Deep Video Analytics

A data-centric approach to Computer Vision

Akshay Bhat
Cornell Tech, Cornell University.

Developments over last 5 years
High quality libraries & pre-trained models

- Theano
- Torch
- ROS
- Caffe
- Tensor Flow
- MXNET
- PyTorch
- Deeplearnjs

- Recognition
  - Inception / VGG / Resnet
- Detection
  - R-CNN / YOLO / SSD
- Face detection / recognition
  - MTCNN / Facenet
- Semantic Segmentation
  - Multipathnet / FCN / CRFasRNN

Developments over last 5 years
A deluge of datasets!

- Open Images
- Yahoo Flickr Creative Com. 100M
- MSCOCO
- ViCom
- Visual Genome
- YouTube-BoundingBoxes / 8M
- AMOS
- imSitu, Charades by AllenAI
- KITTI /Toronto City
- Udacity car dataset
- Caltech, INRIA, ETH Pedestrians
- Stanford Drone Dataset
- Uber text
- THUMOS

Number of datasets ≅ Number of research groups
With each dataset having its own JSON or XML format, incompatible with all others.

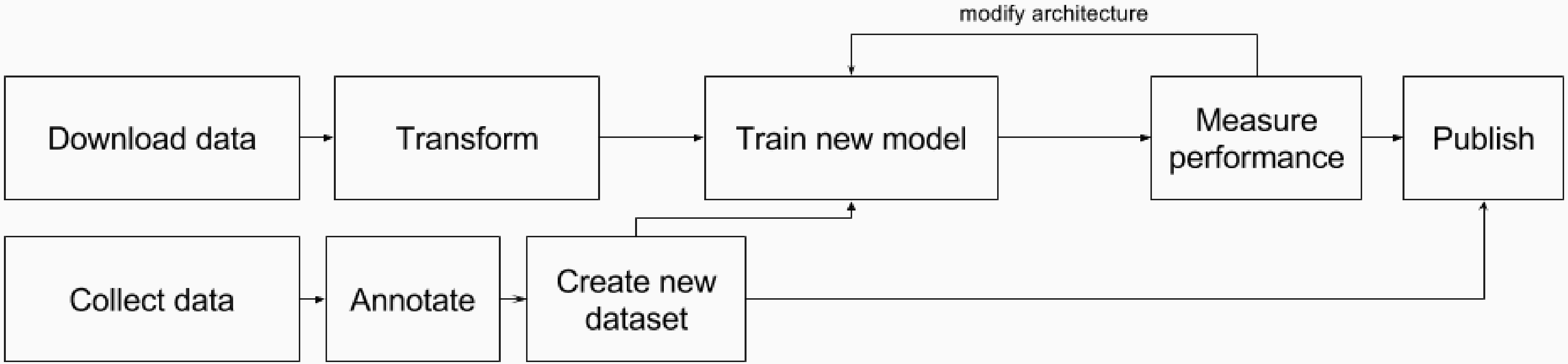
What else changed over last 5 years?

- Container ecosystem (Docker, Kubernetes) enables deployment of complex applications.
- Ability to scale quickly by adding compute capability (including GPUs) billed at minutes / seconds resolution.
- Flexible cloud storage options. (S3, EBS & EFS)

What is hidden in plain sight?

Model-centric approach

Libraries & frameworks are designed with **goal of training and evaluation of models for individual tasks**.



Unsuitable for building systems that learn in interactive manner, or leverage data from multiple sources or combine multiple tasks.

We need a data-centric approach that allows us to combine

- Models for multiple tasks
- Data from multiple sources
- User Interaction / interface

A Relational Model of Data for Large Shared Data Banks. By Edgar F. Codd

Can we develop an equivalent of relational model for visual data?

Relational data : Postgres, MYSQL, SQLite

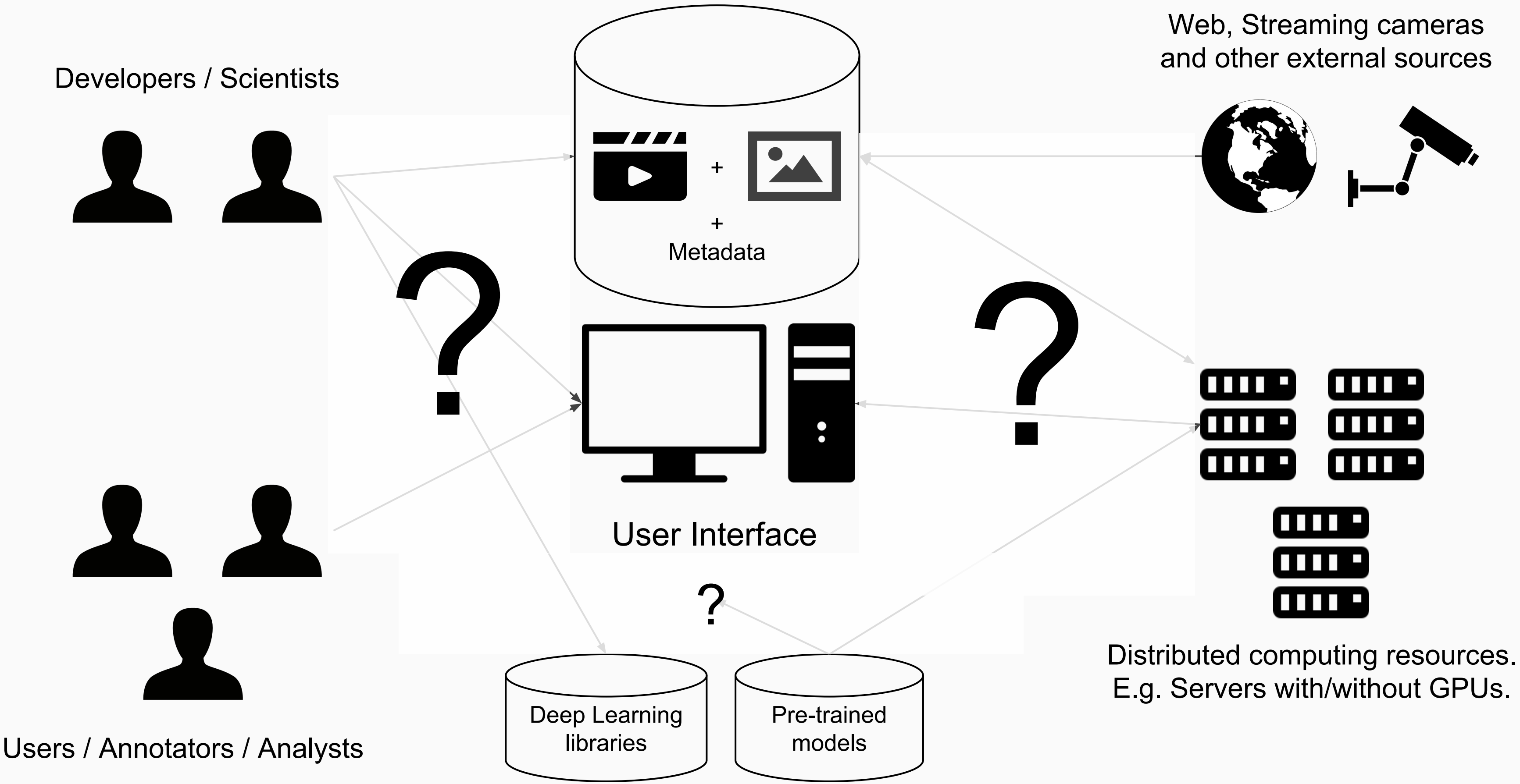
::

Text, HTML : Lucene/Solr, Elasticsearch

::

Videos & Images : _______________________

How do we structure Visual Data processing?



Previous attempts: LIRE project

- LIRE: Lucene Image Retrieval
  - http://www.lire-project.net/
- Developed pre-Deep Learning
- Functionality limited to computing & storing feature vectors such as Color Layout, Edge Histogram, etc.

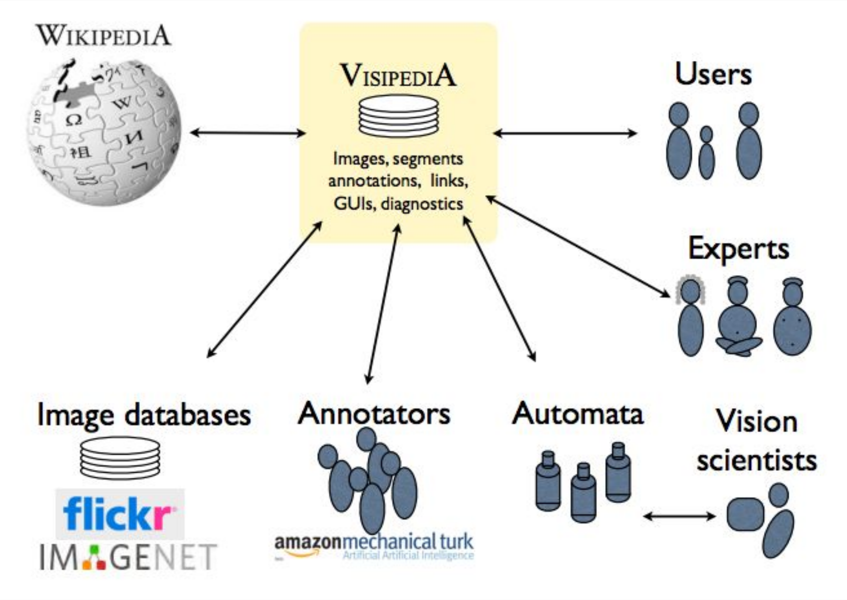
Previous attempts: CloudCV

- Large Scale Distributed Computer Vision as a Cloud Service
- Support for OpenCV, Graphlab, Cafe
- Image Classification, VQA, stitching, etc
- Does not retains state. E.g. you cannot store images.

Previous attempts: NVidia DIGITS

- ”DIGITS (the Deep Learning GPU Training System) is a webapp for training deep learning models. ”
- Load/create datasets, train models, deploy models.
- Aimed at researchers
- Written in Python/Flask with Torch & Caffe supported

Previous attempts: Visipedia



Taken from Vision of a Visipedia, Perona et. al.

Previous attempts: Visipedia

- Collaborative creation of visual data
- Pre-defined set of concepts E.g. Birds, Trees
- Different type of participants
    - Experts, Annotators, Citizen Scientists, Users, Computer scientists
- Retains state

Previous attempts: VMX.ai

- Underfunded Kickstarter project Circa Jan 2014
- by Tomasz Malisiewicz
- Pre Tensor Flow, Pre Deep Learning
- Allow developers to create real time detectors
- Support for training model

Quick summary

- LIRE: limited functionality (Lucene add-on)
- CloudCV: Provides a service, cannot retain “state”
- NVidia Digits: Intended for training not inference
- Visipedia: Intended to be a monolithic deployment

Few ongoing attempts

- Scanner by Alex Poms (CMU) & Will Crichton (Stanford)
  - https://github.com/scanner-research/scanner
- Kitware Image and Video Exploitation and Retrieval
  - https://github.com/Kitware/kwiver
- VISE project by Oxford VGG group
  - https://gitlab.com/vgg/vise

Relational data : Postgres, MYSQL, SQLite

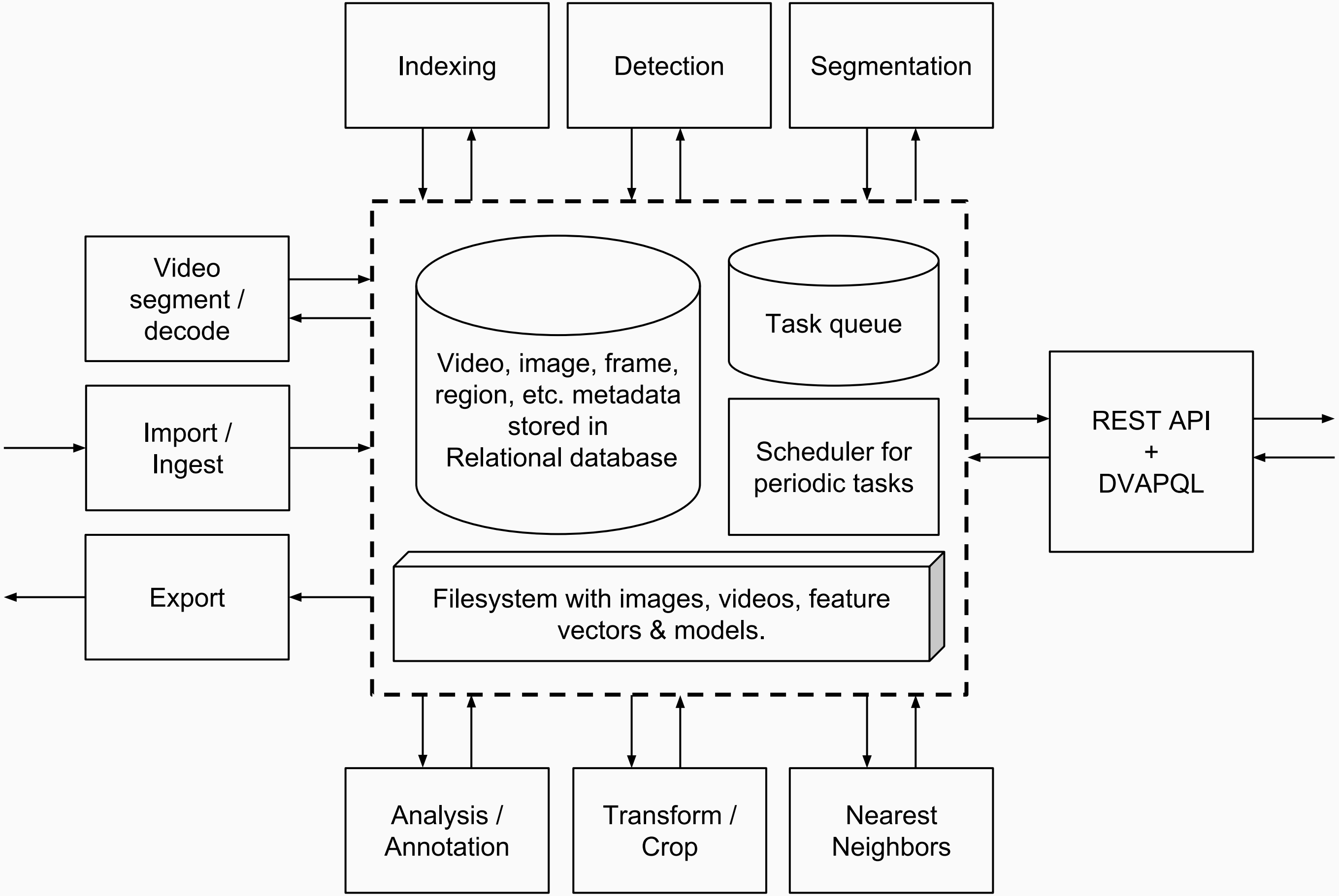
::

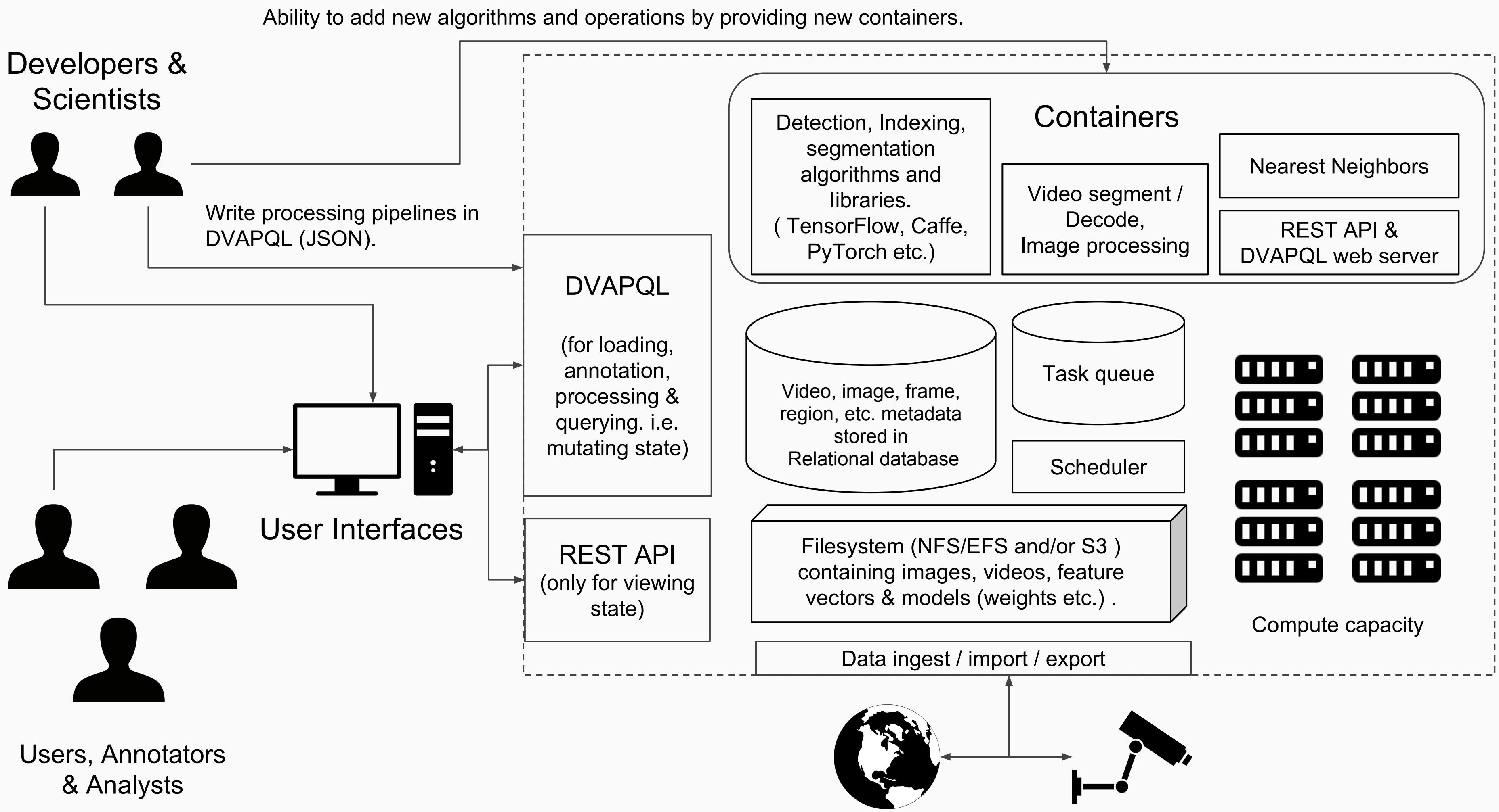
Text, HTML : Lucene/Solr, Elasticsearch

::

Videos & Images : ***Deep Video Analytics***

Model-centric to **Data-centric**

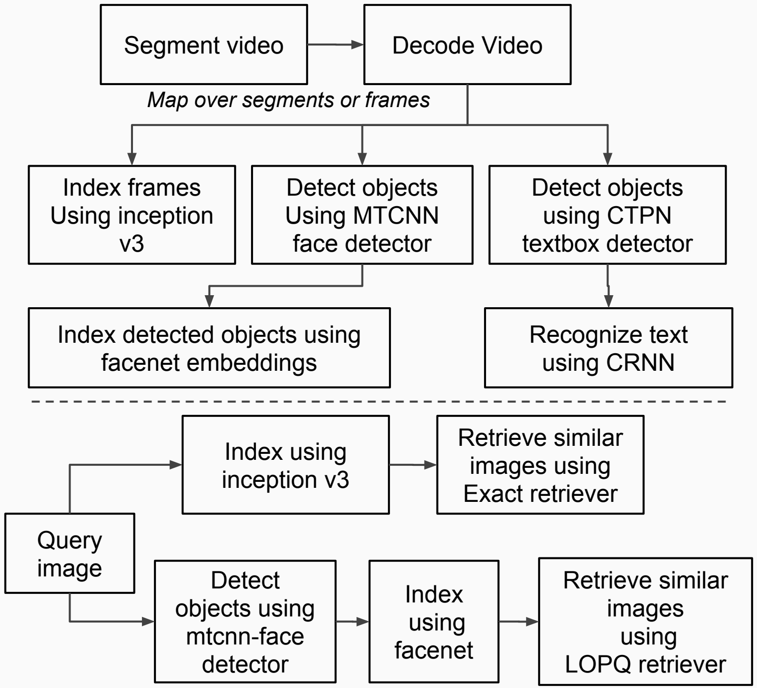




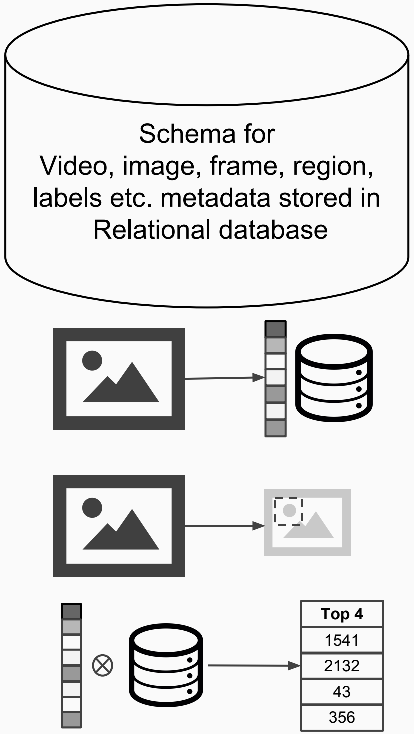
Ability to add new algorithms and operations by providing new containers.
Developers & Scientists
Write processing pipelines in DVAPQL (JSON).
Containers
Detection, Indexing, segmentation algorithms and libraries. (TensorFlow, Caffe, PyTorch etc.)
Video segment / Decode, Image processing
Nearest Neighbors
REST API & DVAPQL web server
DVAPQL
(for loading, annotation, processing & querying. i.e. mutating state)
Video, image, frame, region, etc. metadata stored in Relational database
Task queue
Scheduler
User Interfaces
REST API (only for viewing state)
Filesystem (NFS/EFS and/or S3) containing images, videos, feature vectors & models (weights etc.) .
Compute capacity
Data ingest / import / export
Users, Annotators & Analysts

We provide all three!

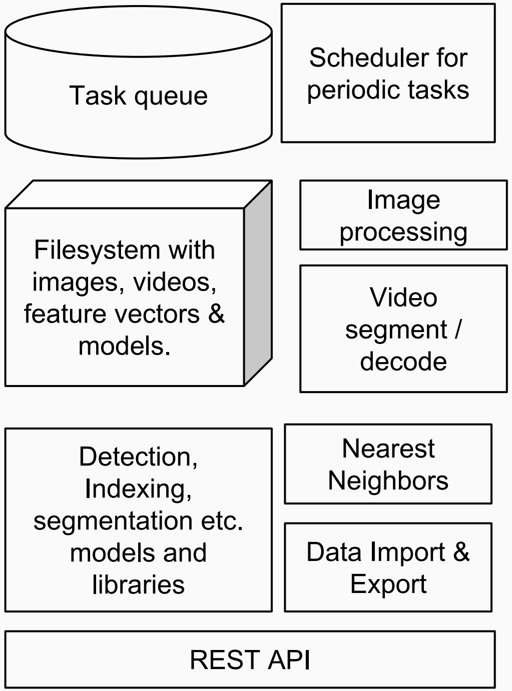
Event based Processing & Query Language

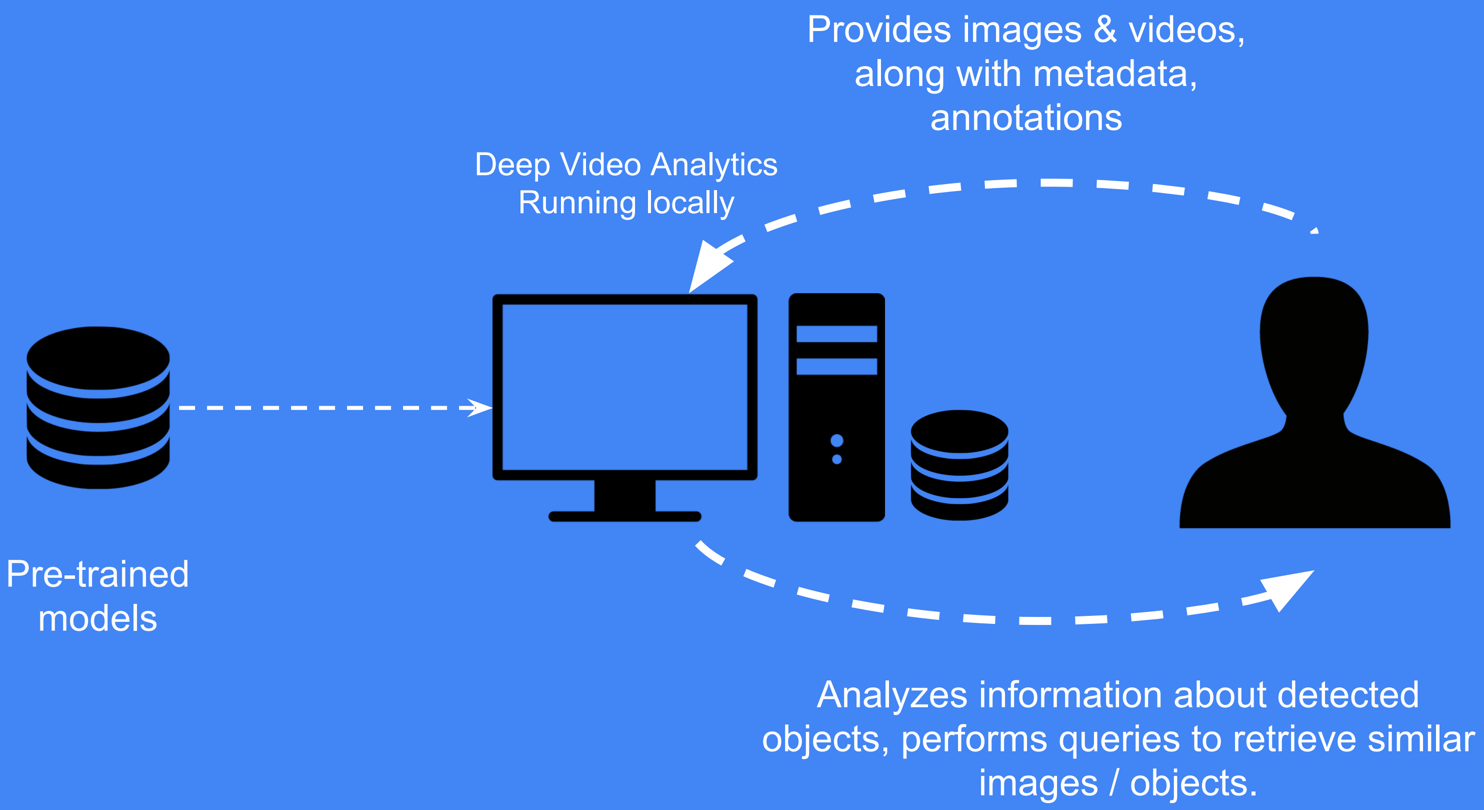


Data & processing model

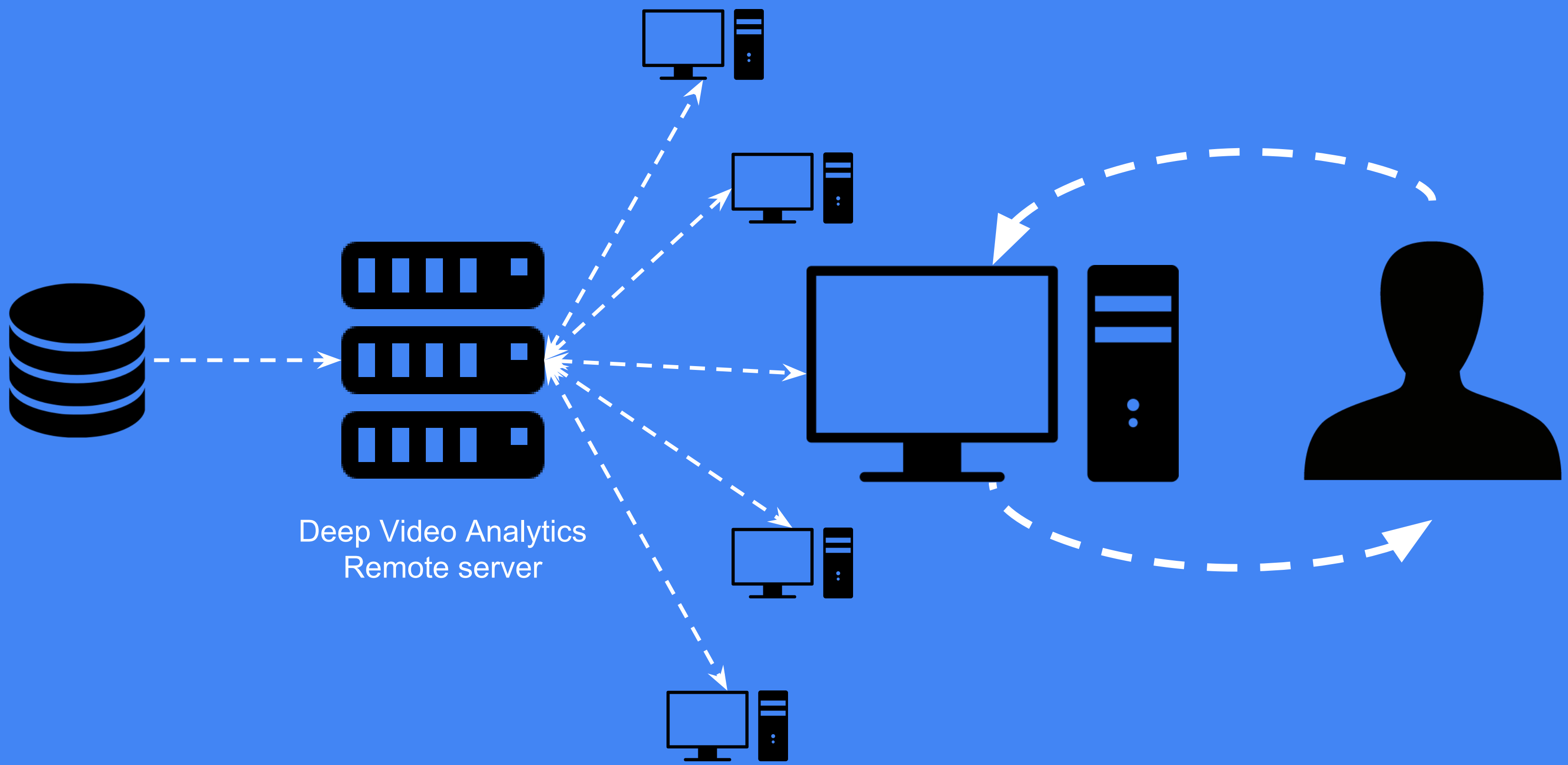


Implementation





Provides images & videos,
along with metadata,
annotations
Deep Video Analytics
Running locally
Pre-trained
models
Analyzes information about detected
objects, performs queries to retrieve similar
images / objects.


Deep Video Analytics
Remote server

Design goals

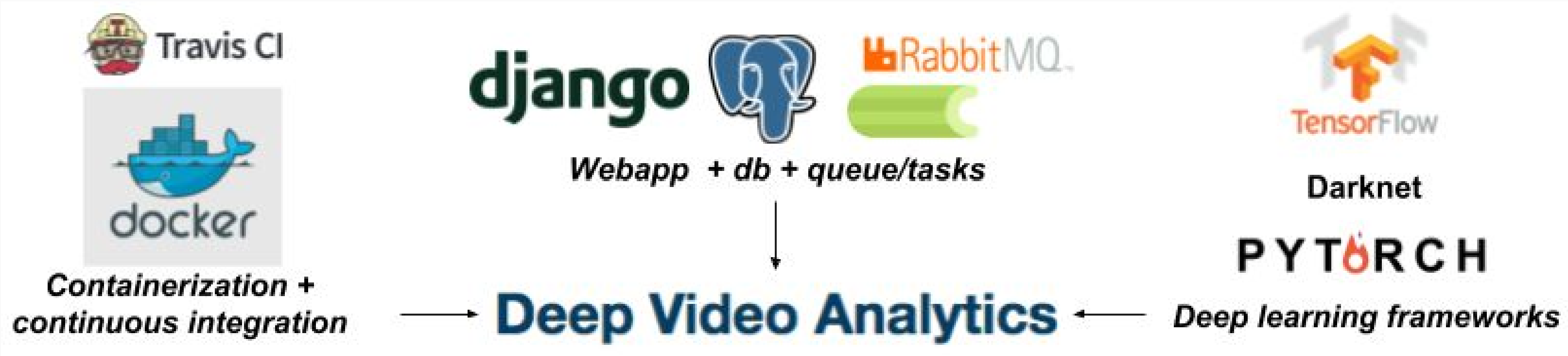
- Usable by non-researchers
- Visual Search as a “Primary User Interface”
- Users can provide data easily (via upload, youtube-dl, annotation UI etc.)
- Batteries-included approach with an indexing and detection pipeline
    - Tensor Flow Inception v3, VGG-16, Single Shot Detector trained on COCO
    - Face detection / alignment / recognition
    - Deep OCR using CRNN & CTPN. Train new detectors using YOLO+Keras.
- Pre-indexed datasets from different domains can be quickly loaded
- Can be easily customized by developers & researchers.

Technical goals

- Useful without having to write code or config
- Works on machines with and without GPUs
  - Works (albeit slowly) without a GPU, tested on Linode VPS with 8Gb RAM & 4 Cores
- Handles uploads and continuous index updates
- Data can be easily imported, exported and shared
- Can be easily modified by technical users
  - E.g. Adding more operations to processing pipeline
- Can be scaled out by adding more GPUs / Machines

Frameworks & libraries used

- Django, Postgres, Celery, RabbitMQ, Docker, NVidia-Docker
- Tensorflow (primary), PyTorch, OpenCV, FFmpeg, LOPQ & Caffe



What are the core primitives for Visual Data Analytics?

Visual Data

=

{ Images, Videos, Annotations, Features}

Data & Processing

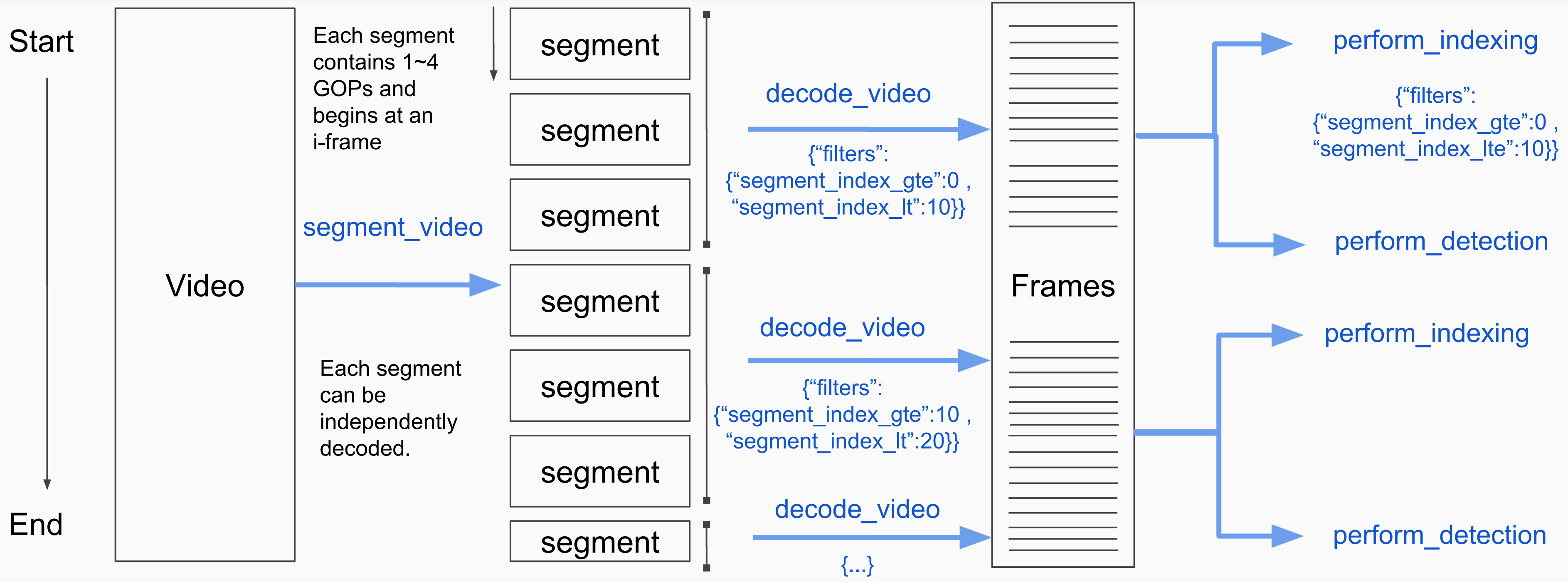
Data

- Video / Segment
- Dataset
- Frame / Image
- Regions over an image
- Tubes over sequence of images
- Feature vectors
- Audio

Processing

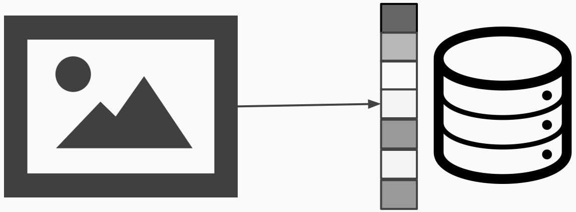
- Video Segmentation + Decode
- Image processing
  - Indexing / Detection / Segmentation / Analysis
- Vector processing
  - Retrieve nearest neighbor / Build K-NN graph
- Image transformation
  - Crop / Resize / Align / Apply segmentation mask

Video processing
Parallelized segment + decode pipeline



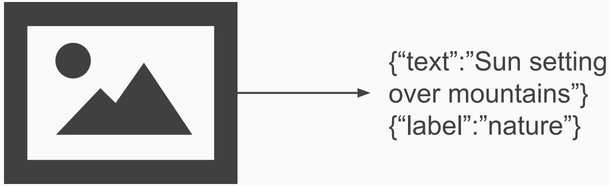
Frame/Region processing operations

Indexing



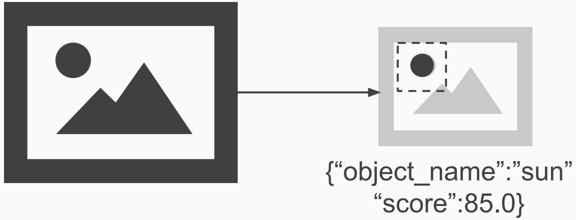
Compute feature vector such as Inception pool, embedding, RGB histogram etc.

Analysis



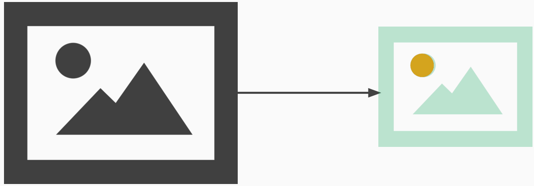
Analyze image/region and generate metadata (E.g. text description) and/or label

Detection



Detect objects and return bounding boxes

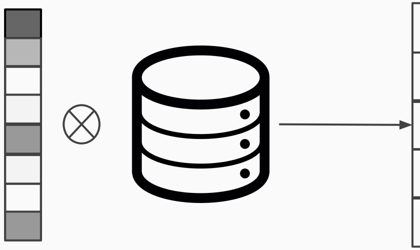
Segmentation



Compute pixel-wise mask using semantic segmentation, superpixels etc.

Vector processing operations

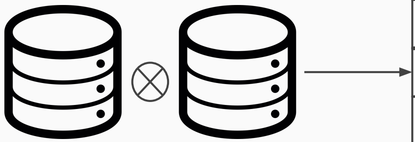
Retrieval



Top 4
1541
2132
43
356

Given feature vector find K-Nearest Neighbors

Matching



Node	1st NN	2nd NN	3rd NN
1	213	114	7761
2	762	132	42
3	2133	673	1233
4	2344	234	9364

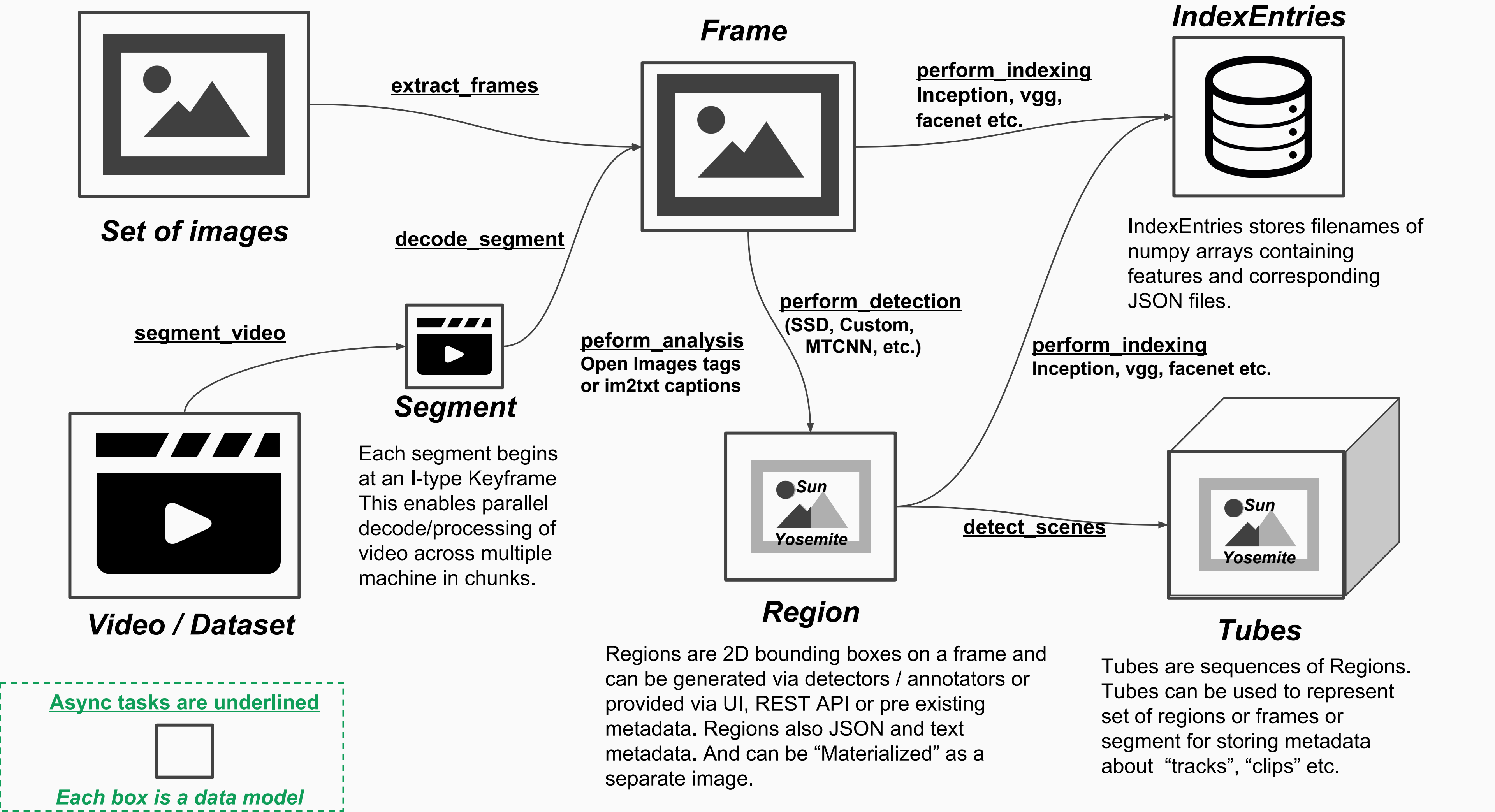
Given a set of vectors generate K-NN graph

Leverage latest open source implementations for approximate & exact Nearest Neighbors

- Yahoo Locally Optimized Product Quantization (Apache)
- Facebook AI Similarity Search (BSD + **PATENTS restrictions**)

Data & Processing
Key insights

- Different operations have different requirements
  - In terms of number of computations and memory
  - Segmentation > Detection > Indexing / Analysis
- Also different I/O access patterns
  - Detection & Analysis does not requires writing to file system only DB and read
  - Indexing requires writing to filesystem to store computed vectors
  - Segmentation requires writing to filesystem to store computed masks as .png files
- By separating operations we can reason about hardware requirements



Set of images
extract_frames
Frame
perform_indexing
Inception, vgg,
facenet etc.
IndexEntries
IndexEntries stores filenames of
numpy arrays containing
features and corresponding
JSON files.
decode_segment
segment_video
Segment
Each segment begins
at an I-type Keyframe
This enables parallel
decode/processing of
video across multiple
machine in chunks.
Video / Dataset
peform_analysis
Open Images tags
or im2txt captions
perform_detection
(SSD, Custom,
MTCNN, etc.)
perform_indexing
Inception, vgg, facenet etc.
Sun
Yosemite
Region
Regions are 2D bounding boxes on a frame and
can be generated via detectors / annotators or
provided via UI, REST API or pre existing
metadata. Regions also JSON and text
metadata. And can be “Materialized” as a
separate image.
detect_scenes
Sun
Yosemite
Tubes
Tubes are sequences of Regions.
Tubes can be used to represent
set of regions or frames or
segment for storing metadata
about “tracks”, “clips” etc.
Async tasks are underlined
Each box is a data model

DVAPQL
Deep Video Analytics Processing & Query Language

- Specified in JSON
- Launch multiple hierarchical tasks
- Three types of processes
  - Query
    - Retrieve similar images, etc.
  - Process
    - Import video, index images, detect, etc.
  - Schedule
    - Monitor video stream, etc.
- REST API for viewing state & submitting DVAPQL

Example

{ “process_type” : “V”, “tasks”: [
{“operation”:”perform_video_segmentation”, ...]}

{ “process_type” : “Q”, “b64_image_data”:”.....”,
“tasks”: [{“operation”:”perform_indexing”, ...
]}

{ “process_type” : “S”, “tasks”: [
{“operation”:”ingest_video”, ...]}

A task based hierarchical processing model

{"operation": "perform_detection", "arguments": { "filters": "__parent__", "next_tasks": [ ] }}

{"operation": "perform_transformation", "arguments": { “op”:”crop” , "filters": {"event_id":"__parent_event__"}, "next_tasks": [ ] }}

{"operation": "perform_indexing", "arguments": { "filters": {"event_id" : "__grant_parent_event__", "w_gte" : 50, "h_gte" : 50 }, "indexer": "vgg" }}

{"operation": "perform_indexing", "arguments": { "filters": {"event_id" : "__grant_parent_event__", "w_gte" : 50, "h_gte" : 50 }, "indexer": "inception" }}

All above tasks run on a specific video / dataset which is not shown for brevity.

Queues for optimal task processing

- Different tasks have different requirements
  - Retrieval / Nearest neighbors: High Memory for storing Index / Approximate index
  - Indexing : GPU for computing embeddings
  - Detection / Segmentation : GPU with higher memory
  - Video decode: GPU optional
  - Crop / Transform / Extract : CPU
- Primitives for Queue management
  - launching queues
  - Monitoring GPU Memory utilization / allocation

Routing tasks
Two methods according to memory use

Routing by task name

- Used for routing task **without** persistent memory use **between tasks**.
- E.g. perform_dataset_extraction, perform_video_decode, perform_clustering etc.
- There is no state/memory that persists between tasks.
- q_extract, q_clusterer, q_trainer

Routing by model & task name

- Used for routing task **with** persistent memory use **between tasks**.
- E.g. perform_retrieval, perform_indexing, perform_detection
- Above tasks require keeping model, index in memory. Crucial to avoid model loading overhead and memory use under control.
- q_indexer_1, q_retriever_1, q_detector_3

Launching workers at container launch vs. dynamically

Via **environment variables** at container launch

- Launch by queue_name
  E.g. LAUNCH_Q_qextract=1
- Launch by model name and task type (indexer/retriever/detector, etc.) E.g. LAUNCH_BY_NAME_indexer_inception, LAUNCH_BY_NAME_retriever_inception, LAUNCH_BY_NAME_detector_coco
- Model name gets replaced by the primary_key in the database at launch.

Dynamically via **perform_host_management**

- Launch dynamically by sending message to any host on q_manager
- Launch task “perform_host_management” With arguments specifying host_name and queue_name to consume.
- Used when new detector, indexer, analyzer, etc. models are created. Also to dynamically shutdown workers to free GPU memory.

Code organization
dvaapp & dvalib

dvaapp: a django app/project

- Handles UI and data processing
- Data model & Filesystem handling
  - Video, Frame, Region
  - Query, QueryResult
  - Event, Process etc.
- Data processing framework using Celery
  - Perform tasks
  - Manage queues
  - Monitor resource use
- Uses dvalib to carry out tasks

dvalib: library for implementing models

- Database & Message queue agnostic library
- Defines interface & implementations for
  - Detection / Indexing / Segmentation / Analysis
  - Retrieval
  - Training
- Implements models defined using PyTorch, TensorFlow and Caffe
- Can be tested independently without dvaapp

Emulating datacenter on a machine

Docker enables same codebase across all configurations {a laptop, multi-GPU machine, datacenter}

Docker-compose used for simulating distributed environment for testing and single machine deployment

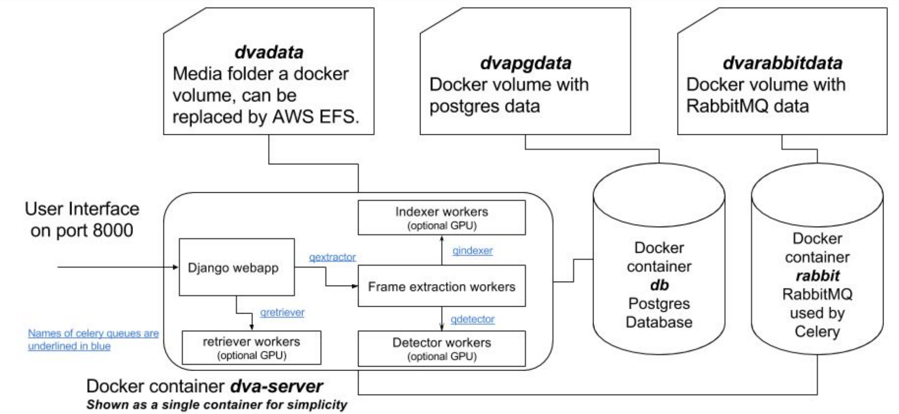
Docker container image and :tags

1. dva-auto:latest (CPU Tensorflow + PyTorch
2. dva-auto:caffe-cpu (CPU Caffe)
3. dva-auto:gpu   (GPU Tensorflow + PyTorch)
4. dva-auto:caffe (GPU Caffe)

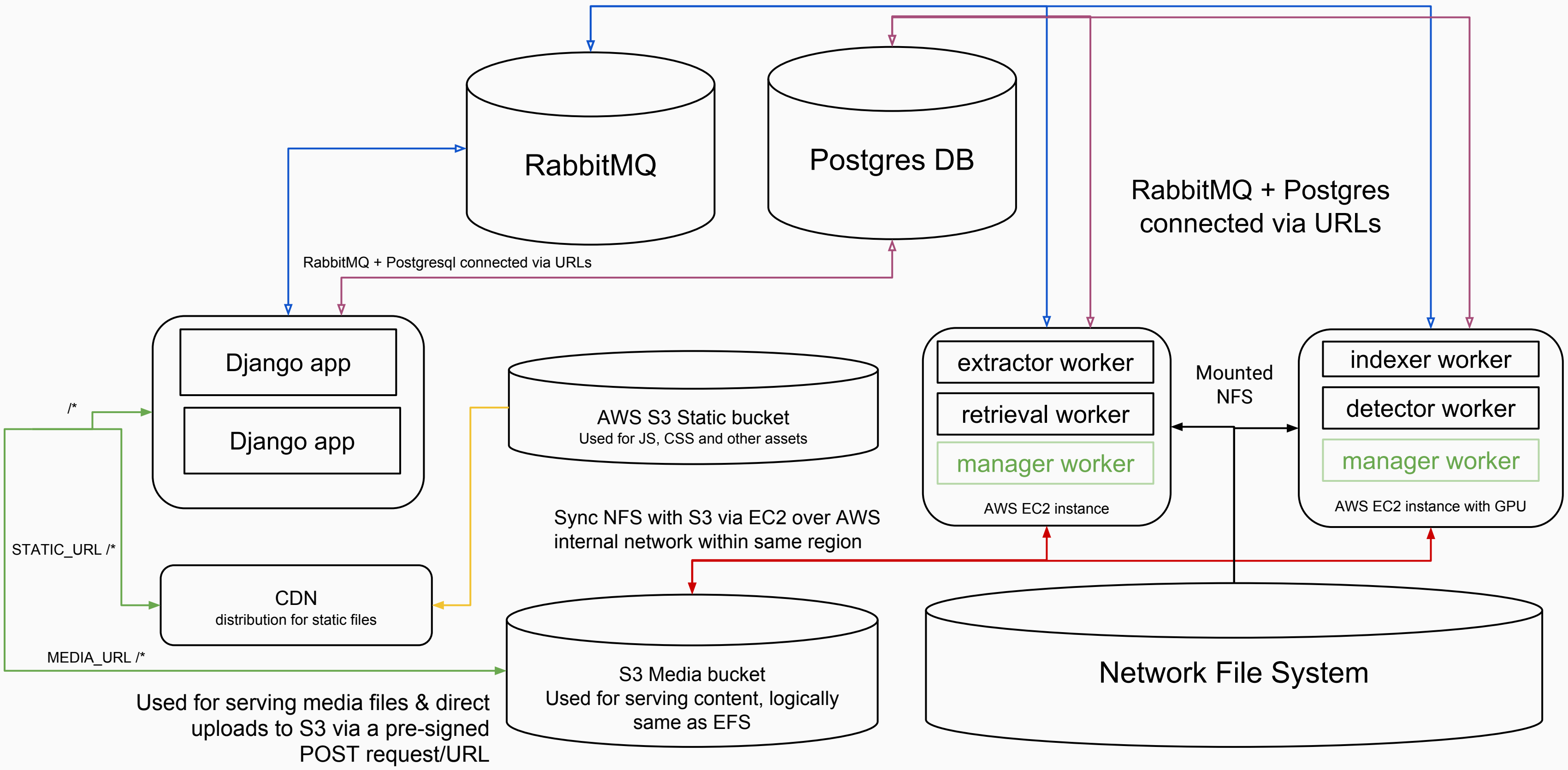
All images are automatically built on docker hub

Docker volumes

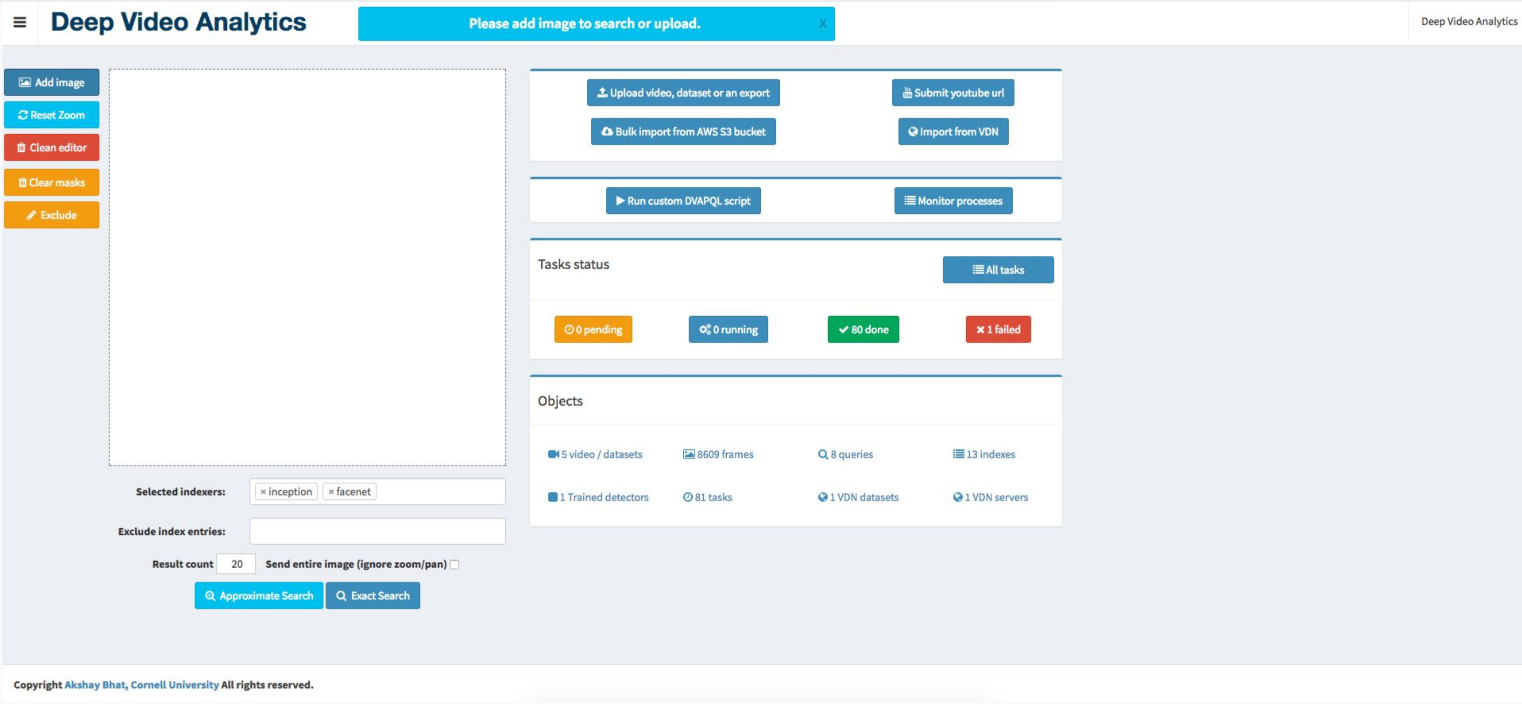
1. dvadata / shared file-system
2. dvapgdata (when DB is containerized)
3. dvarabbitdata (when rabbitmq is containerized)



Scalability with distributed architecture

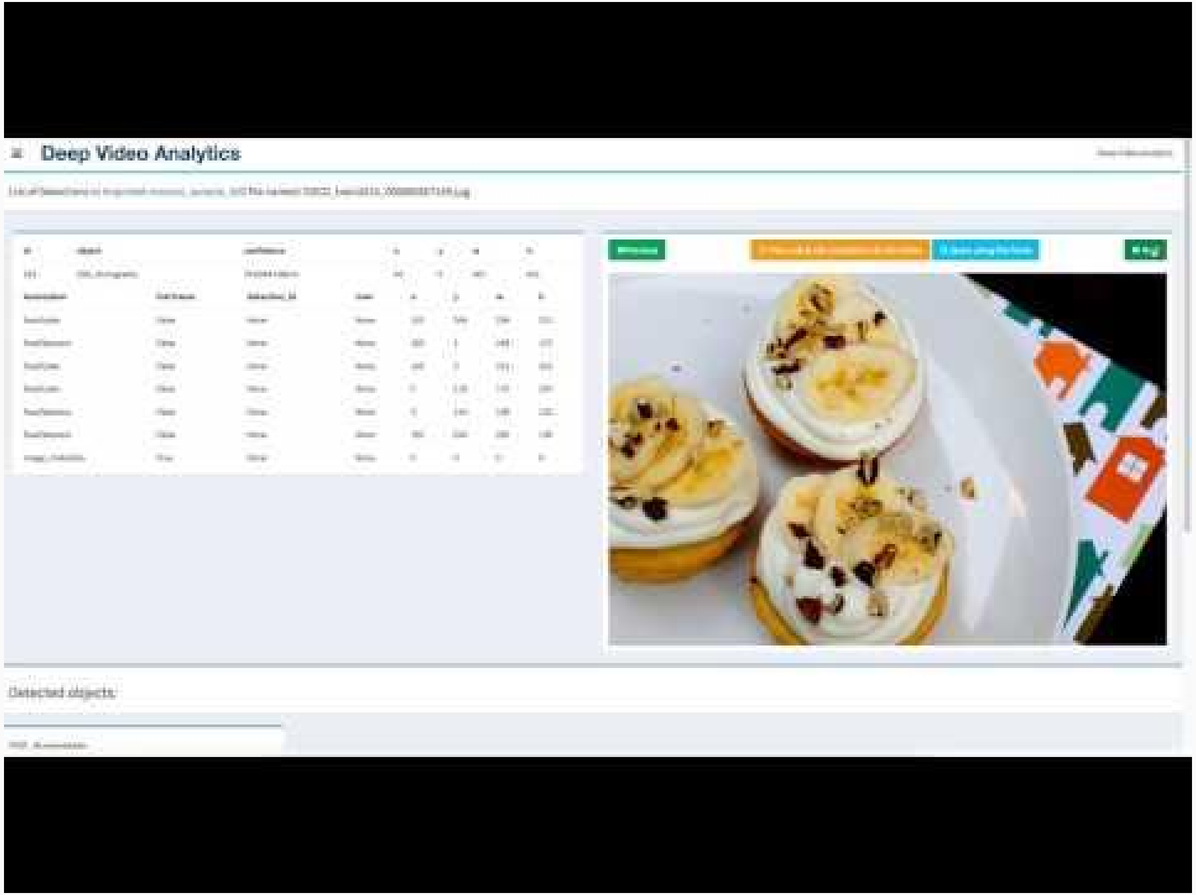


User Interface

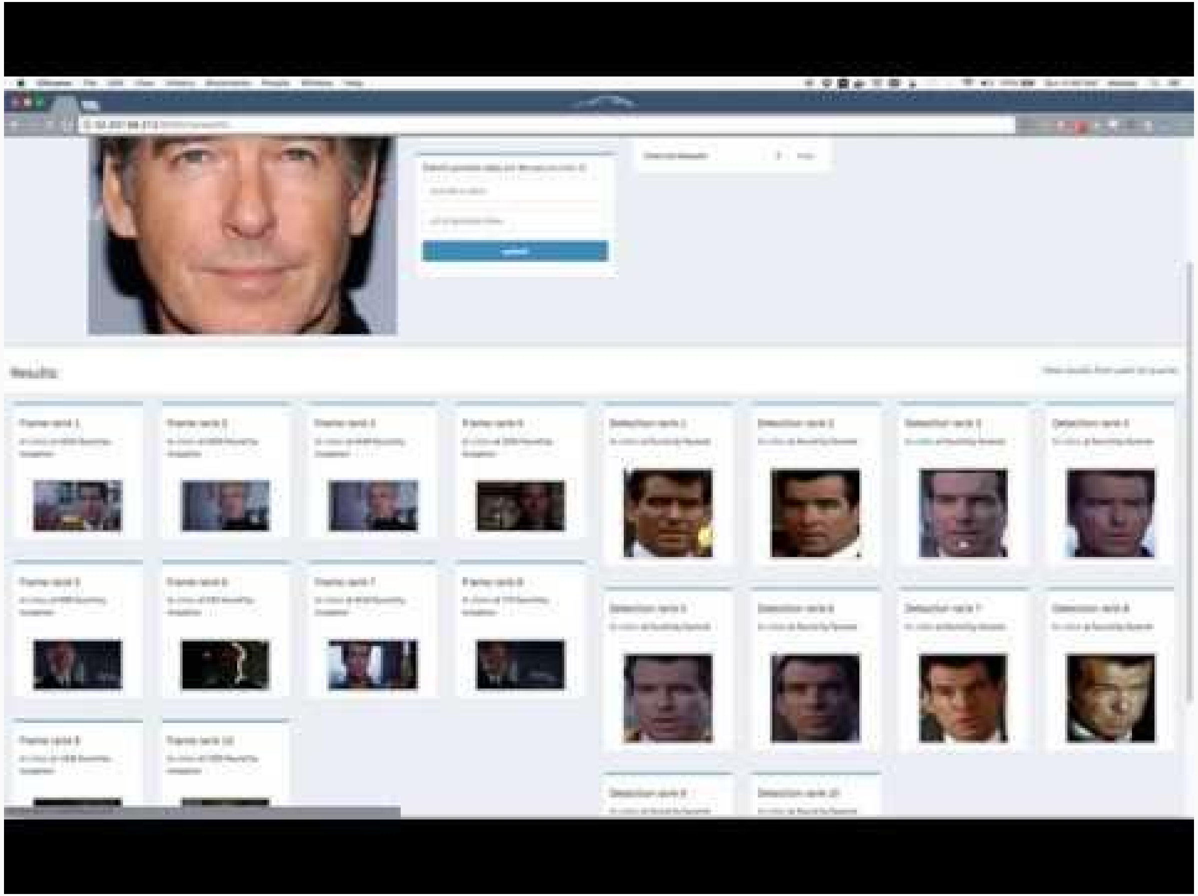


Latest version beta, 17th August 2017

7^{th} April 2017



15th March 2017



People : Facebook

::

Code : Git / GitHub, GitLab

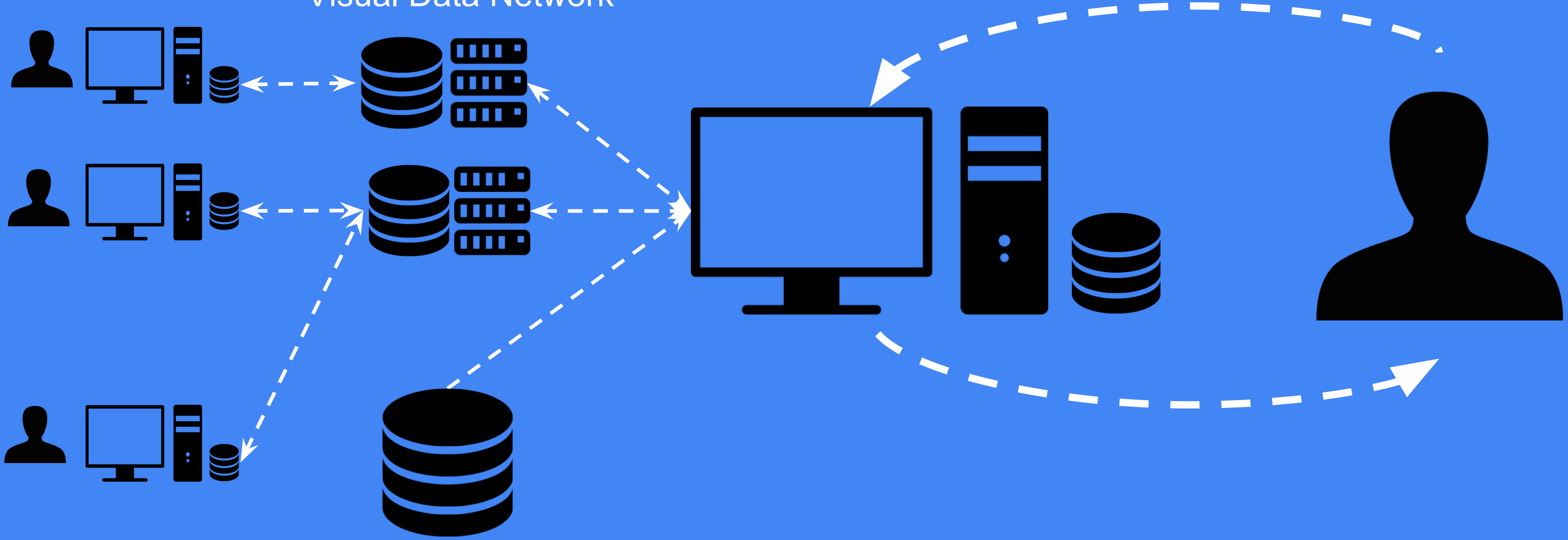
::

Visual Data: ***Visual Data Network***

Sharing data using Visual Data Network

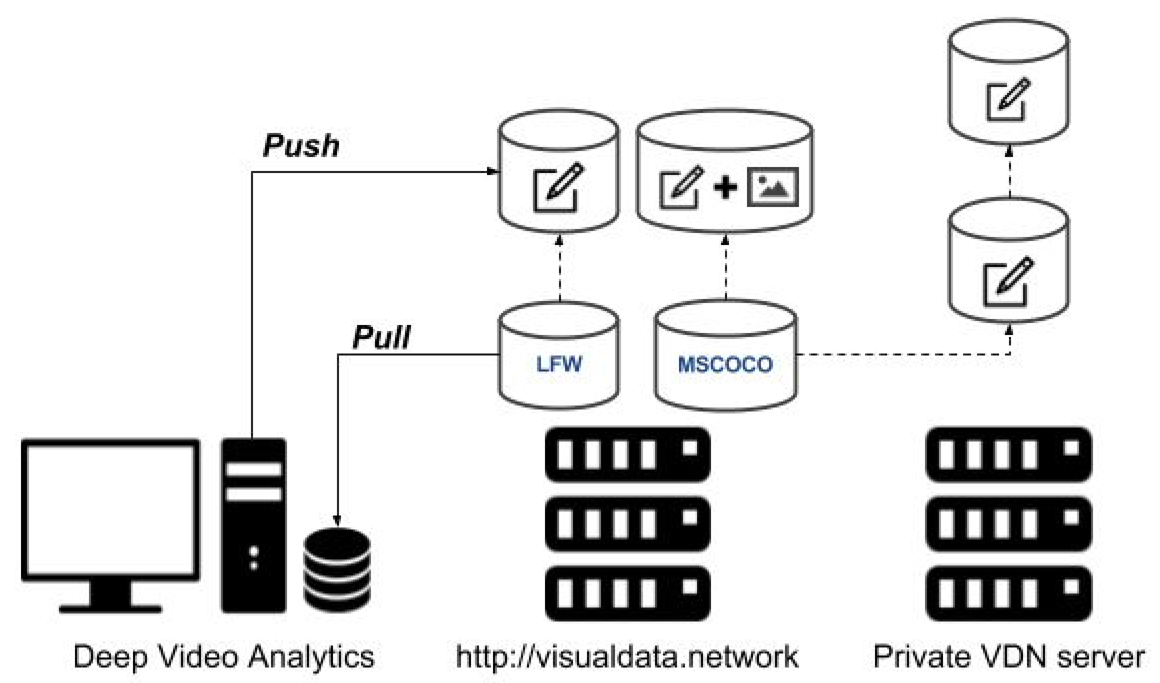
Import & export new datasets / annotations
share with other users

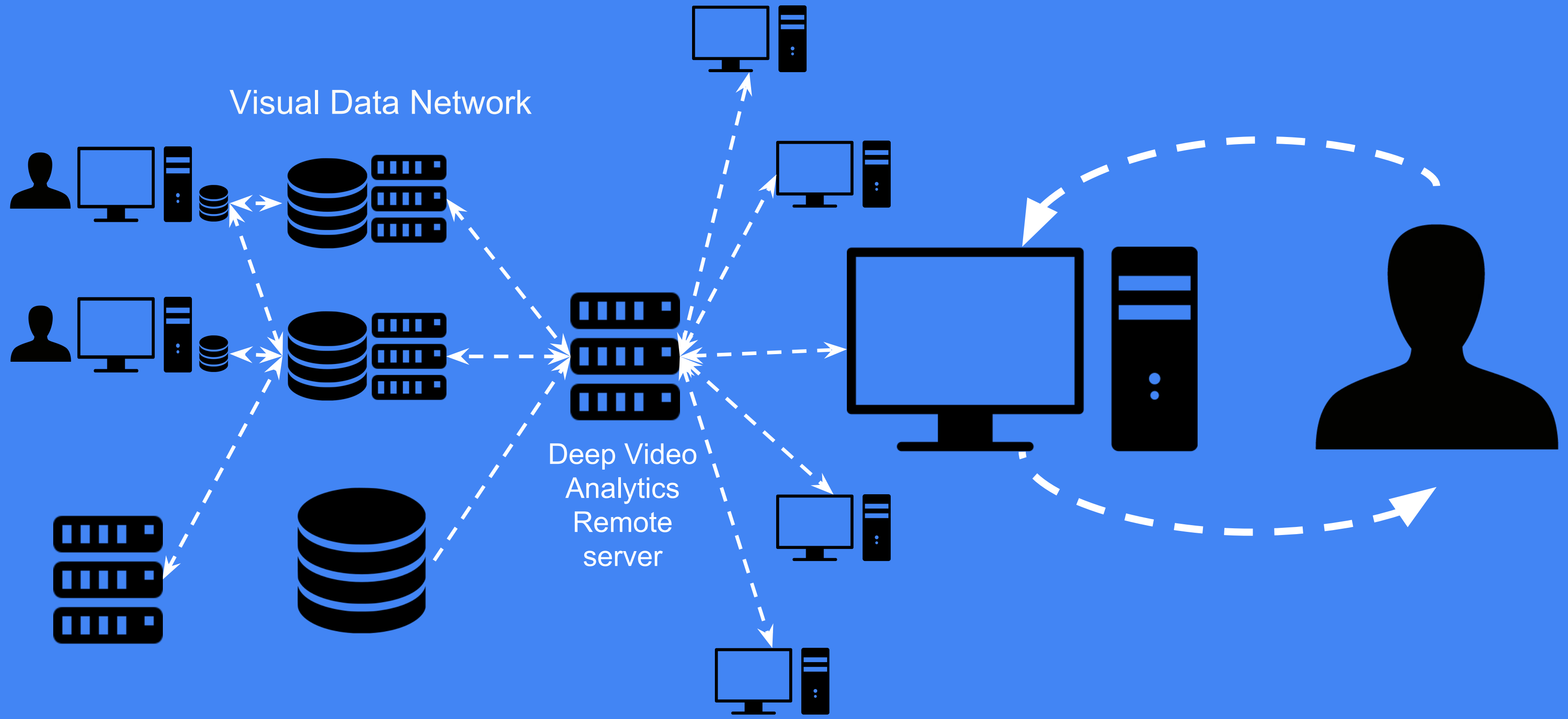
Visual Data Network



Visual Data Network enables seamless sharing

Push, Pull video / dataset, Annotations, just like you would with GitHub




Visual Data Network
Deep Video
Analytics
Remote
server

Open questions:
A work in progress

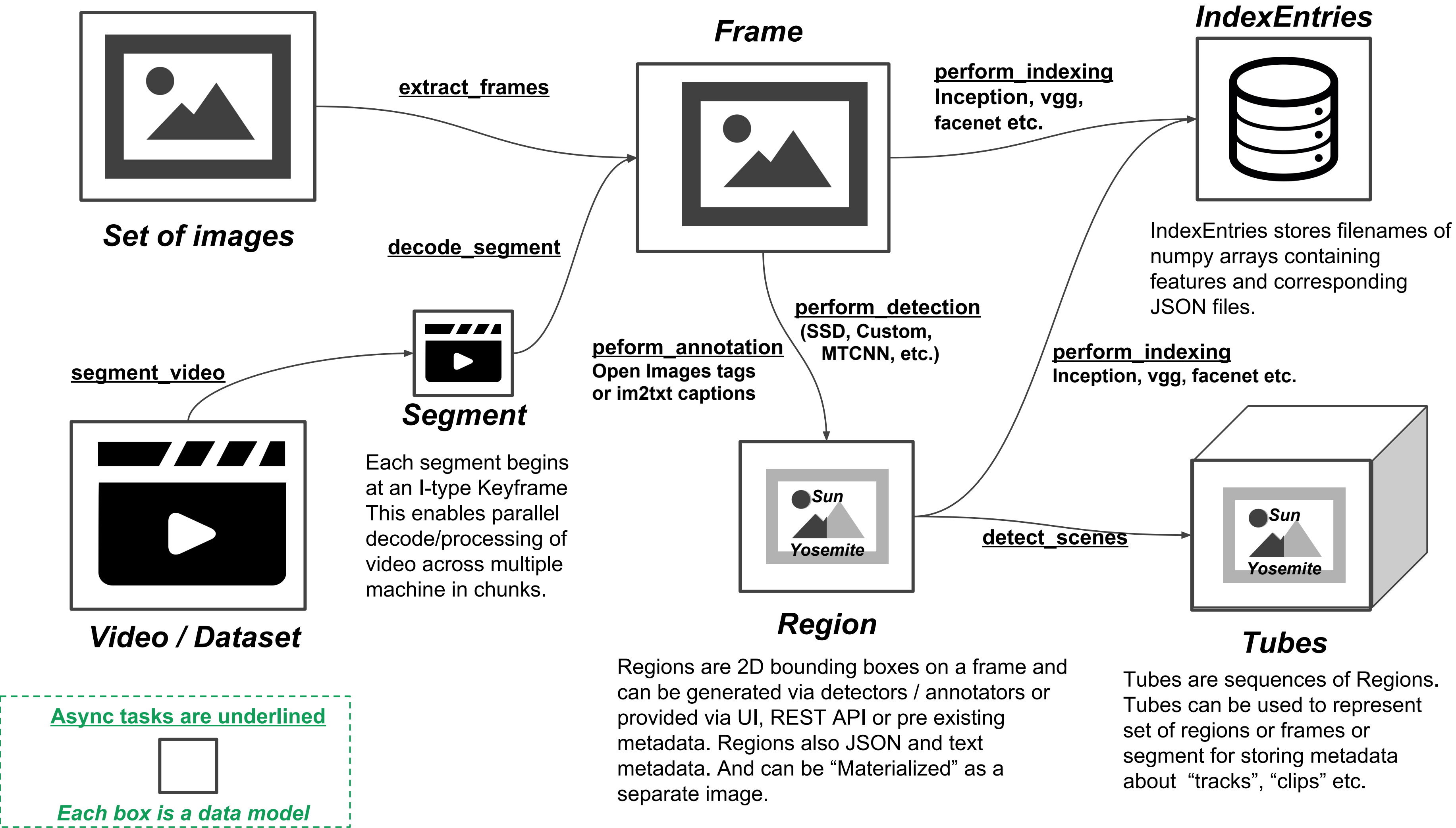
- How to effectively manage GPU memory & utilization?
- How to balance fast/static vs slow/dynamic indexes?
- How to learn continuously from annotations/feedback?
- How to minimize storage requirements via compaction?
- How to enable Real time processing?

Thanks!

Contact me

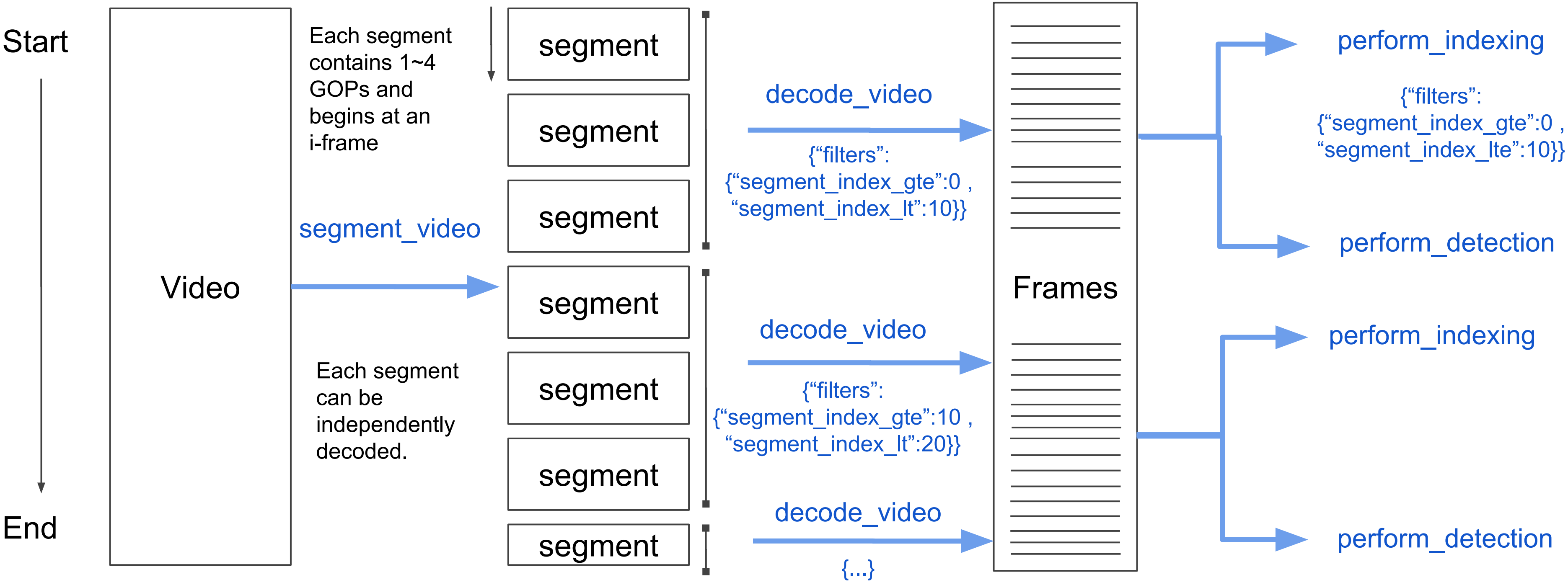
akshayubhat@gmail.com

www.akshaybhat.com



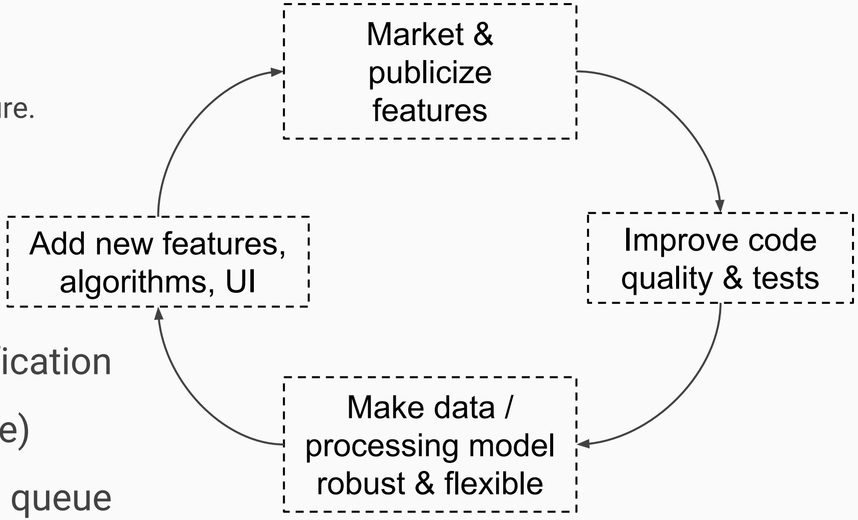
Set of images
extract_frames
Frame
perform_indexing
Inception, vgg, facenet etc.
IndexEntries
IndexEntries stores filenames of numpy arrays containing features and corresponding JSON files.
decode_segment
segment_video
Segment
Each segment begins at an I-type Keyframe This enables parallel decode/processing of video across multiple machine in chunks.
Video / Dataset
perform_detection
(SSD, Custom, MTCNN, etc.)
peform_annotation
Open Images tags or im2txt captions
perform_indexing
Inception, vgg, facenet etc.
Sun
Yosemite
Region
Regions are 2D bounding boxes on a frame and can be generated via detectors / annotators or provided via UI, REST API or pre existing metadata. Regions also JSON and text metadata. And can be “Materialized” as a separate image.
detect_scenes
Sun
Yosemite
Tubes
Tubes are sequences of Regions. Tubes can be used to represent set of regions or frames or segment for storing metadata about “tracks”, “clips” etc.
Async tasks are underlined
Each box is a data model

Distributed processing using hierarchical tasks



Software Development approach or "How I developed Deep Video Analytics"

Partly inspired by "**Worse is better**"



- Start at "final scale" at which it's intended to be used
  - Easy to optimize each component, difficult to change architecture.
- Write "high level" tests rather than "unit tests"
  - E.g load video -> extract frames -> build index -> query
- Observability is crucial, develop UI for visual inspection
- Create start-from-zero config and use it for manual verification
- Keep everything in a single repo (including User Interface)
- **DO NOT** write a new database or roll your own message queue
  - Both Postgres and RabbitMQ are natively / cheaply supported in Travis / Heroku
  - It's a nightmare to debug concurrency primitives also difficult to convince others to trust / maintain your code.
- Optimize for one goal (Features, Correctness, Consistency, Simplicity) at a time (over days / week)
  - E.g. Trade consistency/quality when adding new features. Once feature is done/verified/popular improve code quality. Once code quality has improved, transition to a more consistent / simple model. Use consistency to add new features.