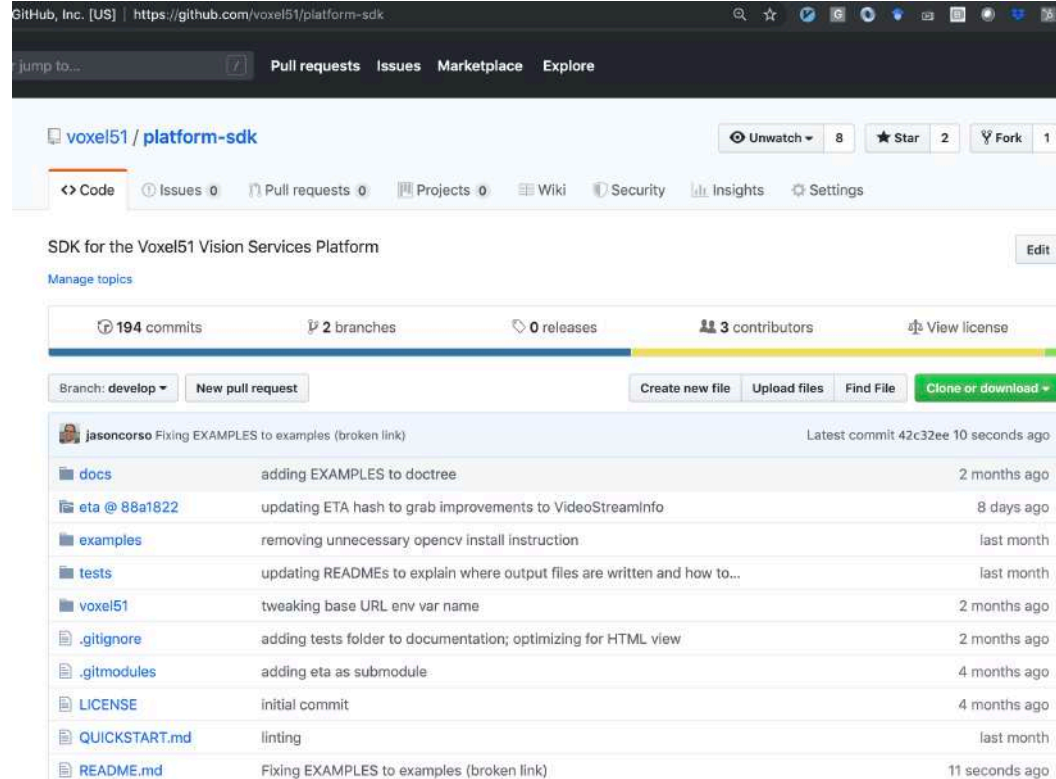
Time 





# 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



GitHub, Inc. [US] | <https://github.com/voxel51/platform-sdk>

voxel51 / platform-sdk

Unwatch 8 Star 2 Fork 1

Code Issues 0 Pull requests 0 Projects 0 Wiki Security Insights Settings

SDK for the Voxel51 Vision Services Platform

Manage topics

194 commits 2 branches 0 releases 3 contributors View license

Branch: develop New pull request Create new file Upload files Find File Clone or download

File	Commit Message	Time
docs	adding EXAMPLES to doctree	2 months ago
eta @ 88a1822	updating ETA hash to grab improvements to VideoStreamInfo	8 days ago
examples	removing unnecessary opencv install instruction	last month
tests	updating READMEs to explain where output files are written and how to...	last month
voxel51	tweaking base URL env var name	2 months ago
.gitignore	adding tests folder to documentation; optimizing for HTML view	2 months ago
.gitmodules	adding eta as submodule	4 months ago
LICENSE	initial commit	4 months ago
QUICKSTART.md	linting	last month
README.md	Fixing EXAMPLES to examples (broken link)	11 seconds ago

# Example Third-Party: Privacy-Preserving Analytic

- Handle FOIA requests with automated ease.



Original

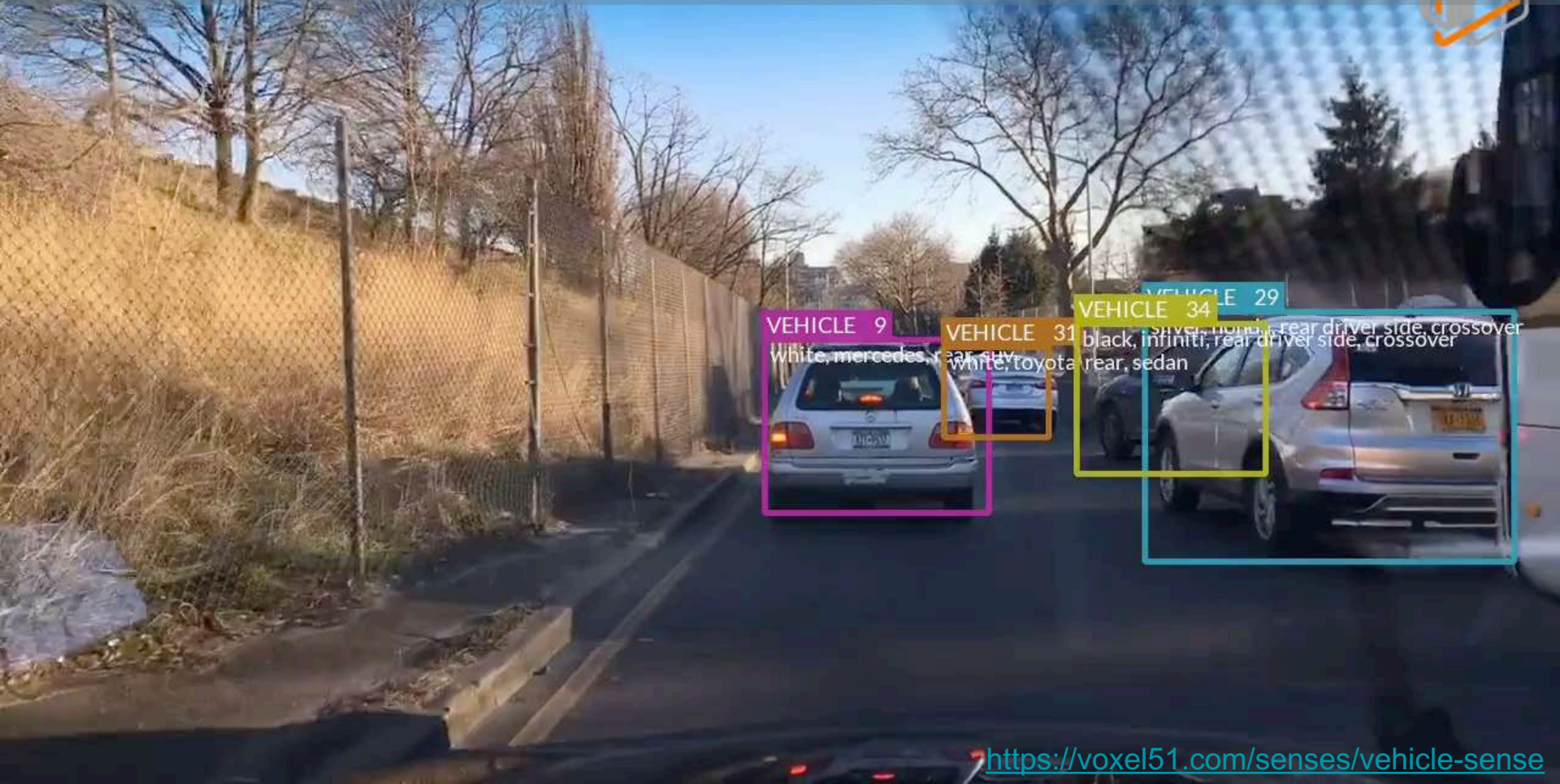


Anonymized Faces





# VehicleSense: fine-grained vehicle recognition





# RoadSense: adds road signs and road scenes

road: urban  
scene: complex intersection  
vehicle density: heavy







Ground Truth: Explosion  
Prediction: Explosion



Ground Truth: theft  
Prediction: theft

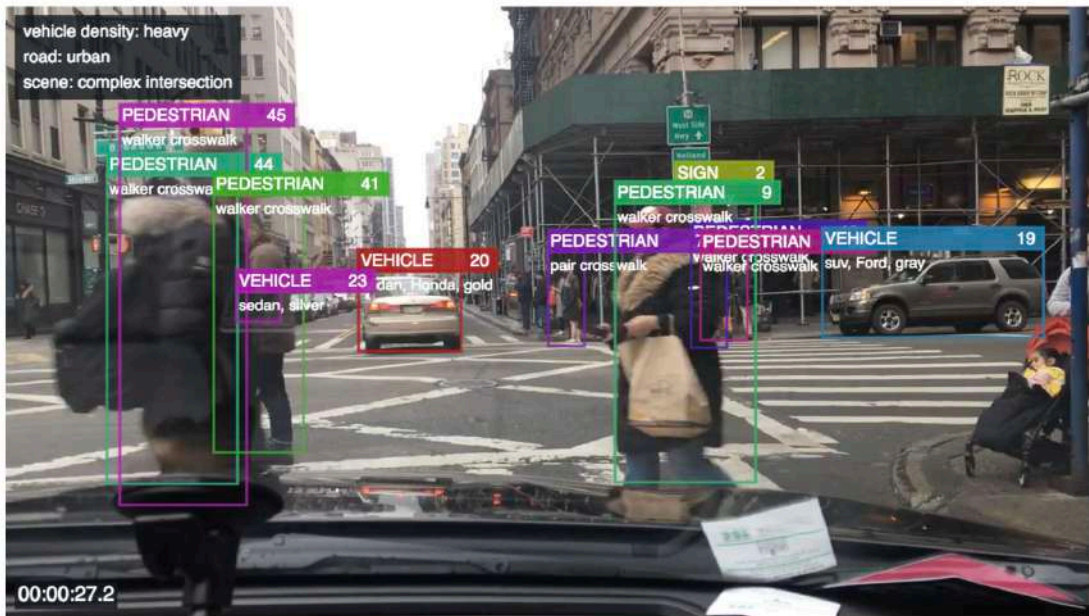




Starts at 00:00:22.8

Duration: 4.1s

Reset Clip



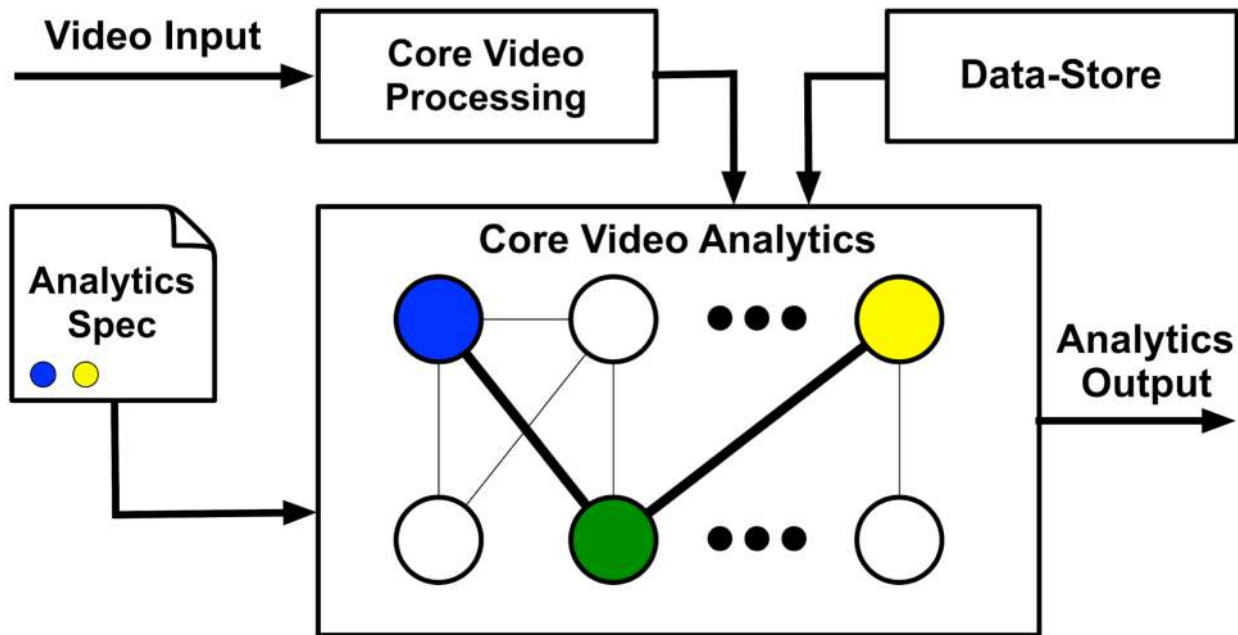
Added by: support@voxel51.com

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

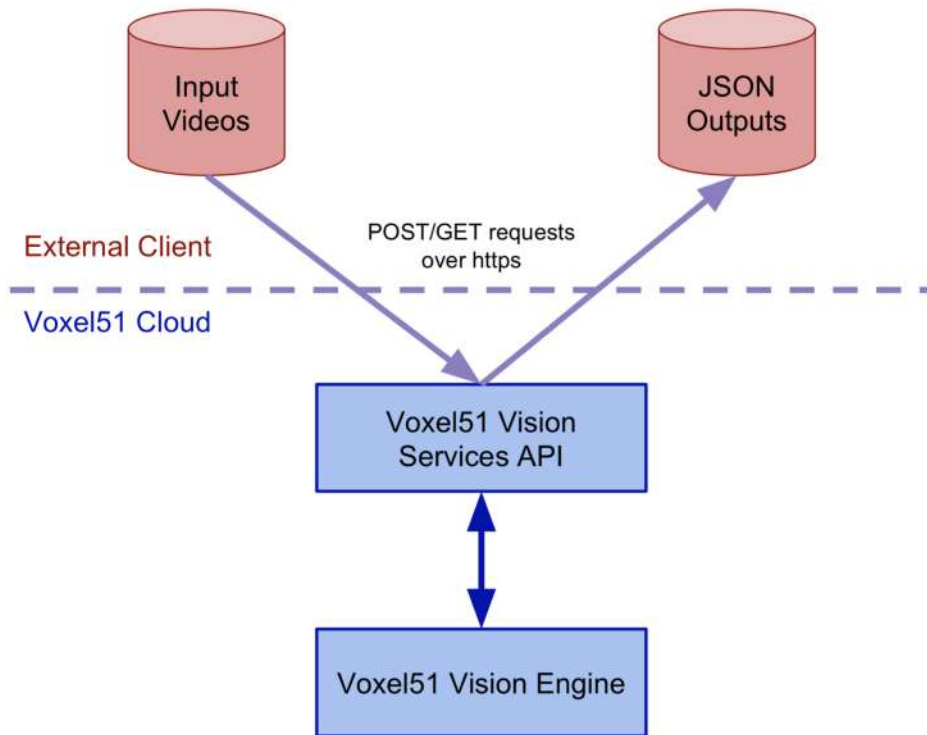
## Core Video Processing

- 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

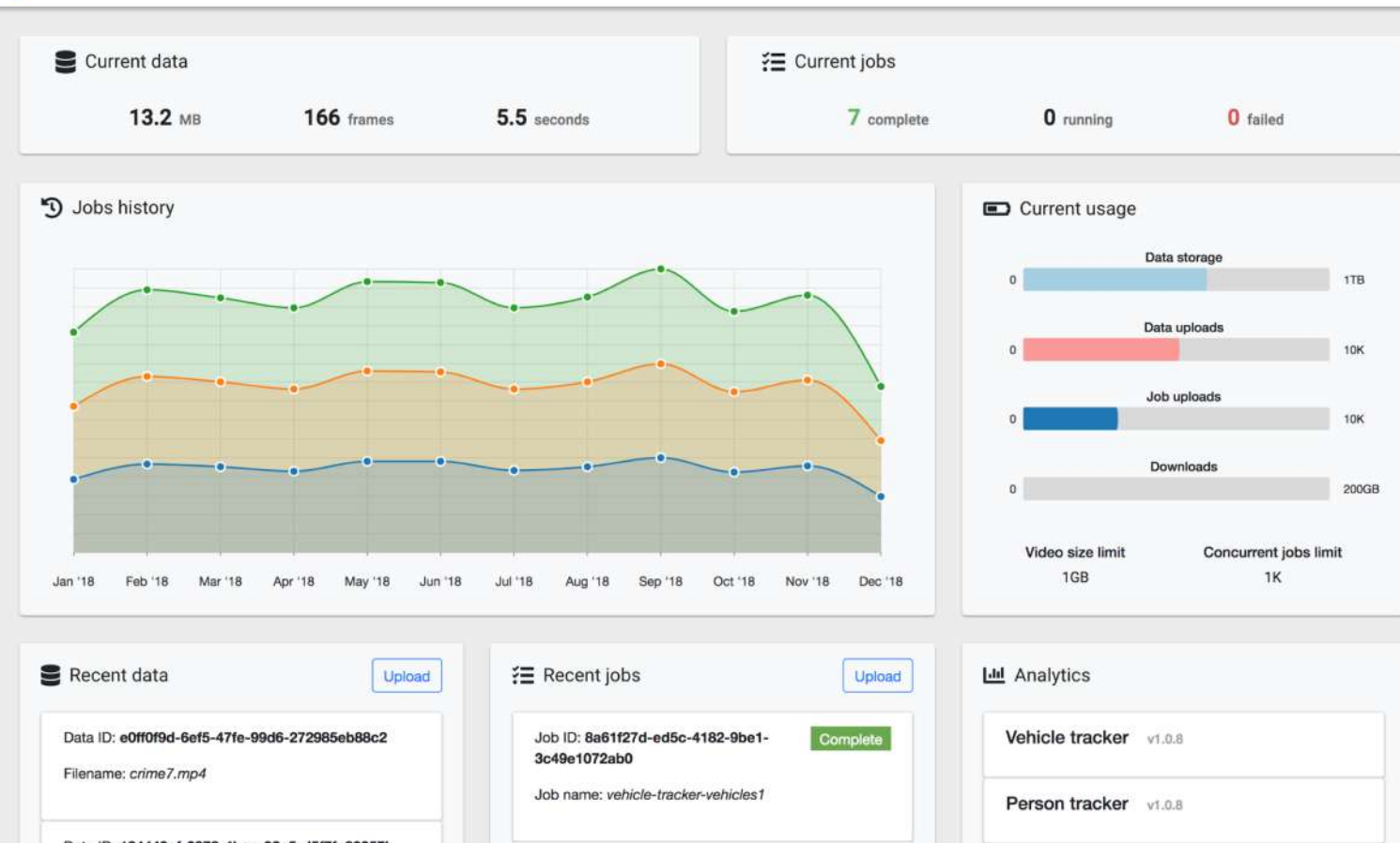
- Implements **core computer vision analytics** such as detection, tracking, classification, and recognition
- Relies heavily on open-source libraries (OpenCV, TensorFlow, Keras, etc.) and publicly available research implementations of modern vision algorithms

# API Design





# Platform Console



# Platform Console

## Jobs Board

### Jobs

Current

Archived

Upload

Start

Archive

Delete

Update TTL

col\_name:keyword, col\_name:keyword...



		OUTPUT		STATISTICS			
<input type="checkbox"/>	Job name	Job ID	GPU	Status	Output	Upload date ↓	Expiration date
<input type="checkbox"/>	sign-tracker-v3-signs	cc3f01d4-783a-486d-a267-cee99d338bb7	✓	<span>Complete</span>		Mar 19, 2019, 3:25 PM	Mar 22, 2019, 3:25 PM
<input type="checkbox"/>	person-tracker-people	89232296-fc51-4f6b-9716-9be0285fefce	✓	<span>Complete</span>		Mar 19, 2019, 2:57 PM	Mar 22, 2019, 2:57 PM
<input type="checkbox"/>	video-formatter-signs	c0fcb65b-9ada-45d4-aad2-773a160043a2	✓	<span>Complete</span>		Mar 19, 2019, 2:56 PM	Mar 22, 2019, 2:56 PM
<input type="checkbox"/>	vehicle-tracker-road	52488667-6561-4a8e-acb9-dd04aba59a4a	✓	<span>Complete</span>		Mar 19, 2019, 2:56 PM	Mar 22, 2019, 2:56 PM
<input type="checkbox"/>	vehicle-tracker-road	0bfc13d3-1d5e-4a06-b3af-7268d2caab8a	✓	<span>Complete</span>		Mar 18, 2019, 1:36 PM	Mar 21, 2019, 1:36 PM

Rows per page: 25 ▾ 1-5 of 5 < >



# Scoop pulls this all together

<https://voxel51.com/demo>

The screenshot displays the Scoop web application interface. At the top, the browser address bar shows the URL <https://demo.voxel51.com>. The application header includes the Scoop logo and the user email `demo@voxel51.com`.

The main dashboard provides a summary of video data:

- 52 Videos**
- 93 Clips**
- 23.6K Frames**
- 13.1 min Total Duration**

The interface is divided into several sections:

- Left Sidebar:** A navigation menu with expandable sections for Vehicles (18), People (0), Activity (0), Infrastructure (4), Road Signs (0), Scene (4), and Vehicle Density (0). The Scene section is expanded, showing various scene types with checkboxes, such as "Complex Intersection" and "Four Way Intersection".
- Statistics:** A section titled "Statistics" showing counts for Vehicles (1998), People (204), and Infrastructure (373). Below this are three horizontal bar charts for MAKE (992), TYPE (689), and COLOR (577).
- Geospatial Information:** A section titled "Geospatial Information" featuring a map view with "Map" and "Satellite" tabs. The map shows a geographic area with several blue and green heatmaps overlaid, indicating specific locations of interest.
- Matching Clips:** A section titled "Matching Clips" at the bottom, displaying a horizontal row of video thumbnails that correspond to the selected filters.

# 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



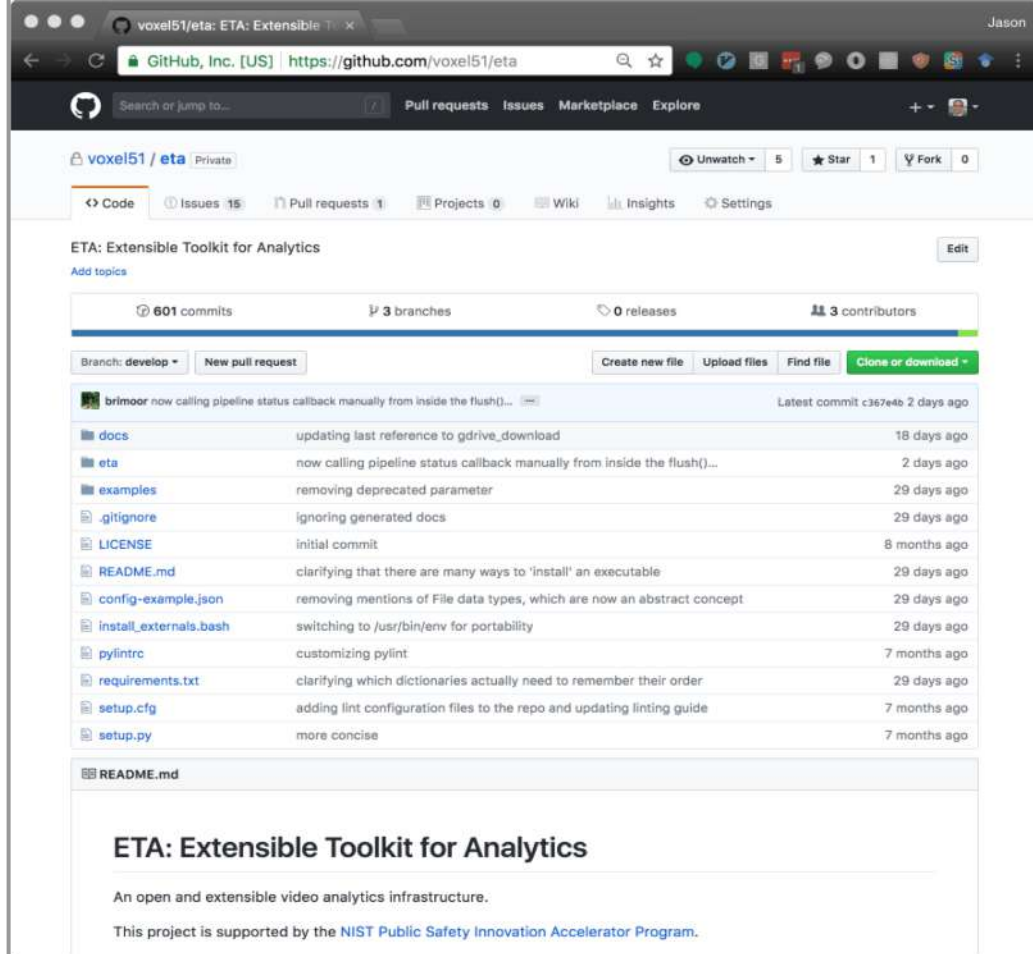
# Insights

- 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>



Screenshot of the GitHub repository page for `voxel51/eta`. The page displays the repository name, navigation tabs (Code, Issues, Pull requests, Projects, Wiki, Insights, Settings), and a list of files and folders. The `README.md` file is expanded, showing the title **ETA: Extensible Toolkit for Analytics** and the description: "An open and extensible video analytics infrastructure. This project is supported by the NIST Public Safety Innovation Accelerator Program."





Search

Analytics Documentation

video-formatter

vehicle-detector

vehicle-tracker

vehicle-indexer

person-detector

person-tracker

person-indexer

**sign-tracker**

Inputs

Parameters

Outputs

object-matcher

## sign-tracker

A tool for tracking road signs in videos.

### Inputs

Name	Type	Description
video	Video	the input video

### Parameters

Name	Type	Description	Required	Default
fps	Number	a frame rate at which to process the video. By default, the frame rate of the input video is used	false	-
size	Array	an optional [width, height] to use when processing the video frames. Dimensions can be -1, in which case the input aspect ratio is preserved. By default, the resolution of the input video is used	false	-

### Outputs

Name	Type	Description	Filename
signs	VideoLabels	a JSON file describing the tracked road signs	signs.json

## object-matcher

A tool for finding best matches for a query image in an object corpus.

### Inputs

Name	Type	Description
query	Image	the input query image to use
videos	ZippedVideoFileDirectory	a zipped directory containing the video corpus
objects	ZippedDetectedObjectsSequenceDirectory	a zip file containing directories of JSON files describing detected objects in each video

shell python javascript

### Example Code

```
from voxel51.api import API
from voxel51.jobs import JobRequest

input_data = "/path/to/input/video.mp4"
job_output = "/path/for/signs.json"

api = API()

# Upload data
data = api.upload_data(input_data)
data_id = data["data_id"]

# Upload and start job
job_request = JobRequest("voxel51/sign-tracker")
job_request.set_input("video", data_id=data_id)
job_request.set_parameter("fps", 15) # optional
job = api.upload_job_request(
    job_request, "sign-tracker-test", auto_start=True, use_gp
job_id = job["job_id"]

# Wait until job completes, then download output
api.wait_until_job_completes(job_id)
api.download_job_output(job_id, job_output)
```

### Example Code

```
from voxel51.api import API
from voxel51.jobs import JobRequest

api = API()

# Upload data
data1 = api.upload_data("/path/to/input/query.png")
data2 = api.upload_data("/path/to/input/videos.zip")
data3 = api.upload_data("/path/to/input/objects.zip")
data4 = api.upload_data("/path/to/input/features.zip")
```



**VOXEL51**

# 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

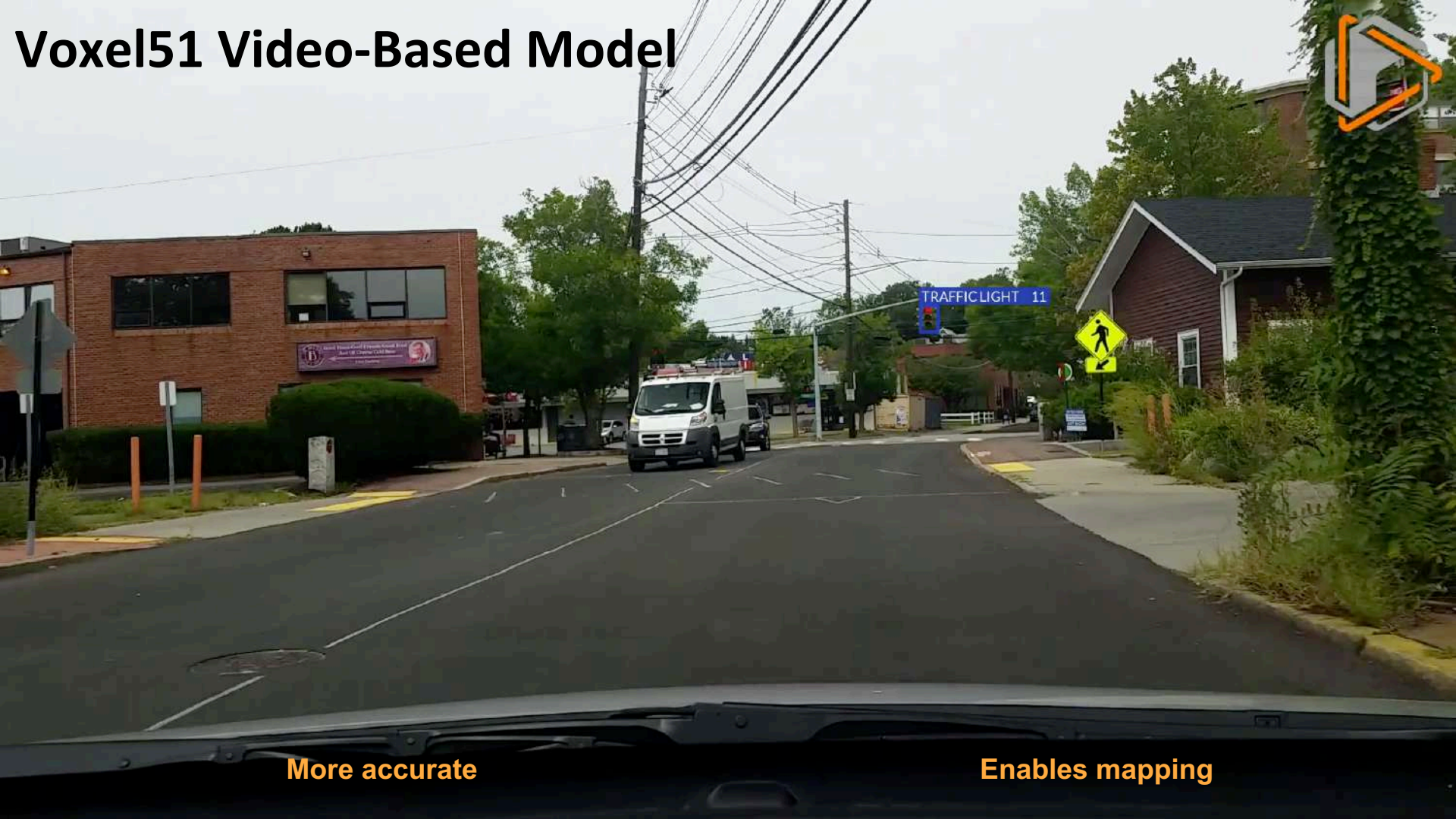


Many spurious detections

Unusable for mapping



# Voxel51 Video-Based Model



More accurate

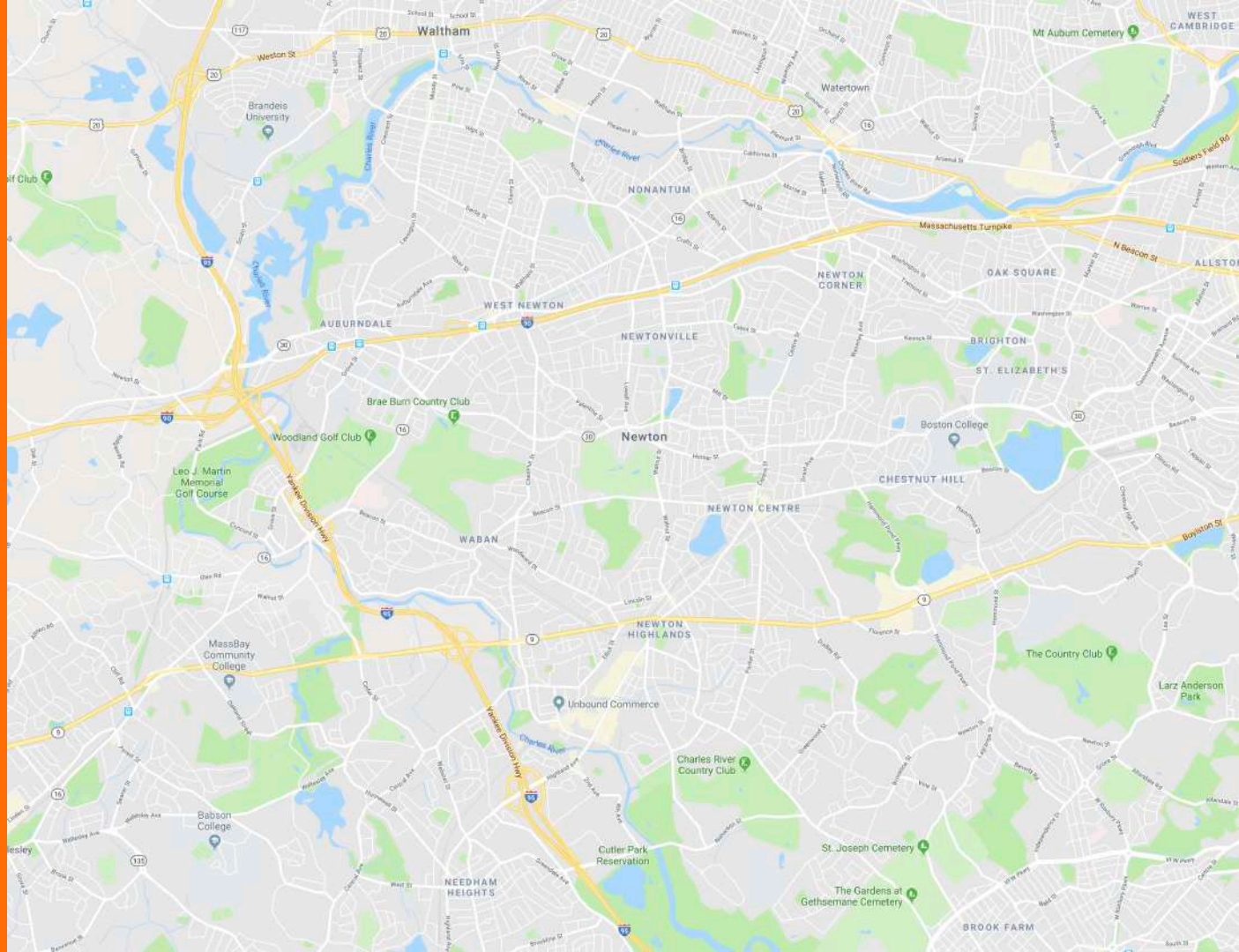
Enables mapping

# Newton, MA

2.6 Days of Video

Few Hundred Miles

10 Hours of Compute

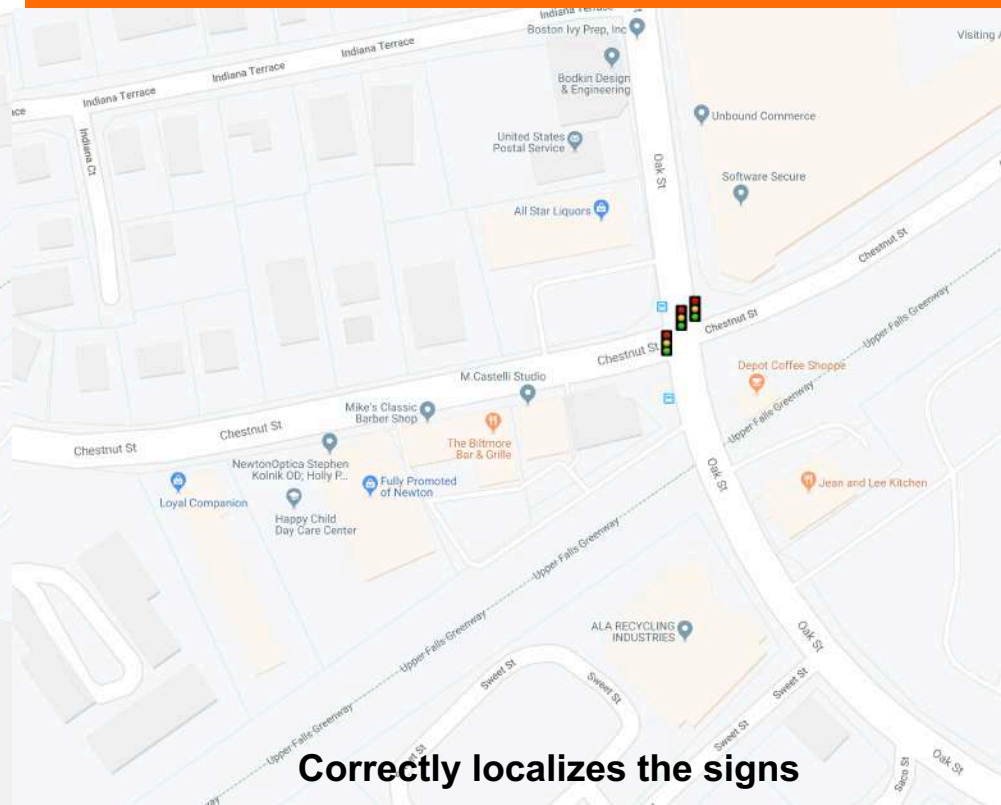




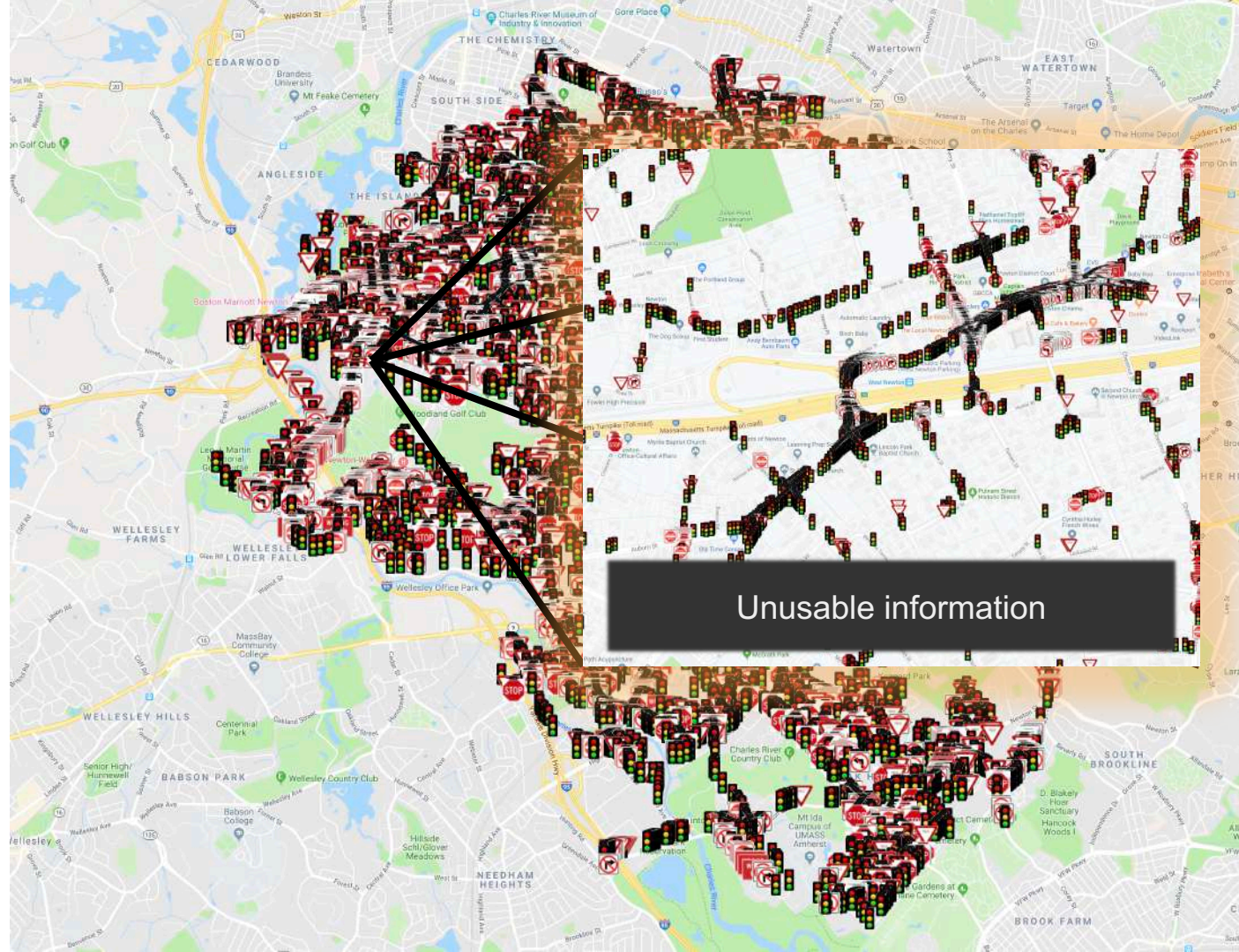
## Frame-Based Models



## Voxel51 Video-Based Model

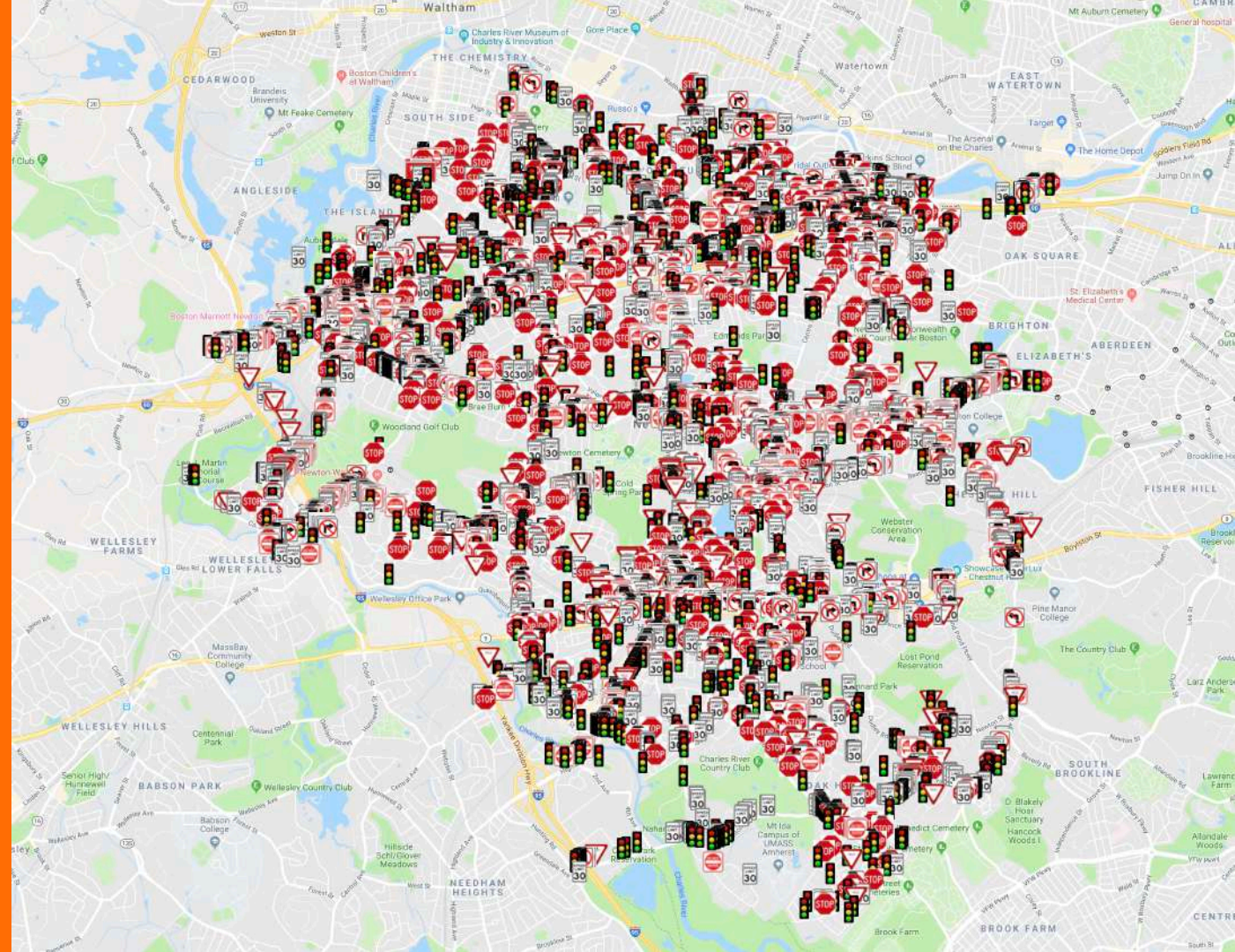


# Frame-Based Road Sign Map





# Voxel51 Road Sign Map







**VOXEL51**

# Case Study Teaser: Baltimore CitiWatch

<https://voxel51.com>

# Fight Detected



Video a78c1b20

Starts at 00:00:13.6

Duration: 9.4s

Reset Clip





[solutions@voxel51.com](mailto:solutions@voxel51.com)  
<https://voxel51.com/demo>

**#PSCR2019**

Come back for the  
**Next  
Session**  
2:25 PM



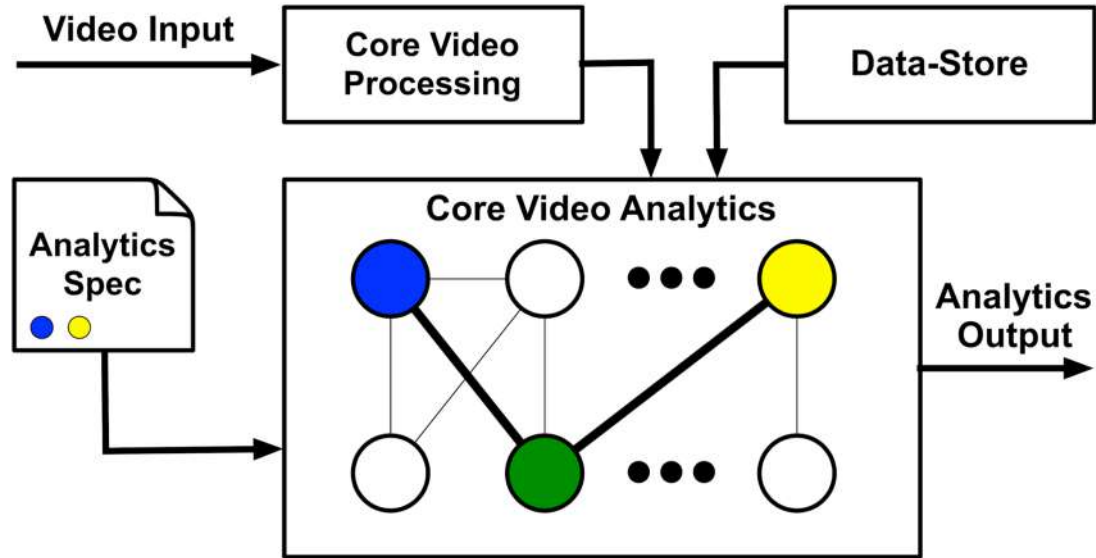


Backup

# Deployment Strategy



# The ETA Library



# ETA Library

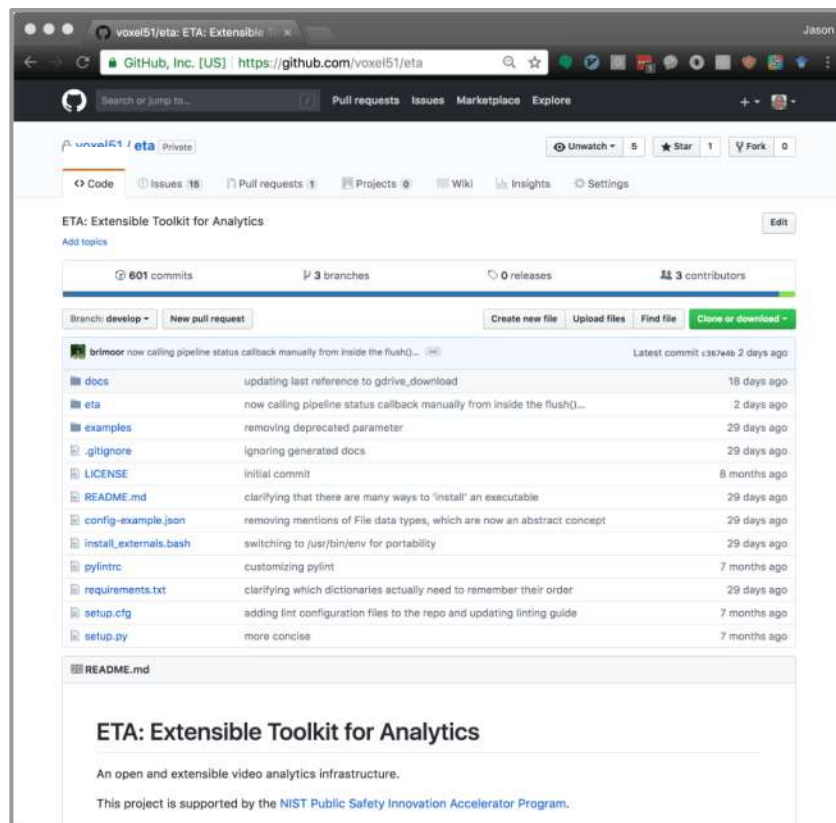
---

- **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!



The screenshot shows a web browser displaying the GitHub repository page for 'voxel51/eta: ETA: Extensible'. The page includes navigation links for Pull requests, Issues, Marketplace, and Explore. The repository is private and has 5 unwatchers, 1 star, and 0 forks. It contains 601 commits, 3 branches, 0 releases, and 3 contributors. The current branch is 'develop'. A table of recent commits is visible, with the latest commit by 'brimoor' 2 days ago. Below the commit list, the README.md file is partially visible, showing the title 'ETA: Extensible Toolkit for Analytics' and a description: 'An open and extensible video analytics infrastructure. This project is supported by the NIST Public Safety Innovation Accelerator Program.'

Commit	Message	Time
brimoor	now calling pipeline status callback manually from inside the flush()	2 days ago
docs	updating last reference to gdrive_download	18 days ago
eta	now calling pipeline status callback manually from inside the flush()	2 days ago
examples	removing deprecated parameter	29 days ago
.gitignore	ignoring generated docs	29 days ago
LICENSE	initial commit	8 months ago
README.md	clarifying that there are many ways to "install" an executable	29 days ago
config-example.json	removing mentions of File data types, which are now an abstract concept	29 days ago
install_externals.bash	switching to ./usr/bin/ for portability	29 days ago
pylintrc	customizing pylint	7 months ago
requirements.txt	clarifying which dictionaries actually need to remember their order	29 days ago
setup.ctf	adding lint configuration files to the repo and updating linting guide	7 months ago
setup.py	more concise	7 months ago

# 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

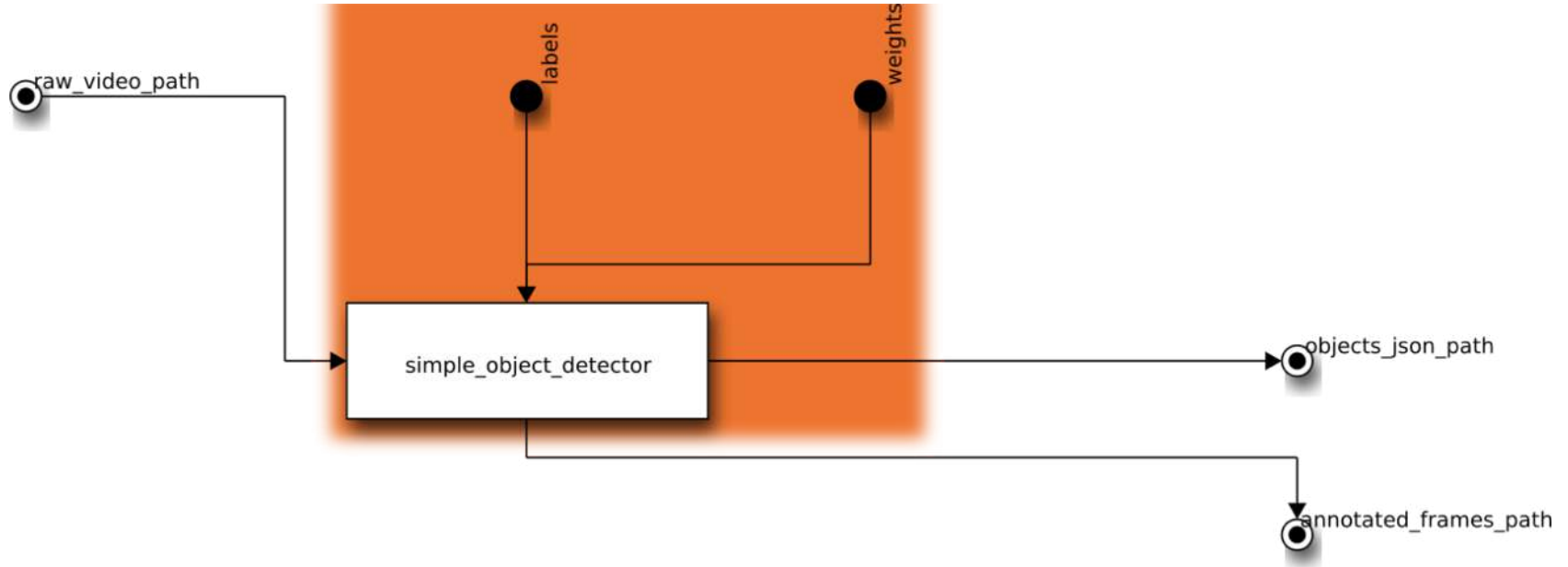
# Module Metadata Files

- 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



Branch: develop eta / eta / core /

brimoor now calling pipeline status callback manually from inside the flush()...

..	
__init__.py	linting __init__
builder.py	renaming log to logfile for consistency
command.py	implementing command-line utility to build and run pipelines
config.py	updating module docstrings
diagram.py	updating to use new node terminology
events.py	linting
features.py	improving docstring of "featurize_if_needed" decorator
geometry.py	technically inputs are coordinates, not RelativePoints. also minimize...
graph.py	adding a simple DirectedGraph implementation with the ability to perf...
image.py	improving exception handling
job.py	removing newlines from logging, which aren't helpful...
log.py	removing newlines from logging, which aren't helpful...
module.py	adding documentation
numutils.py	fixing things for the PR on the CanFeature parts
objects.py	add data classes for counting objects in images
pipeline.py	now calling pipeline status callback manually from inside the flush()...
serial.py	adding a Serializable.to_str() method
status.py	removing pipeline status callback handling from PipelineStatus class
types.py	fixing has_extension typo
utils.py	catching OSError for convenience
vgg16.py	refactored video featurizer code and vgg16 video featurizer code.
video.py	renamed "video.is_video" to "video.is_valid_video"
web.py	cleaning up web module and removing unnecessary CHUNK_SIZE
weights.py	updating to current import naming conventions

# 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!*



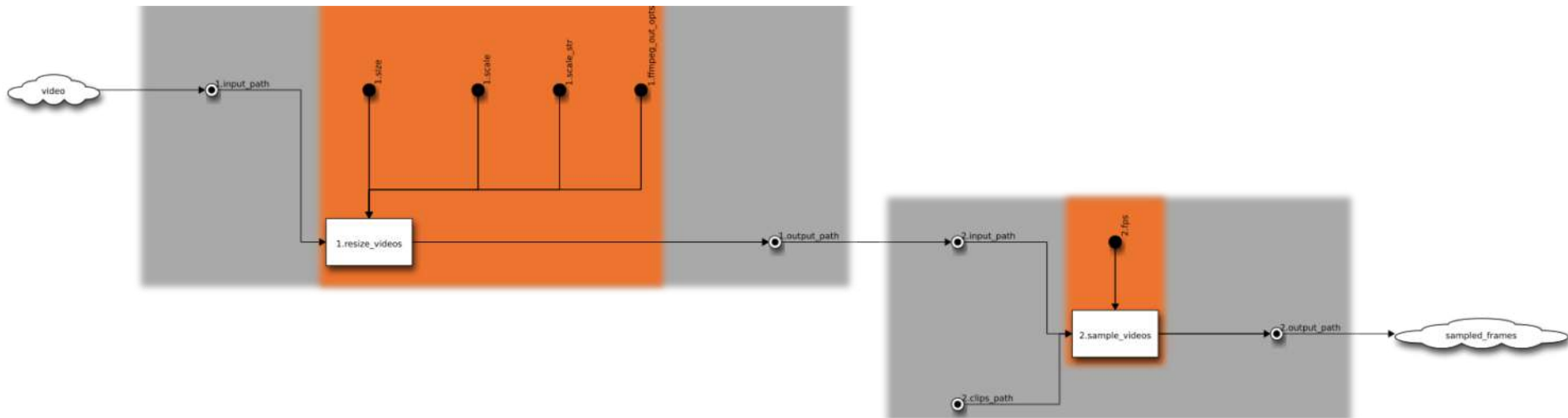
# Pipeline Metadata Files

- 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

Our Vision Services API is the bridge to our state-of-the-art computer vision engine. With JavaScript and Python SDK's available, it's easy integrate our vision engine into your development or production codebases.

Docs: [JS](#) [Python](#) [API](#)

# Vision Services API

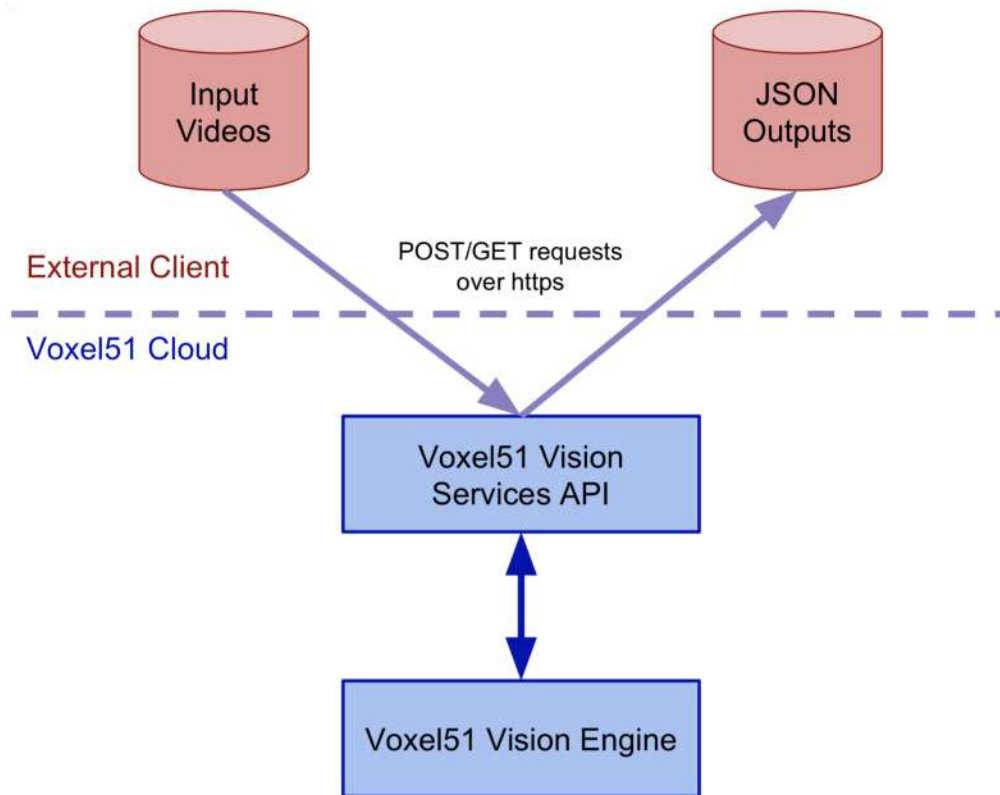
---

- 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-corpora": {
      "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}
      }
    },
    ...
  ]
}
```