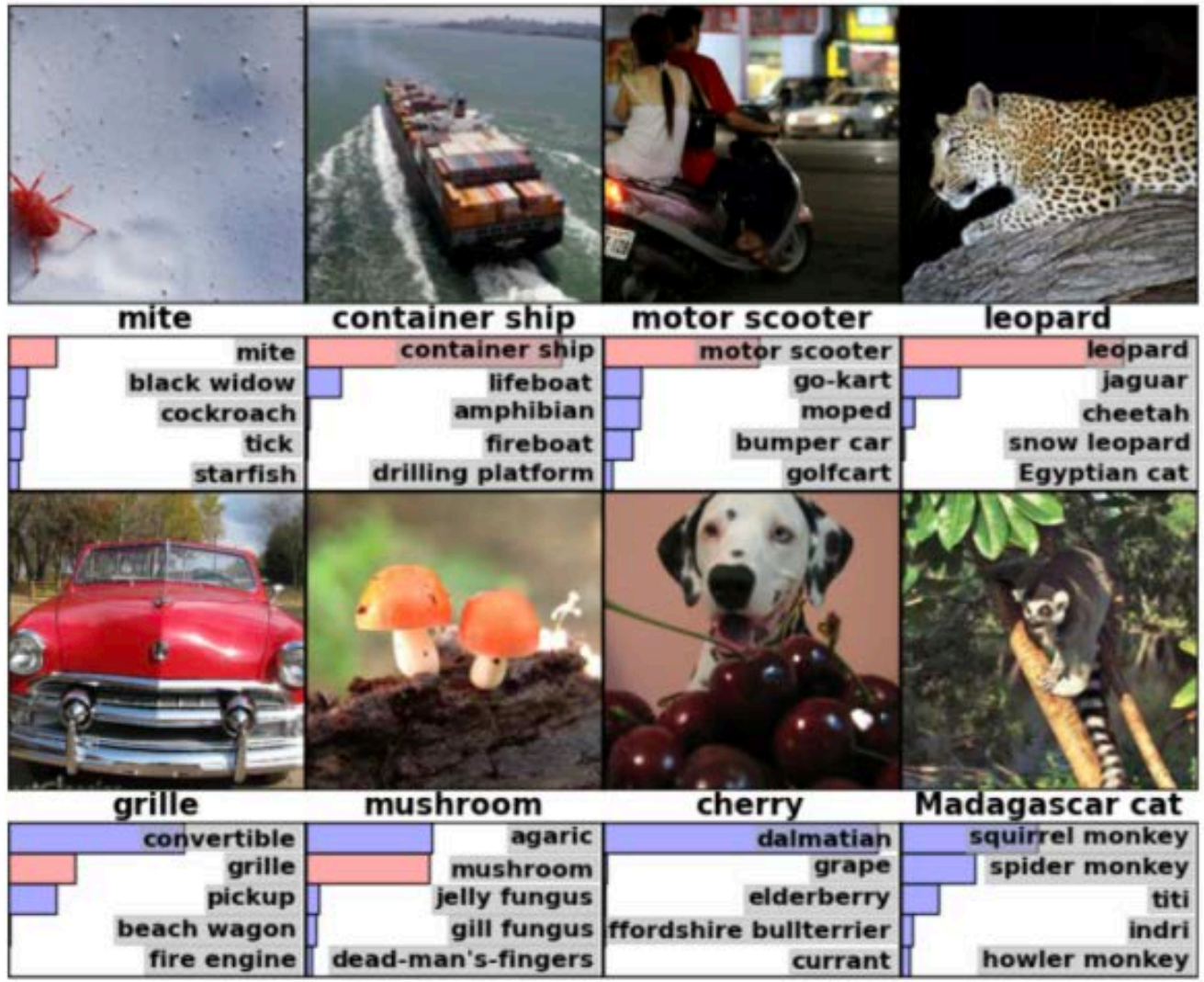
Deep Learning Inference as a Service

Mohammad Babaeizadeh
Hadi Hashemi
Chris Cai

Advisor: Prof Roy H. Campbell


mite
mite
black widow
cockroach
tick
starfish
container ship
container ship
lifeboat
amphibian
fireboat
drilling platform
motor scooter
motor scooter
go-kart
moped
bumper car
golfcart
leopard
leopard
jaguar
cheetah
snow leopard
Egyptian cat
grille
convertible
grille
pickup
beach wagon
fire engine
mushroom
agaric
mushroom
jelly fungus
gill fungus
dead-man's-fingers
cherry
dalmatian
grape
elderberry
ffordshire bullterrier
currant
Madagascar cat
squirrel monkey
spider monkey
titi
indri
howler monkey

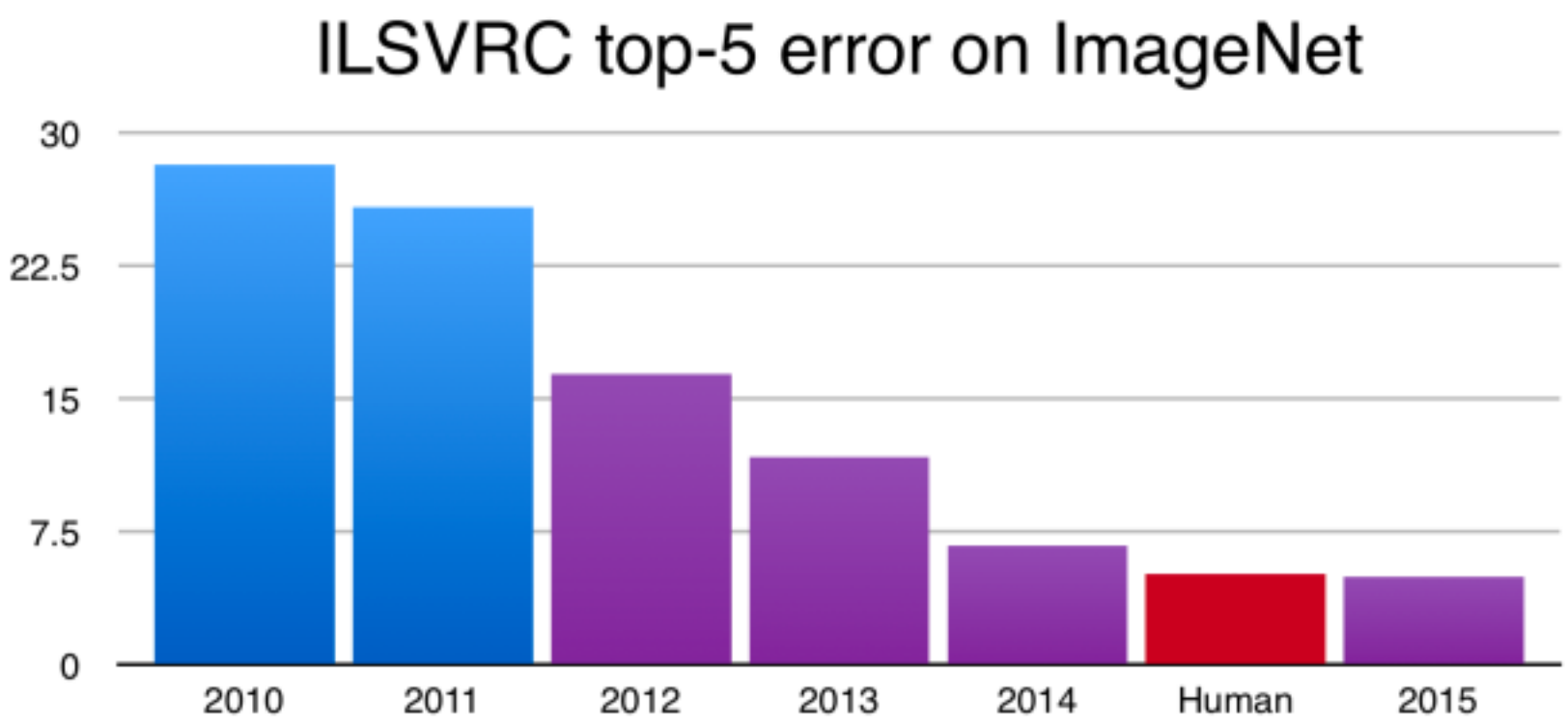
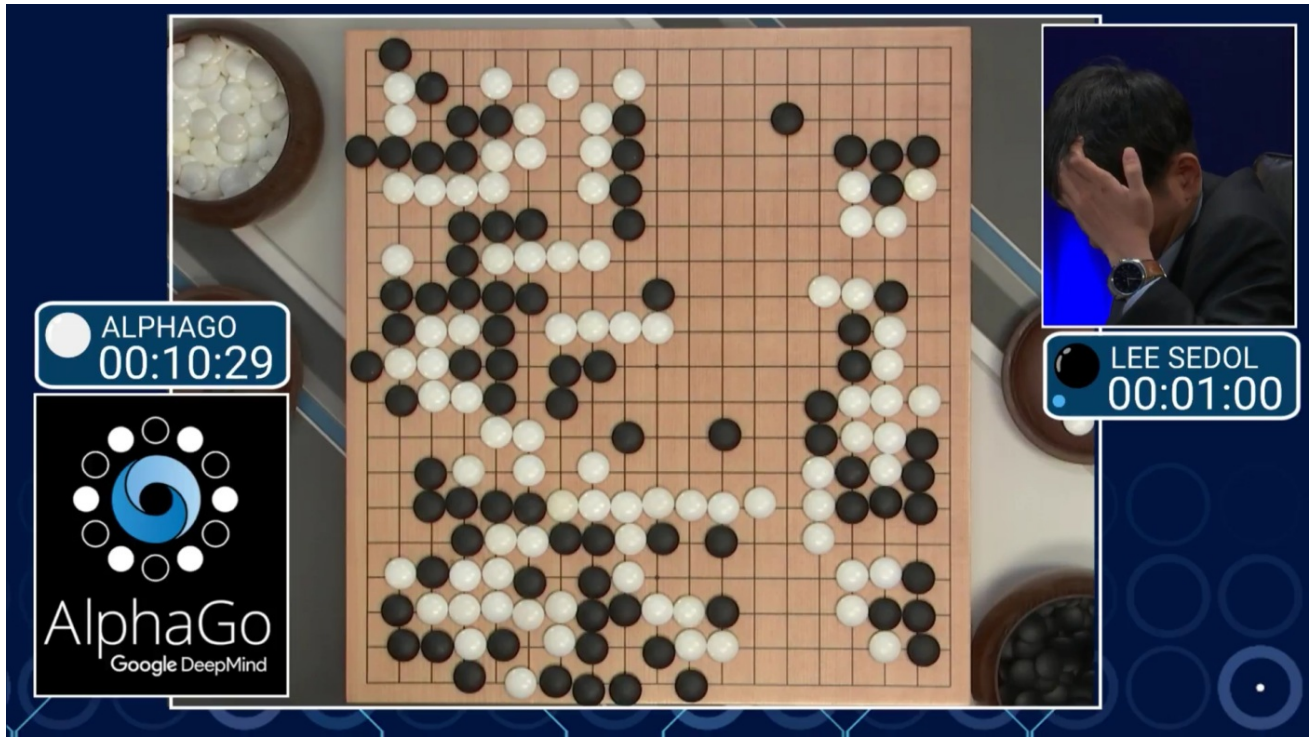
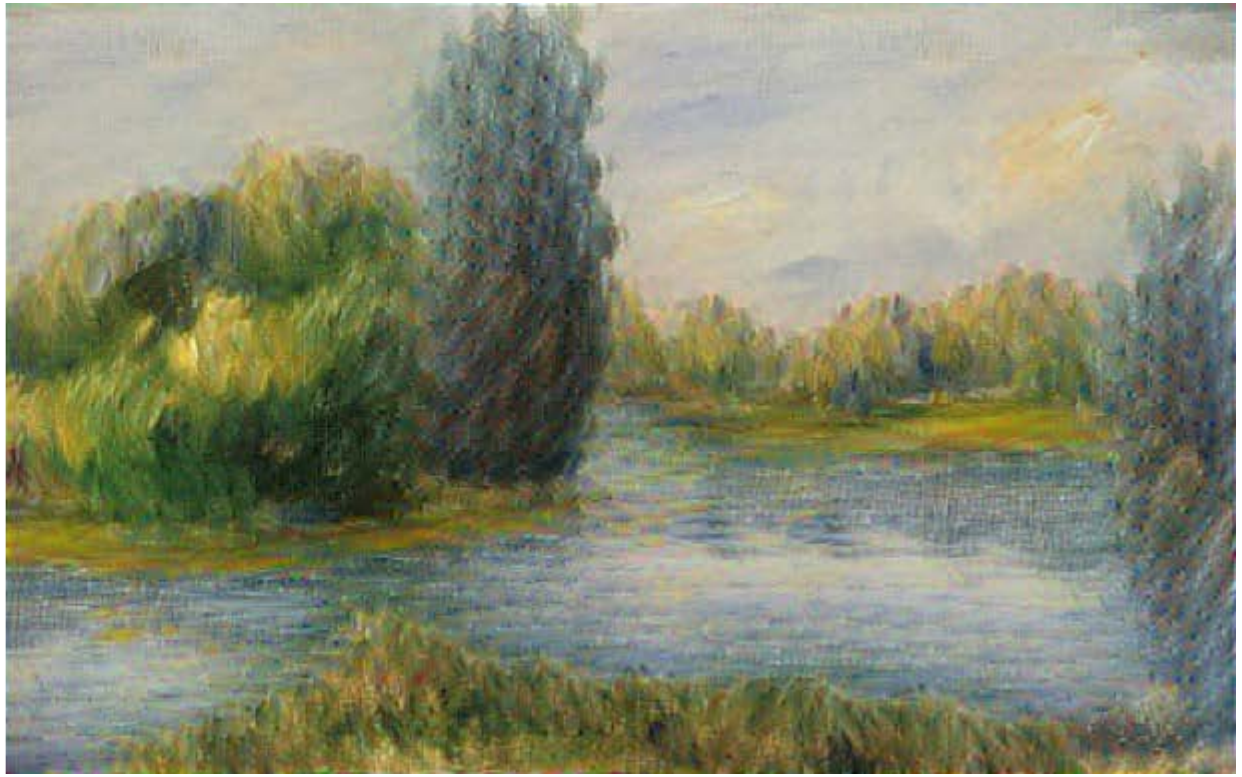

ILSVRC top-5 error on ImageNet
30
22.5
15
7.5
0
2010
2011
2012
2013
2014
Human
2015


image.display
000 5 1


91
91%
191%
MakeAGIF.com


ALPHAGO
00:10:29
LEE SEDOL
00:01:00
AlphaGo
Google DeepMind

Synthesized Image

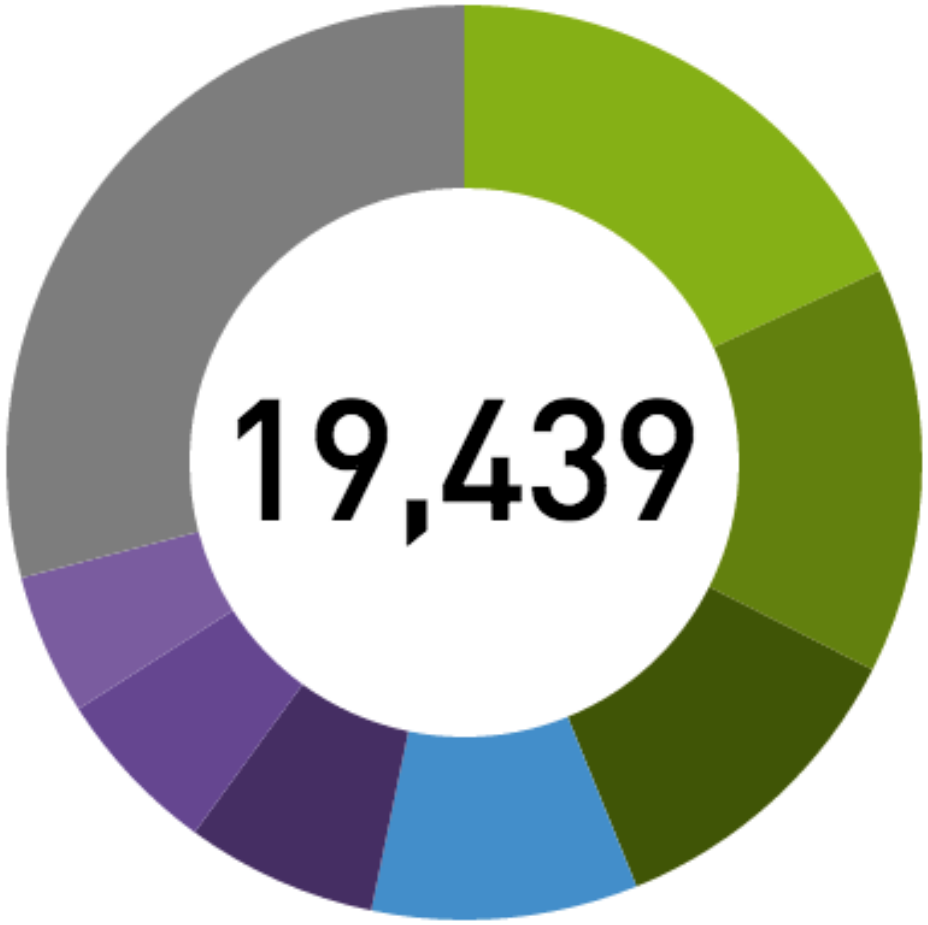


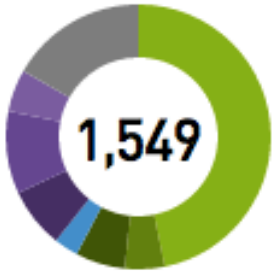
#NeuralDoodle


FaceApp

Organizations Engaged with NVIDIA on Deep Learning

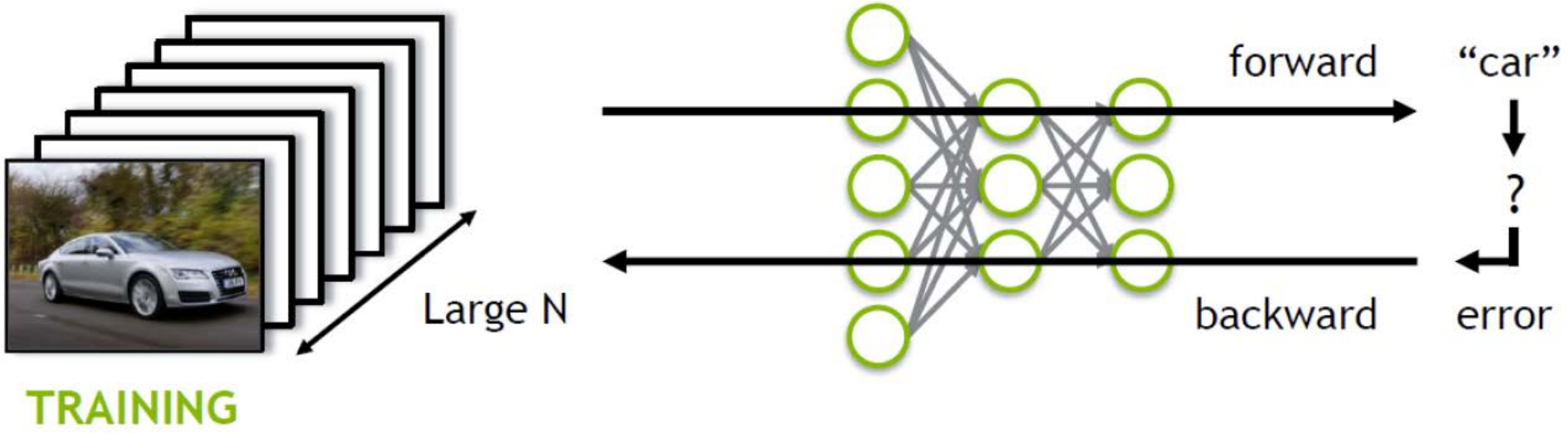
- Higher Education
- Development Tools
- Internet
- Automotive
- Finance
- Government
- Life Science
- Other

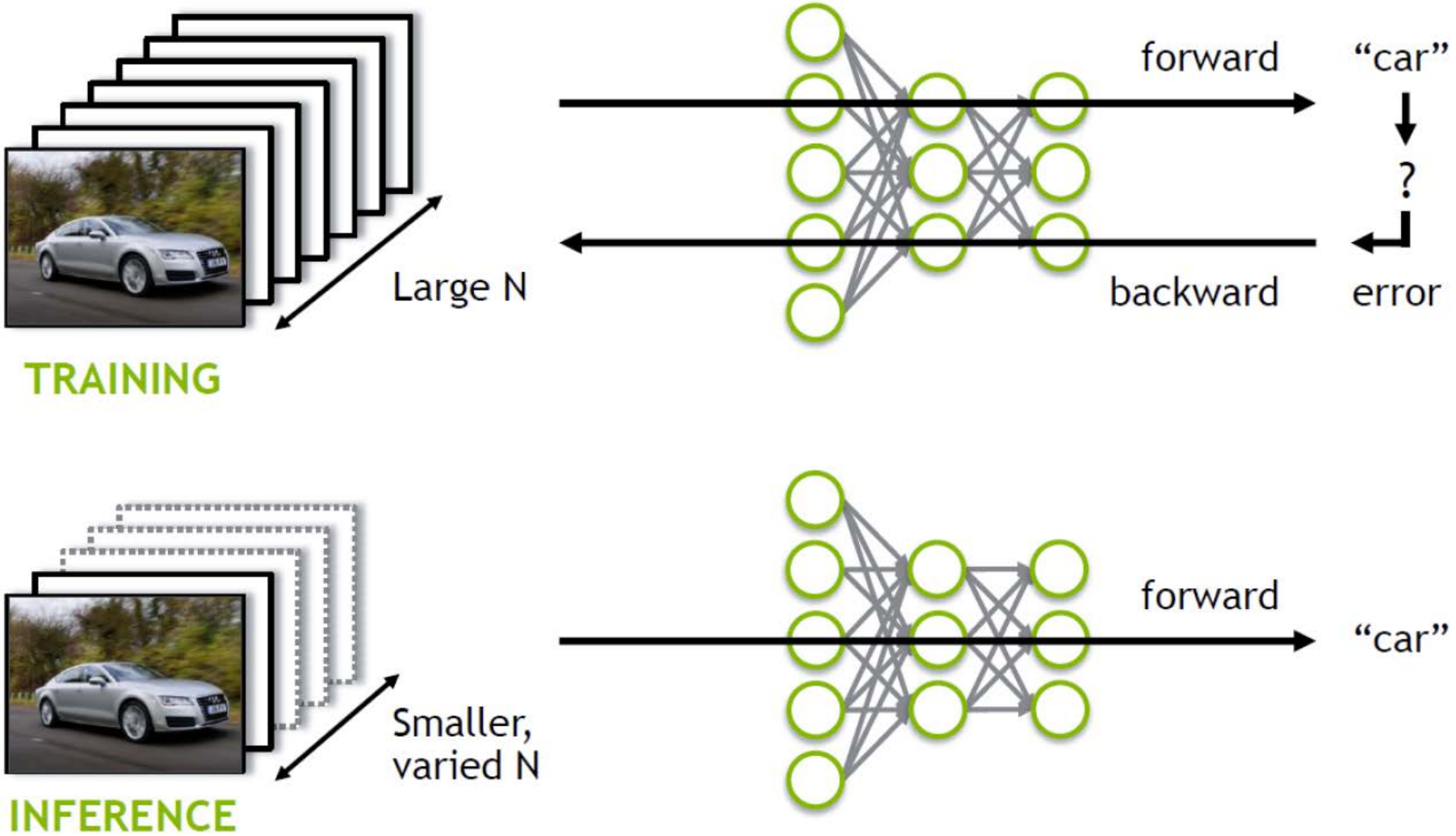




2014

2016

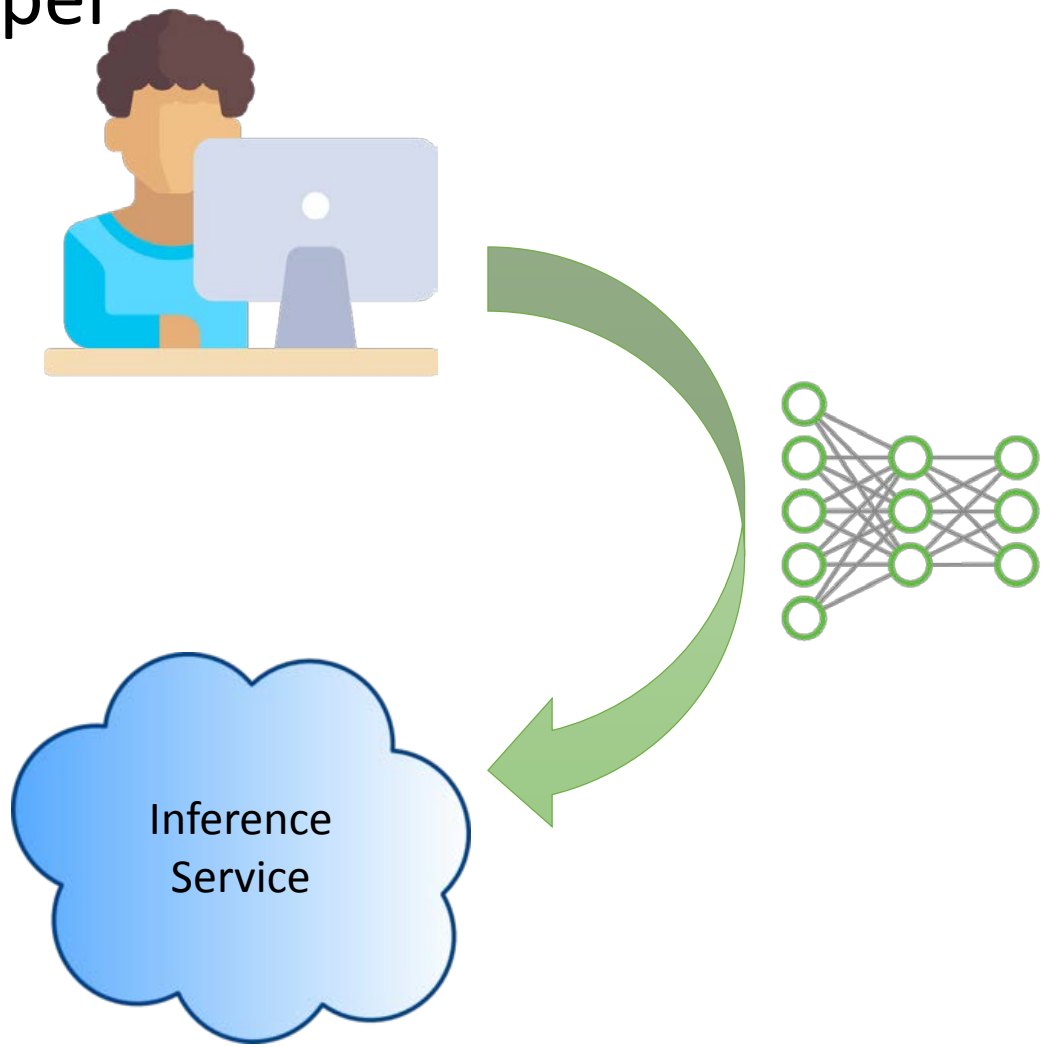

Large N
TRAINING
forward
"car"
?
backward
error


Large N
TRAINING
forward
"car"
?
backward
error
Smaller,
varied N
INFERENCE
forward
"car"

Use case 1: Model Developer

TensorFlow™

Use case 1: Model Developer

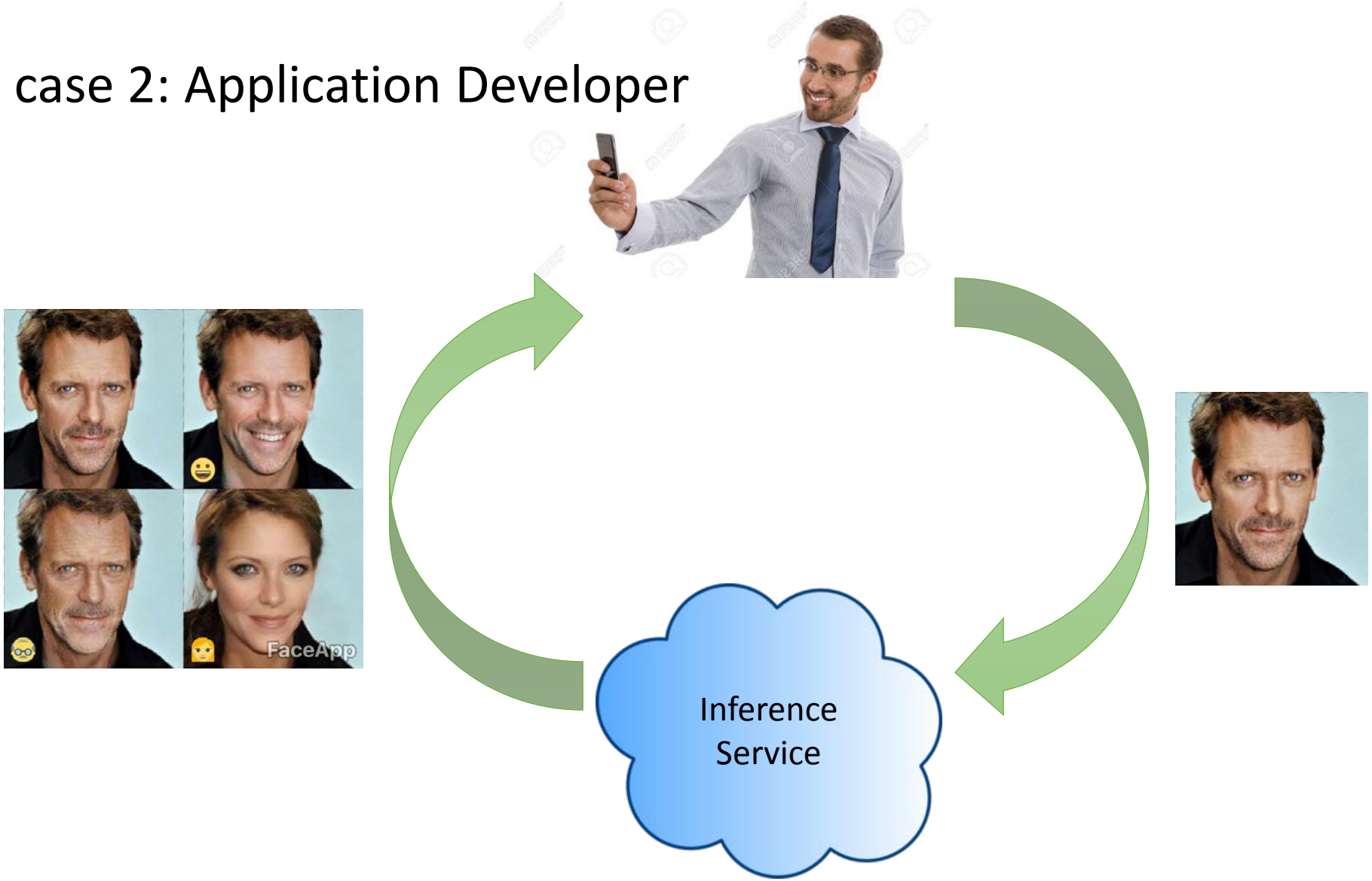


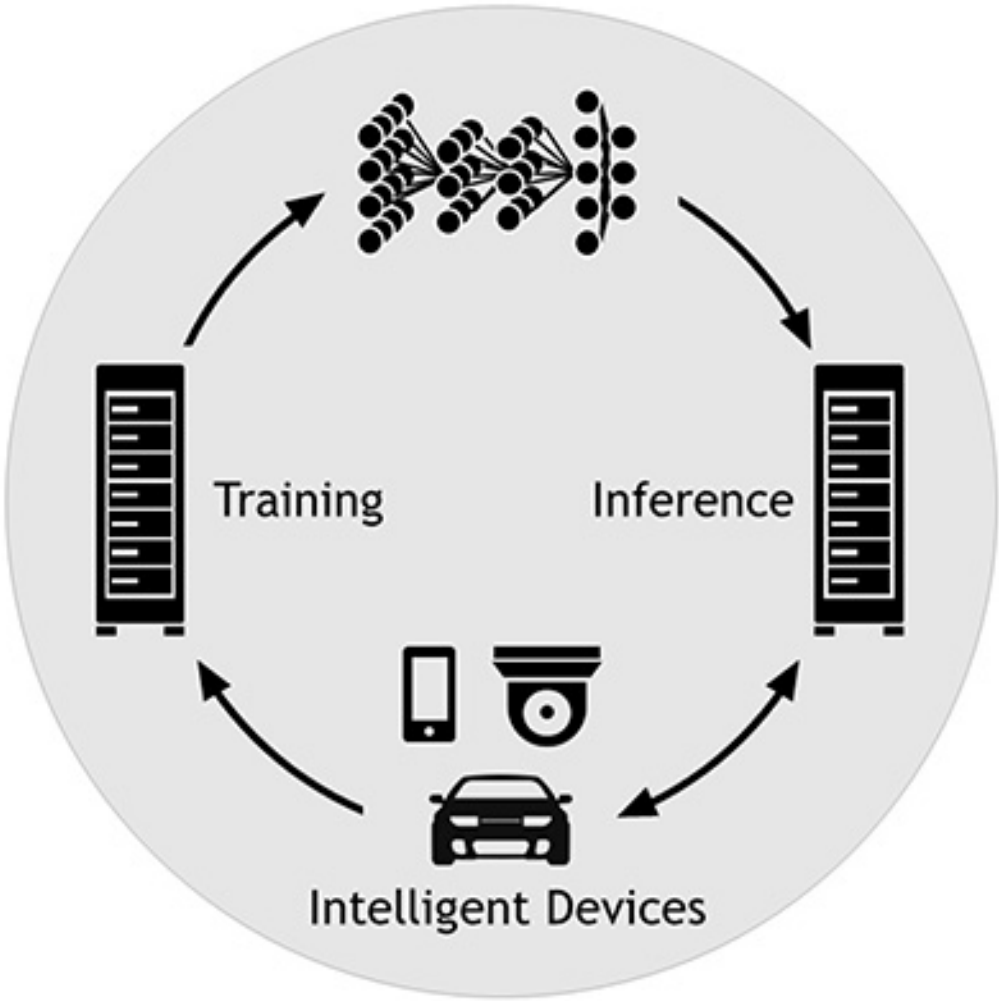
Use case 2: Application Developer

Use case 2: Application Developer

Inference Service

Use case 2: Application Developer




Training
Inference
Intelligent Devices

Caffe

Chainer

DL4J
Deeplearning4j

MINERVA

mxnet

Purine

KERAS

Microsoft
CNTK

MatConvNet

TensorFlow

theano

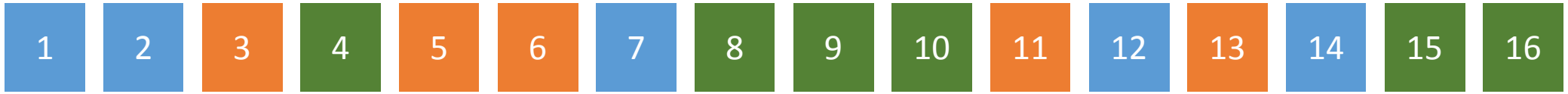
torch

TensorFlow

Problem Formulation

Models

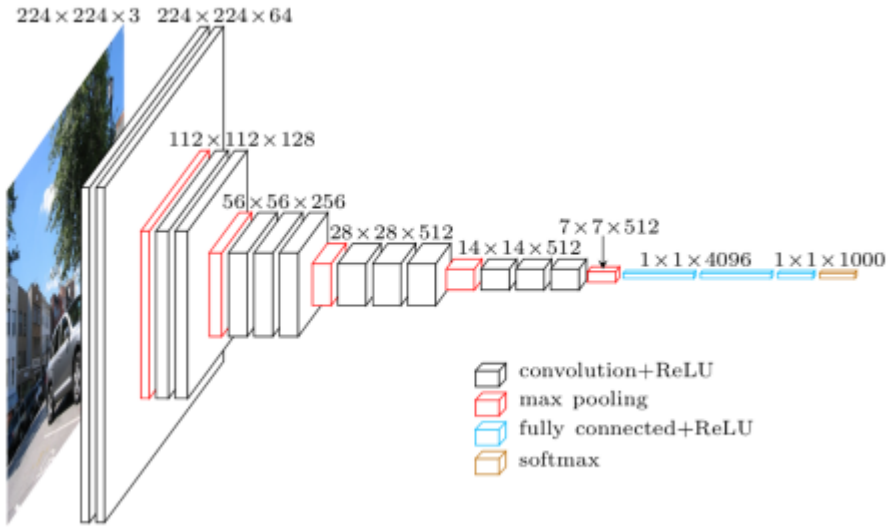
Queries



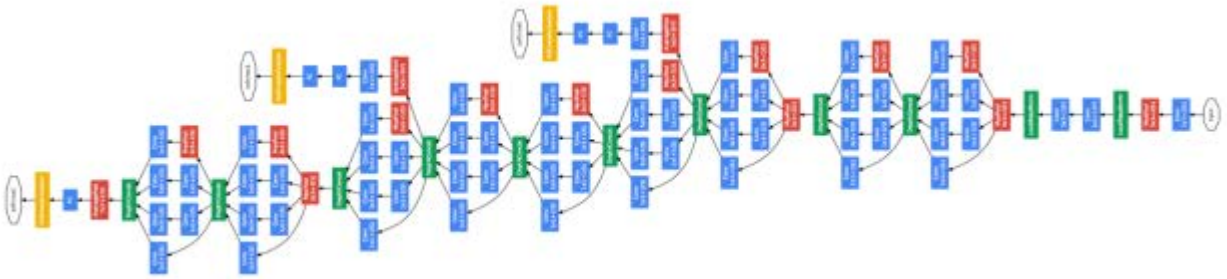
Nodes

Characteristics of DNNs


Disclaimer

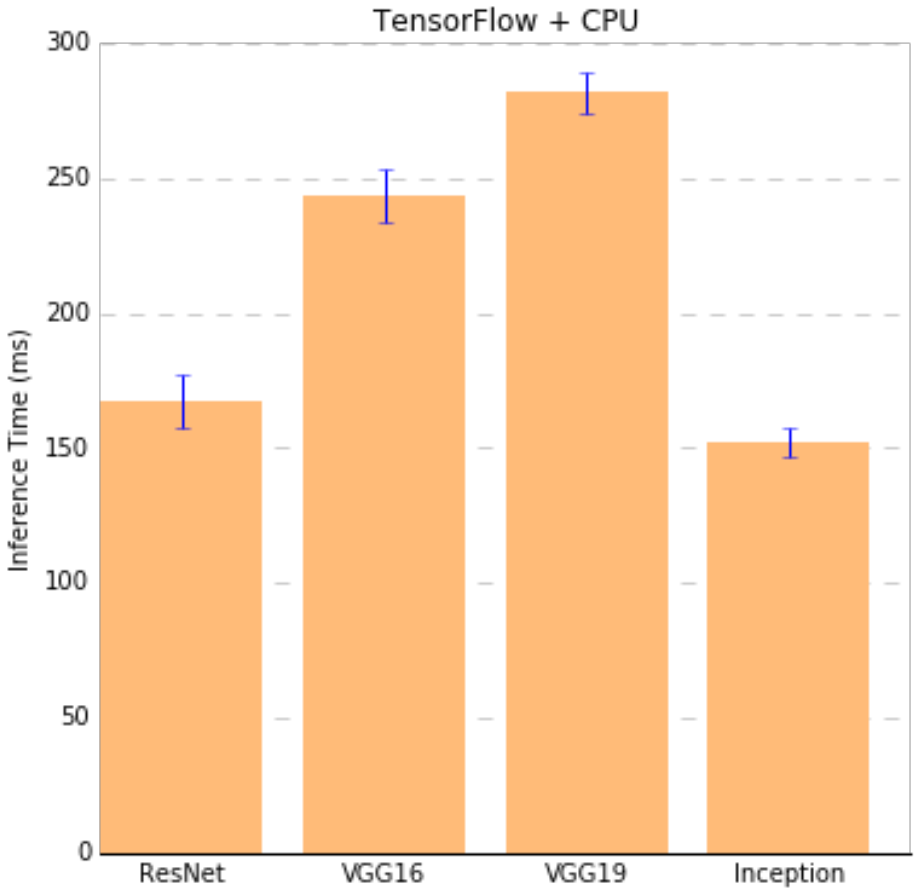


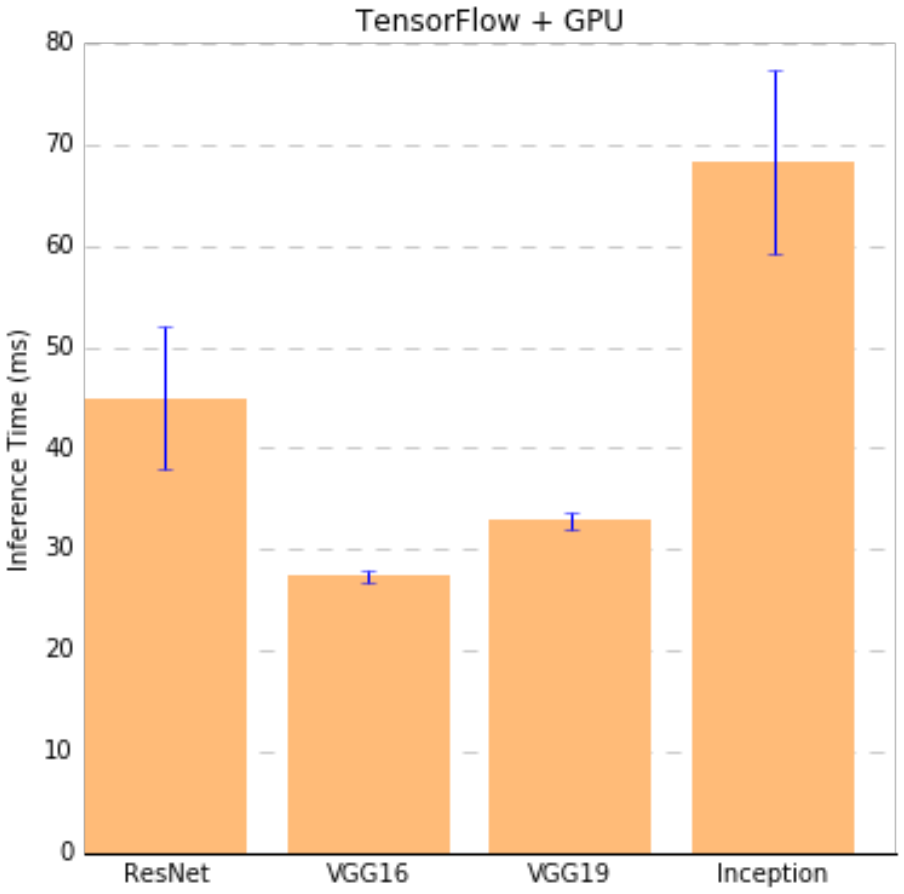
VGG 16



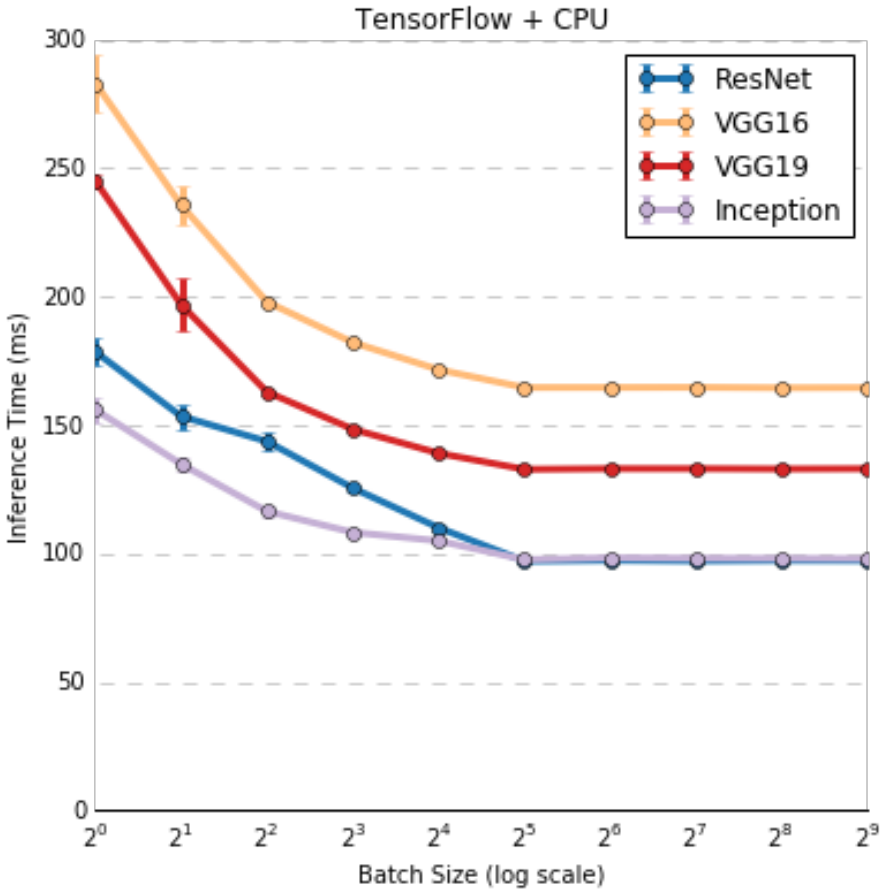
Inception

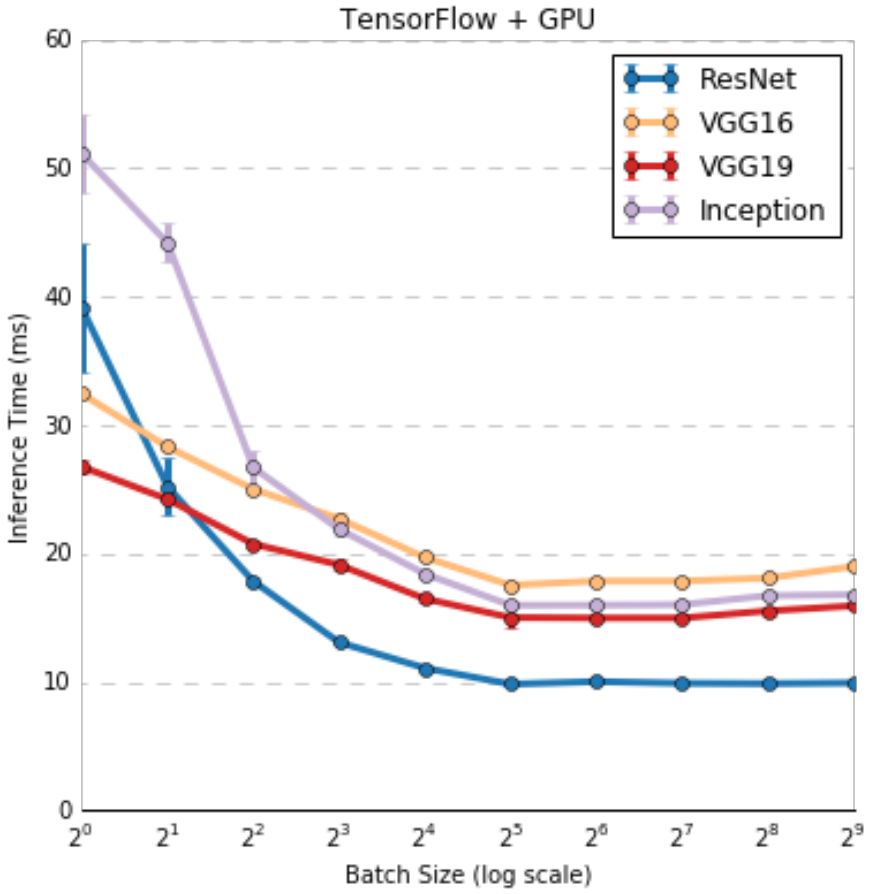
Constant Runtime





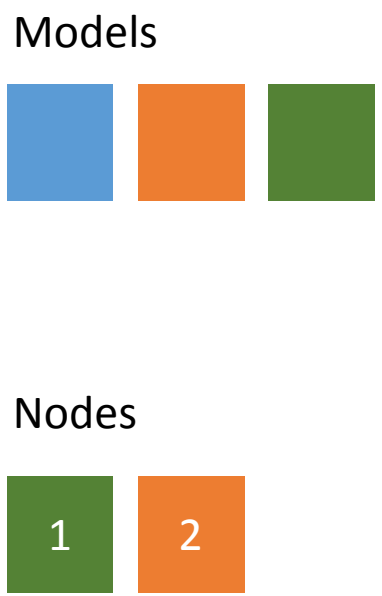
Batch Size (Vectorized Computation)





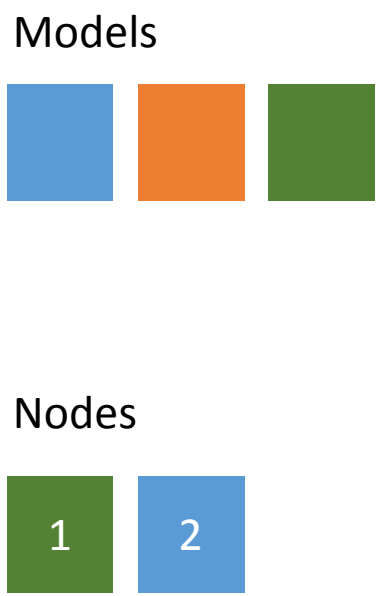
Stateless

- Embarrassingly parallel workload
  - No centralized data storage
- Load/unload models as necessary
  - No data synchronization is needed for load/unload
  - Load/unload is as expensive as running a process
- Light-weight fault-tolerance
  - Bookkeeping only for queries



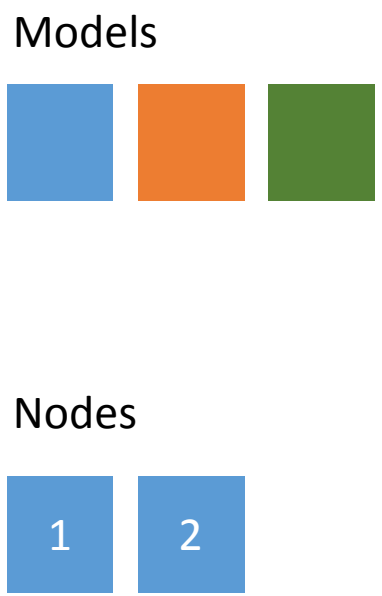
Stateless

- Embarrassingly parallel workload
  - No centralized data storage
- Load/unload models as necessary
  - No data synchronization is needed for load/unload
  - Load/unload is as expensive as running a process
- Light-weight fault-tolerance
  - Bookkeeping only for queries



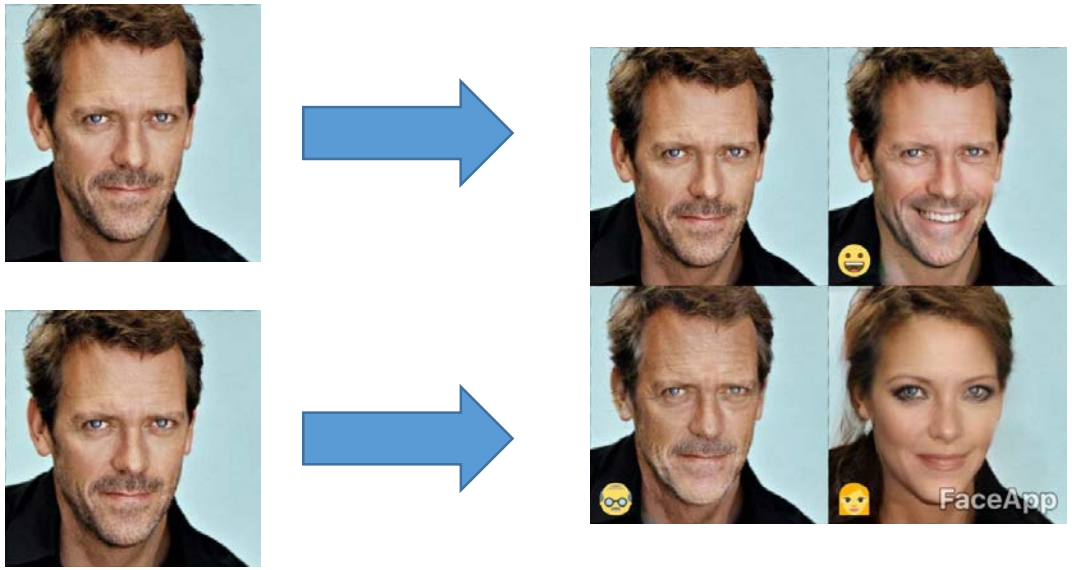
Stateless

- Embarrassingly parallel workload
  - No centralized data storage
- Load/unload models as necessary
  - No data synchronization is needed for load/unload
  - Load/unload is as expensive as running a process
- Light-weight fault-tolerance
  - Bookkeeping only for queries



Deterministic

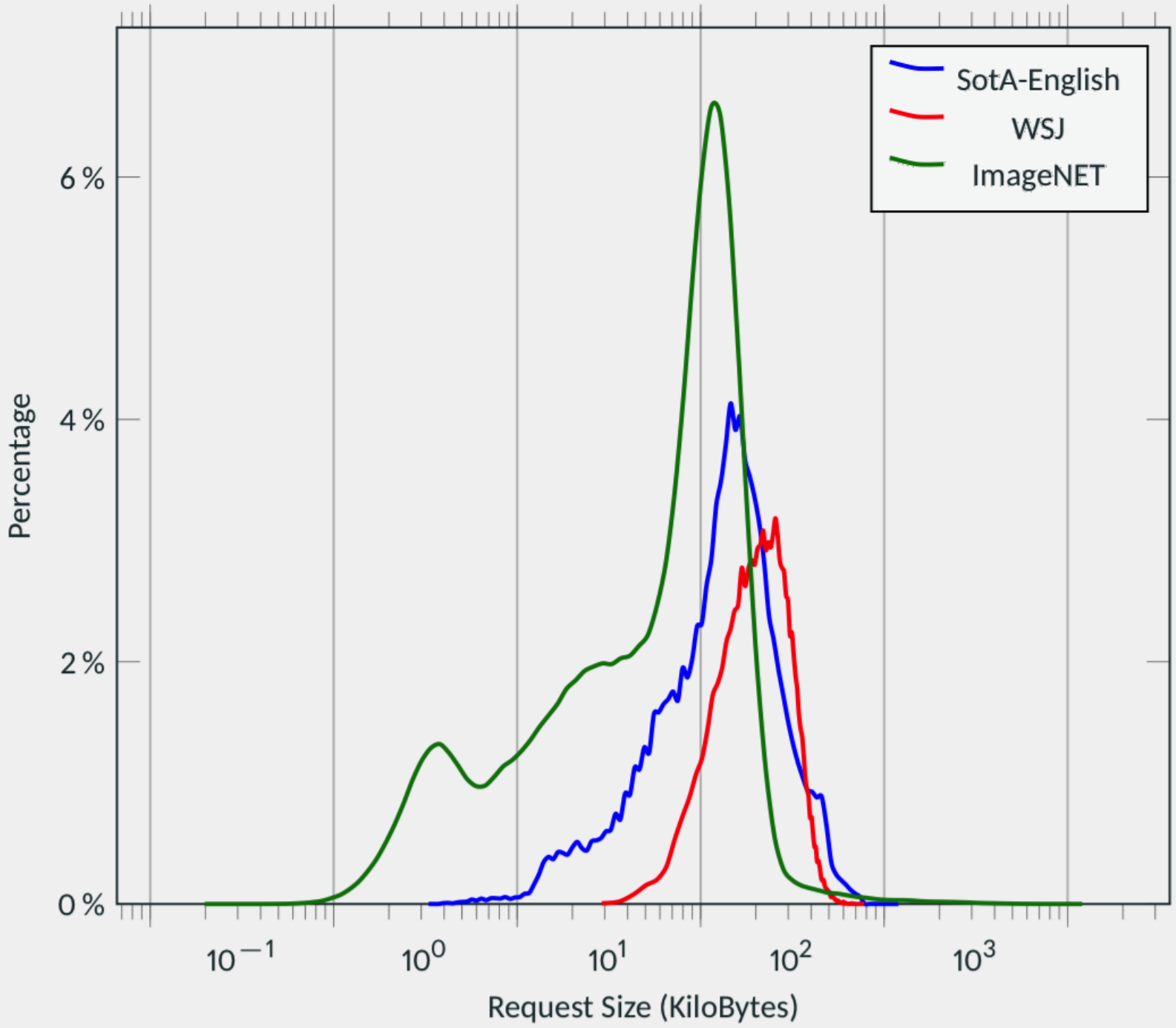
- Output is always the same for the same input
- An effective caching mechanism



Query Characteristics

Query Patterns

- Offline queries
  - Batch queries with high latency (hours) SLA
- Online stateless queries
  - Single queries with low latency (100s ms) SLA
- Online stateful queries
  - Session with a sequence of queries, each with low latency (100s ms) SLA
  - Session lifetime is of minutes


SotA-English
WSJ
ImageNET
Percentage
6 %
4 %
2 %
0 %
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}
Request Size (KiloBytes)

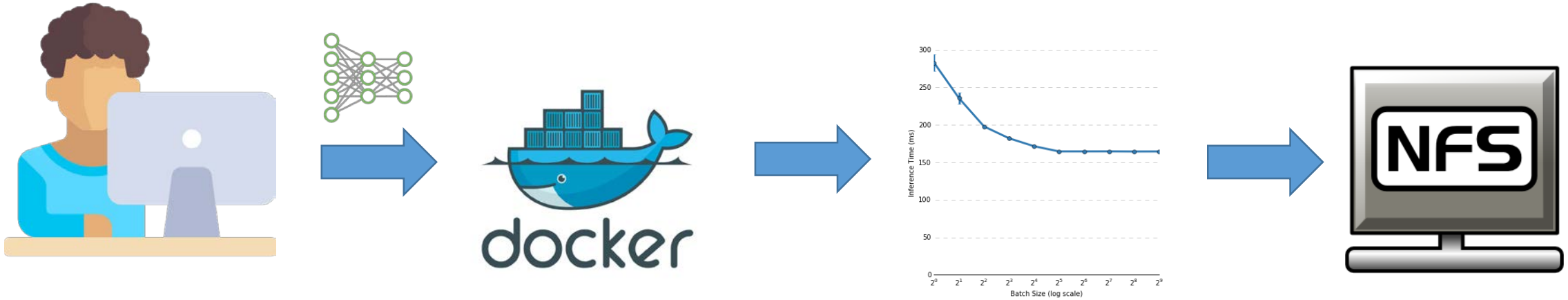
Inference as a Service

Problem Statement

- To serve arbitrary number of models with different service-level objections on minimum resource.
  - Number of models >> Number of nodes (in contrast with previous works)
  - Isolating overall performance of the system from model's. (vectorization is optional)
  - Low overhead on request response time.

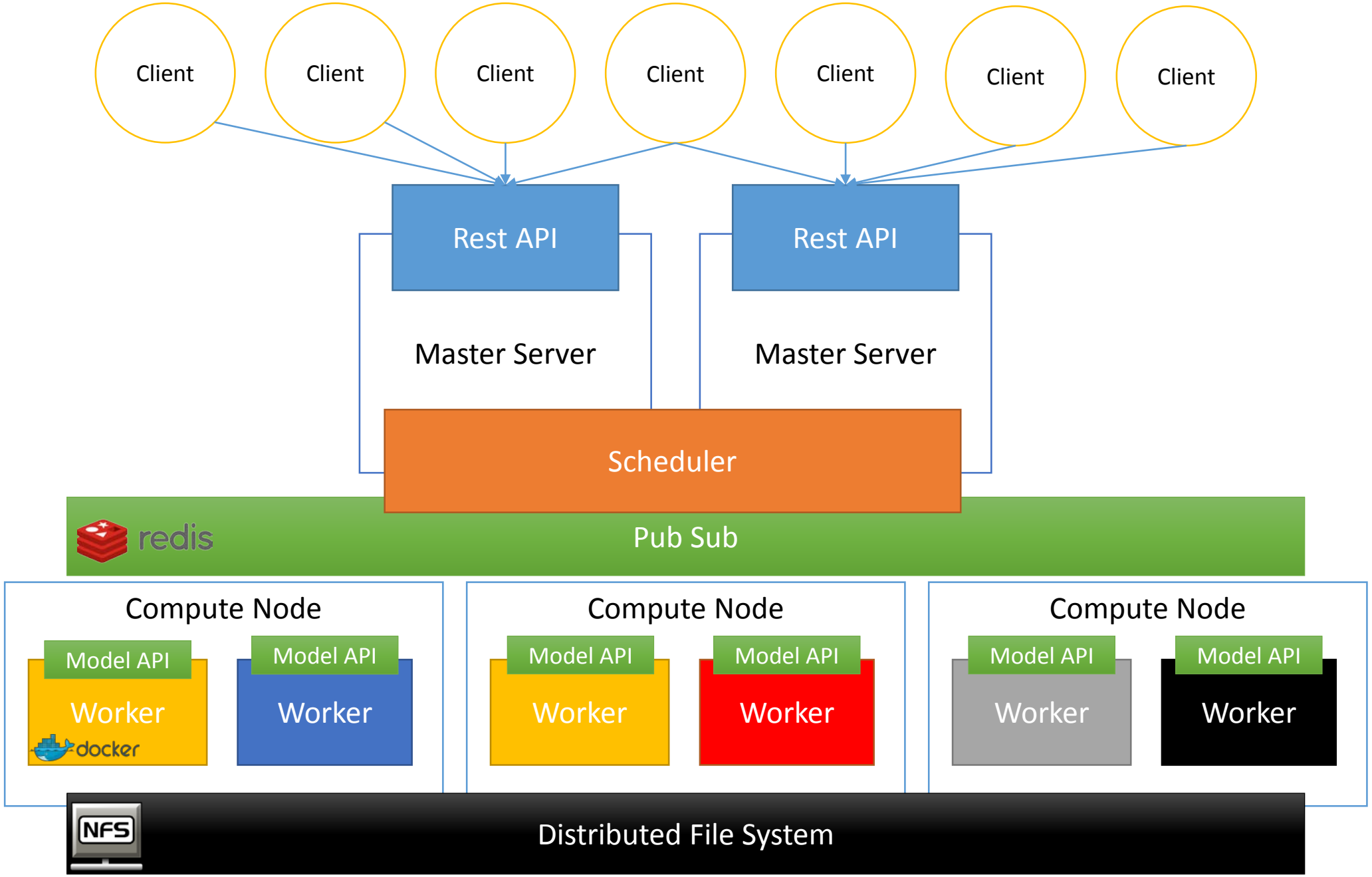
Use case: Registering a new model

1. Model should use our lightweight API
2. Host a model in a container (docker)
3. Benchmark the model (load/runtime) for different batches
4. Reject a model if impossible to server (with respect to SLO)
5. Stored the model on distributed file system



Use case: Submitting a query

1. Client asynchronously sends request(s) to a Master Server and gets request ID(s) as response
2. Master may respond from the cache
3. Otherwise passes the query to scheduler
4. Scheduler assigns each query to a worker, and sends the commands to the pub/sub
    - May unload a loaded model
    - May load a new model
    - May duplicate a running model
    - May wait for more requests to come (for batching)
5. Compute nodes follow the commands to load/unload models off the distributed file system
6. Workers fetch all the requests, server them in batch, and puts the results back in pub/sub
7. Master fetches the results and waits for client to request a responses


Client
Client
Client
Client
Client
Client
Client
Rest API
Rest API
Master Server
Master Server
Scheduler
redis
Pub Sub
Compute Node
Model API
Worker
docker
Model API
Worker
Compute Node
Model API
Worker
Model API
Worker
Compute Node
Model API
Worker
Model API
Worker
NFS
Distributed File System

Client API

- `Send_Requests(data[], model)`
- `Get_Reponses(request_id[])`

- `Start_Session(model)`
- `Send_Requests(data[], session_id)`
- `End_Session(session_id)`

Demo

In progress / Open Problems

- Scheduler
  - Elasticity
  - Analytical model
- API Expansion
  - More languages: Currently Python
  - Pipelines
  - Ensembles
- Model Efficiency
  - Load/Unload
  - Stateful Models
- Model Isolation
  - Currently limited to computation, Outsourced to Docker
  - Memory Bandwidth, PCIE
- Fault Tolerance
  - Currently outsourced to Redis
  - Approximated response

Related Problems

- Model Compression
- DNN specific hardware
- Realtime on mobile devices