Cryptography for

Privacy-Preserving Machine Learning

Morten Dahl

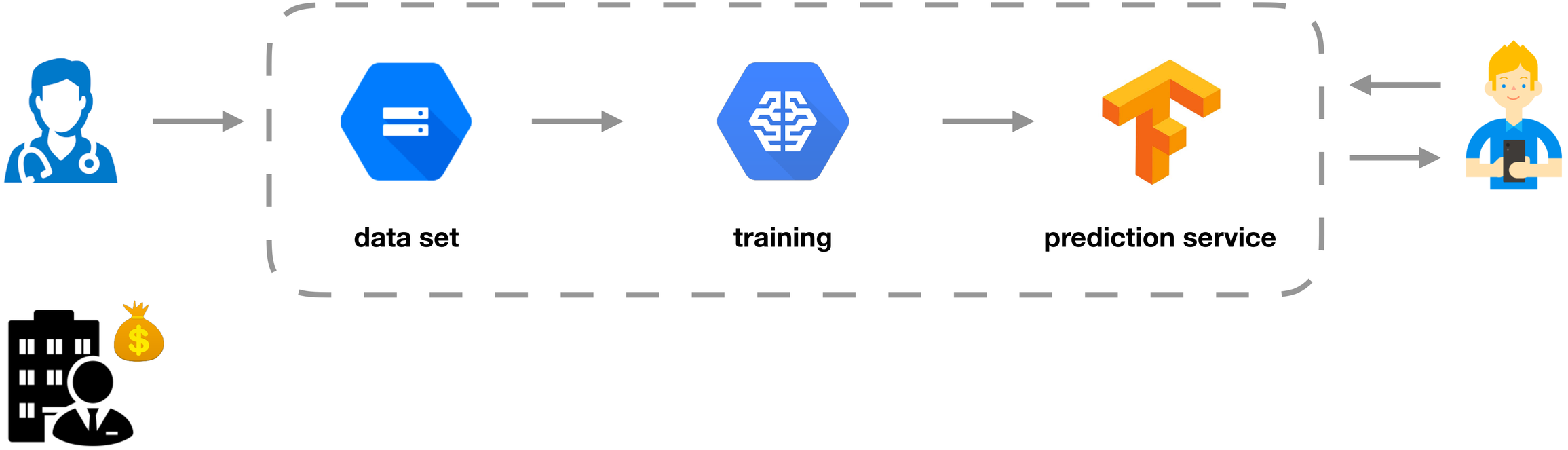
The tf-encrypted Project and Dropout Labs

Hacking Deep Learning, Bar-Ilan University, January 2019

Why?

Machine Learning Process

IMAGENET



Skin Cancer Image Classification

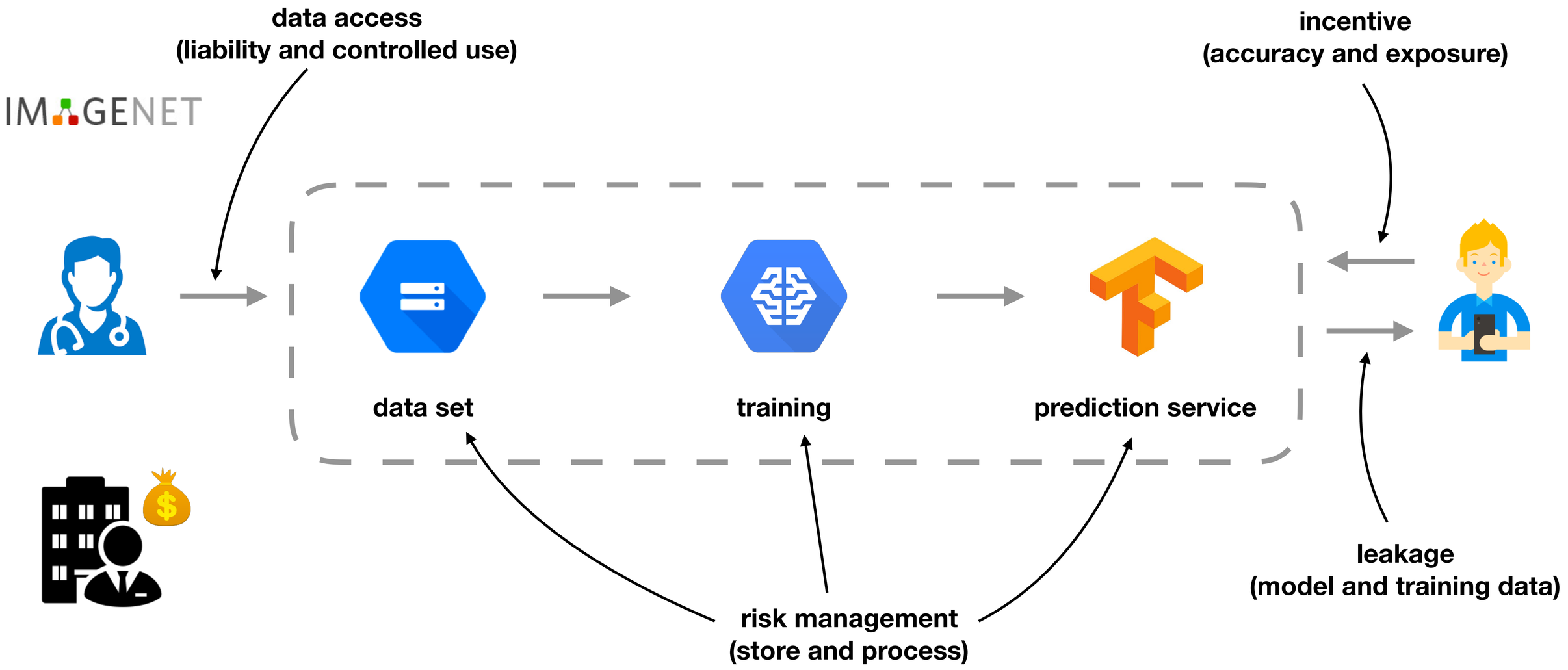
Brett Kuprel

12:30-12:40pm

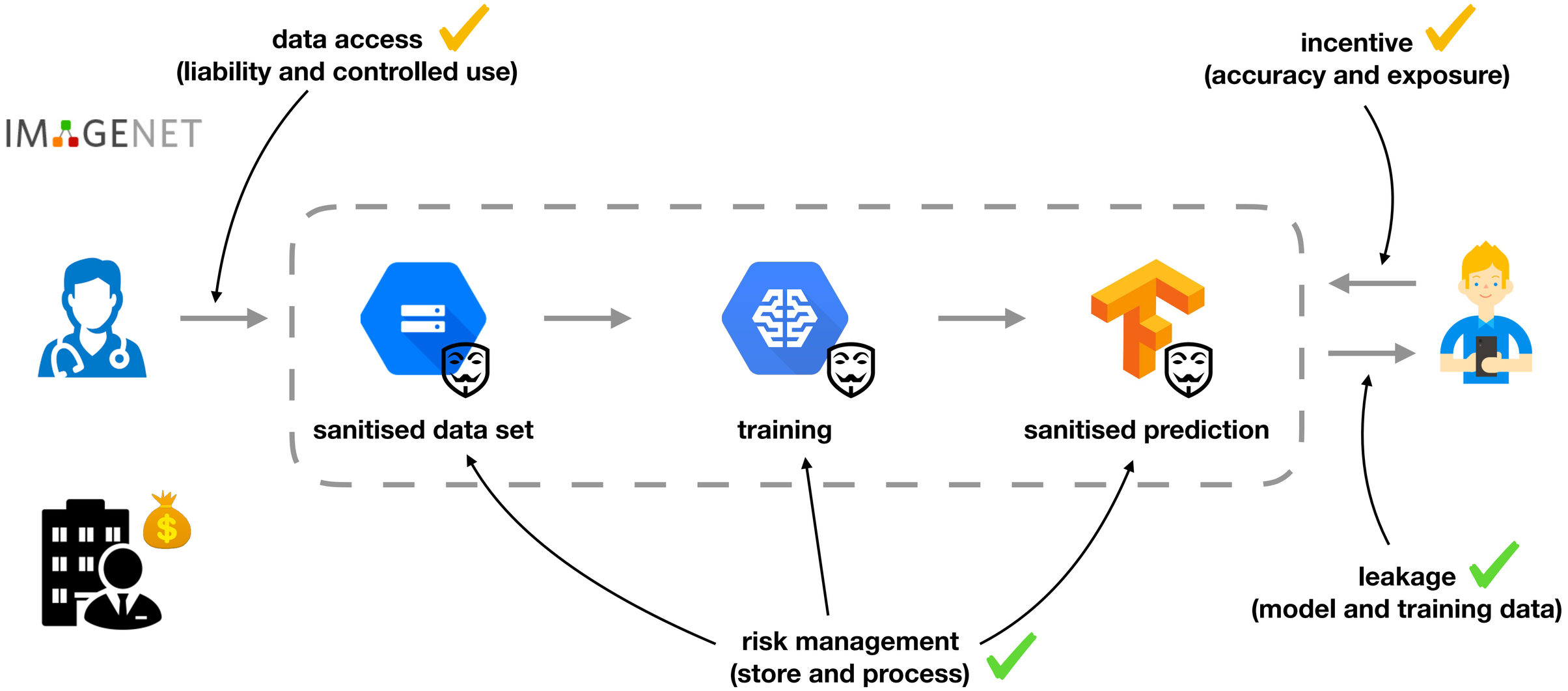
Join Brett Kuprel, and see how TensorFlow was used by the artificial intelligence lab and medical school of Stanford to classify skin cancer images. He'll describe the project steps: from acquiring a dataset, training a deep network, and evaluating of the results. To wrap up, Brett will give his take on the future of skin cancer image classification.

sensitive information is key for future machine learning systems

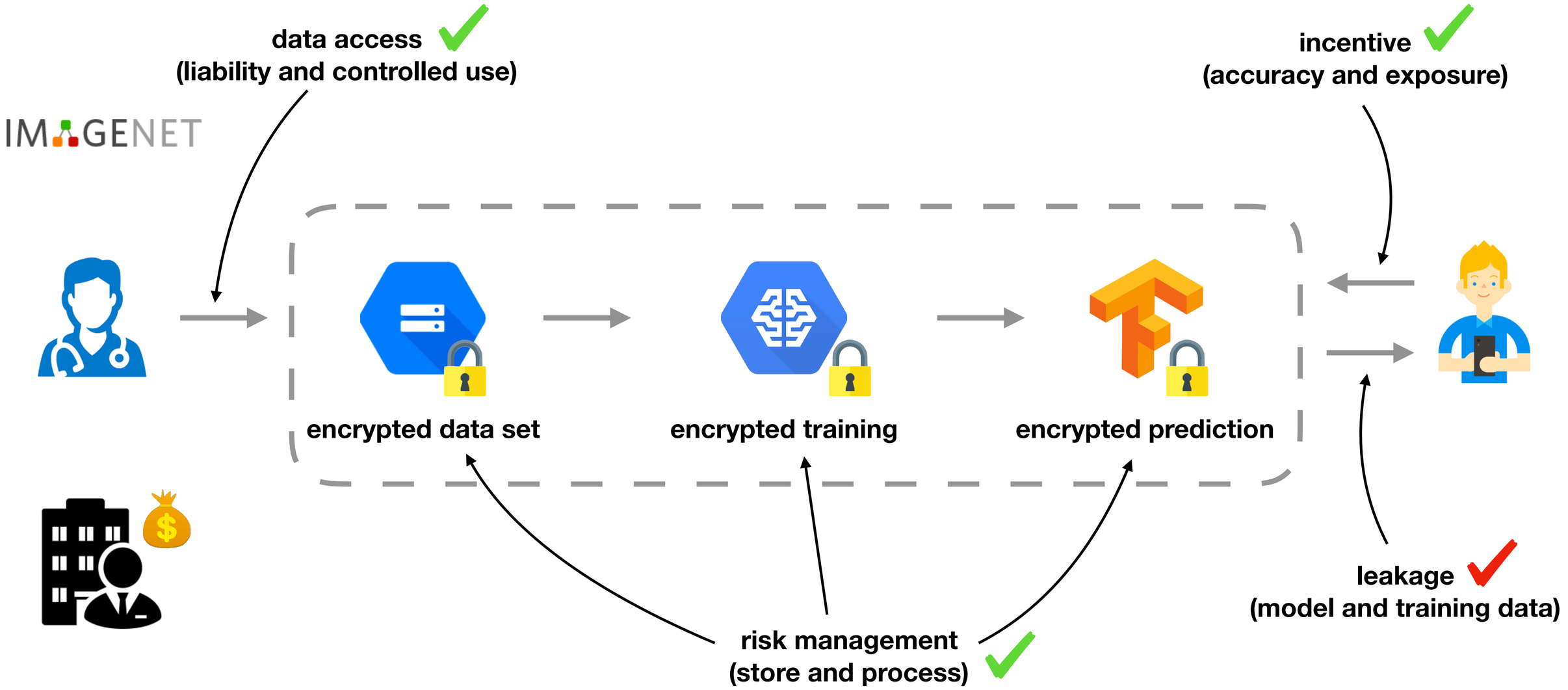
Potential Bottlenecks



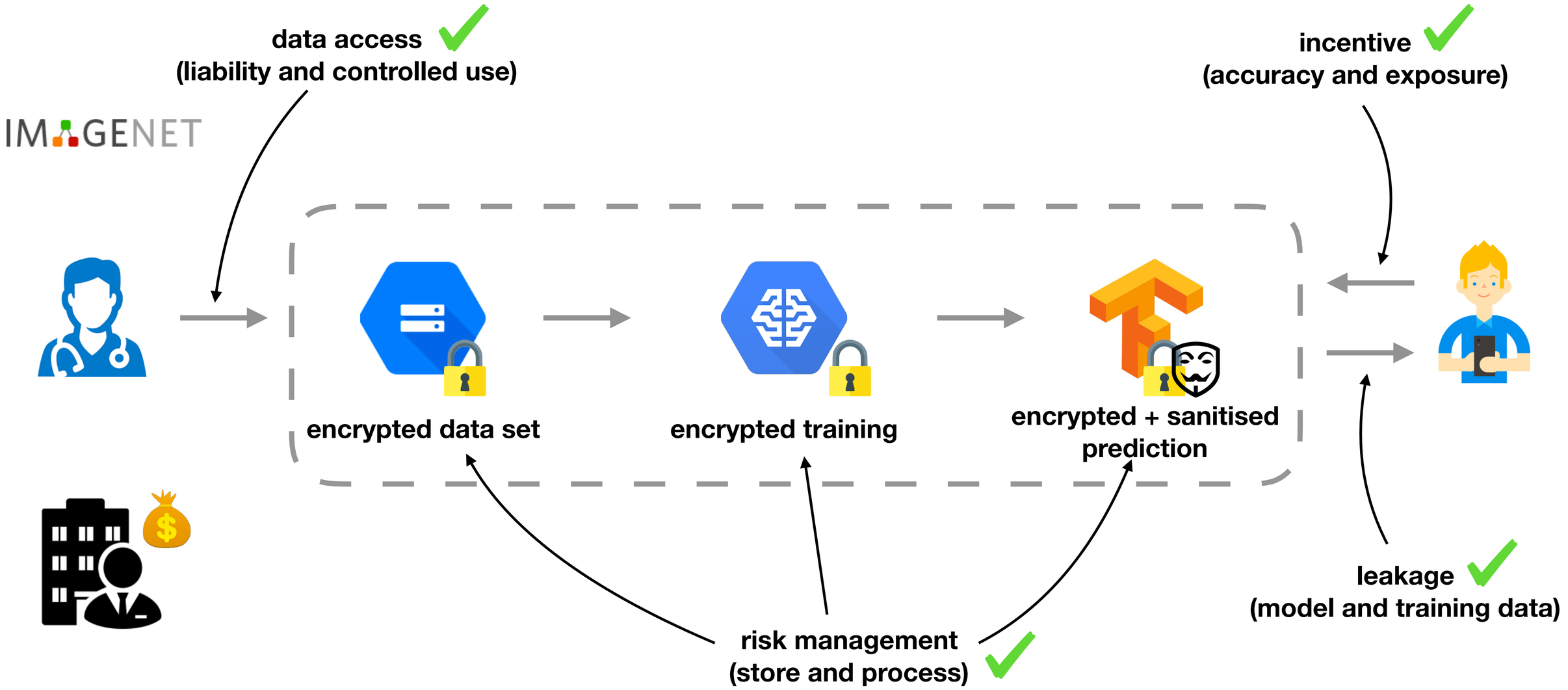
Differential Privacy



Secure Computation



Hybrid


data access
(liability and controlled use)
IMAGENET
incentive
(accuracy and exposure)
encrypted data set
encrypted training
encrypted + sanitised
prediction
leakage
(model and training data)
risk management
(store and process)

Secure Computation

fixed-point encoding

$$\lfloor \pi * 10^4 \rfloor = 31415$$

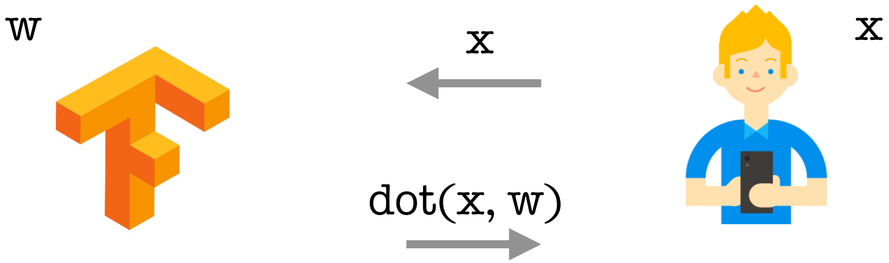
	Homomorphic Encryption	Secret Sharing	Garbled Circuits	Secure Enclaves
Computation	*heavy*	*light*	*medium*	*light*
Communication	*light*	*heavy*	*medium*	*light*
Data types	*integers + bits*	*integers + bits*	*bits*	*any*
Platform	*software*	*software*	*software*	*hardware*

CryptoNets'16	✓			
SecureML'17		✓	✓	
DeepSecure'17			✓	
Gazelle'18	✓		✓	
ABY3'18		✓	✓	
SecureNN'18		✓		
Slalom'18				✓

tendency to mix techniques

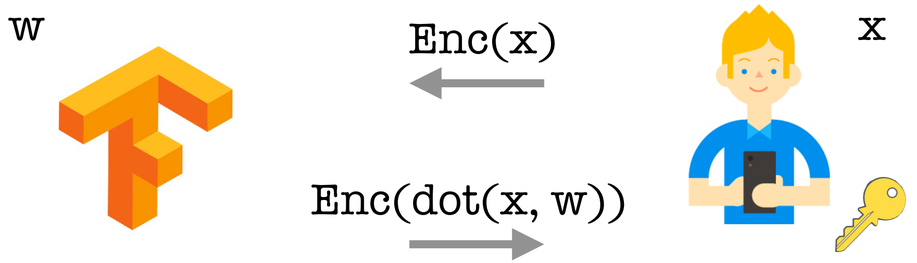
Linear Regression

Prediction with Linear Regression Model



dot(x1 x2 x3 , w1 w2 w3) = x1*w1 + x2*w2 + x3*w3

Using Homomorphic Encryption



homomorphic encryption scheme

public multiplication

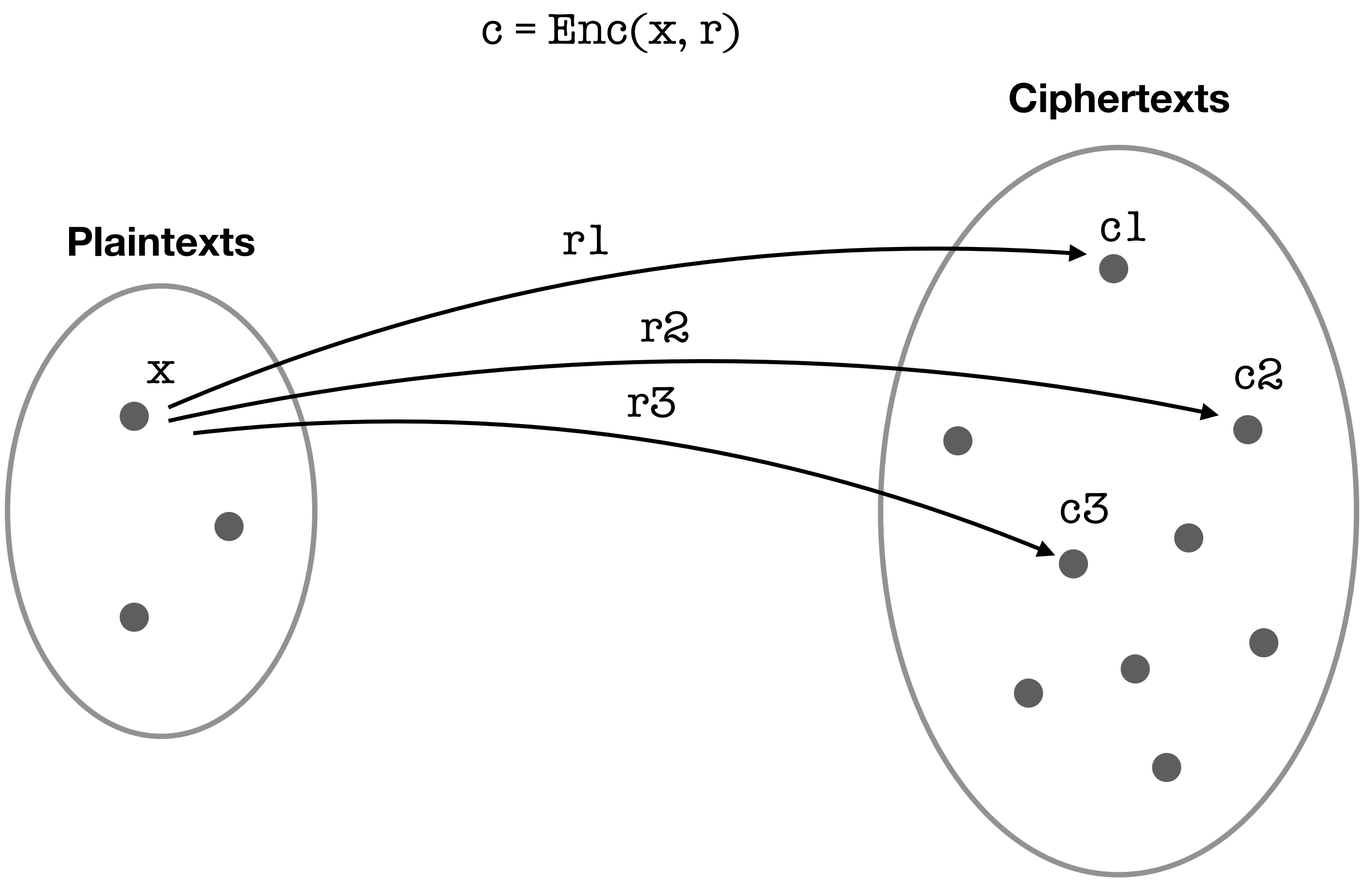
private addition

dot(Enc(x1) Enc(x2) Enc(x3) , w1 w2 w3) = Enc(x1)*w1 + Enc(x2)*w2 + Enc(x3)*w3

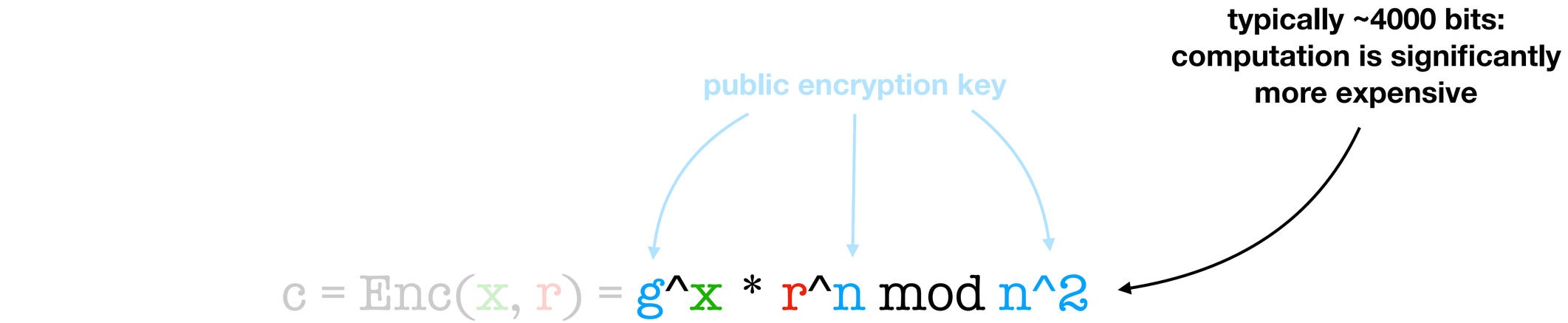
= Enc(x1*w1) + Enc(x2*w2) + Enc(x3*w3)

= Enc(x1*w1 + x2*w2 + x3*w3)

Probabilistic Encryption



Paillier Homomorphic Encryption



g = 36
n = 35
n^2 = 1225

Enc(5, 2) = 36^5 * 2^35 mod 1225 = 718

Enc(5, 4) = 36^5 * 4^35 mod 1225 = 674

Private Addition in Paillier

$$Enc(x, r) * Enc(y, s)$$

$$= (g^x * r^n \bmod n^2) * (g^y * s^n \bmod n^2)$$

$$= g^{(x+y)} * (r * s)^n \bmod n^2$$

$$= Enc(x+y, r*s)$$

$$Enc(5, 2) * Enc(5, 4)$$
$$= 718 * 674$$
$$= 57$$
$$= 36^{10} * 8^{35}$$
$$= Enc(10, 8)$$

Public Multiplication in Paillier

$$\text{Enc}(x, r) \hat{} w$$

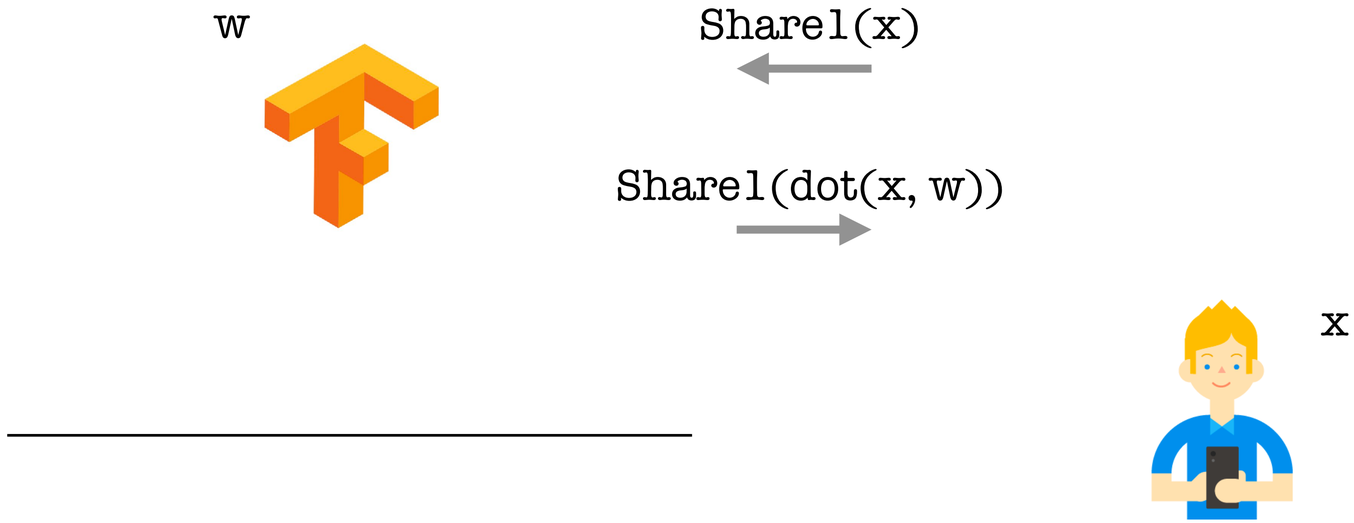
$$= (g \hat{} x * r \hat{} n \bmod n \hat{} 2) \hat{} w$$

$$= g \hat{} (x * w) * (r \hat{} w) \hat{} n \bmod n \hat{} 2$$

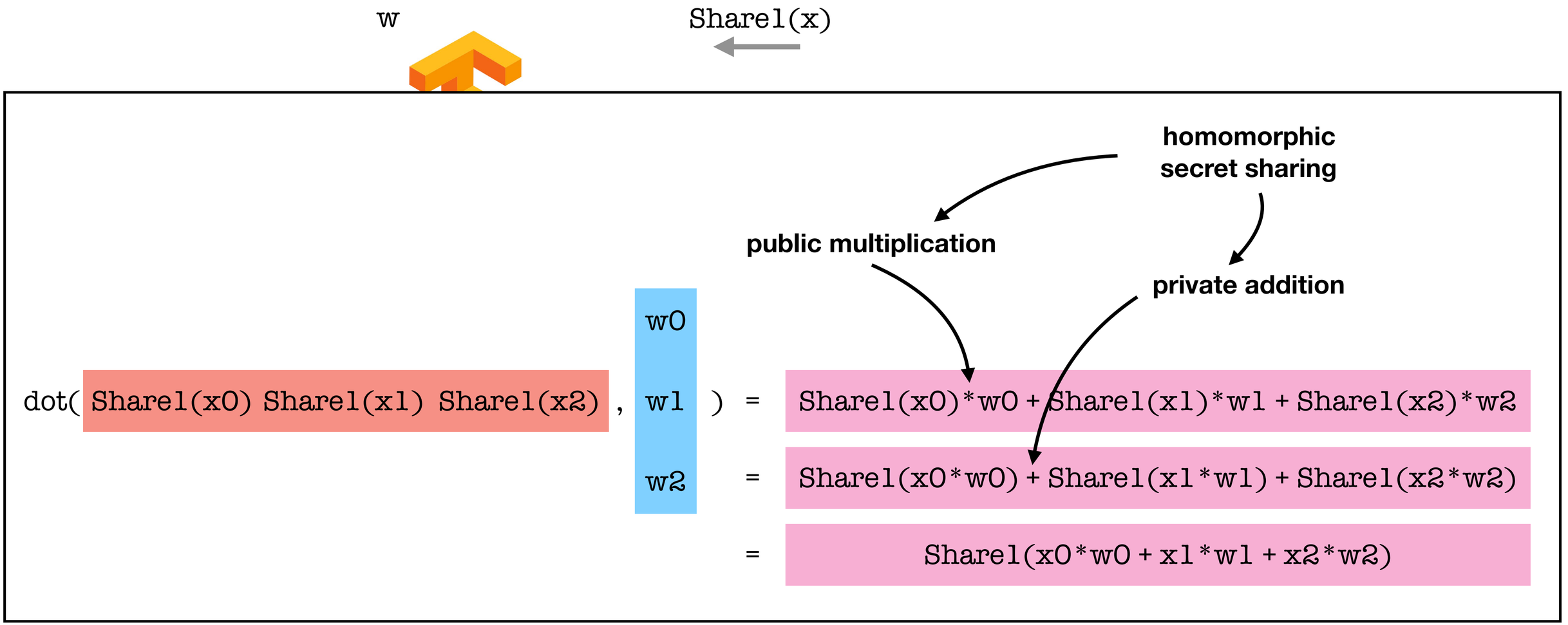
$$= \text{Enc}(x * w, r \hat{} w)$$

$$\text{Enc}(5, 2) \hat{} 2$$
$$= 718 * 718$$
$$= 1024$$
$$= 36 \hat{} 10 * 4 \hat{} 35$$
$$= \text{Enc}(10, 4)$$

Using Secret Sharing



Using Secret Sharing



SPDZ Secret Sharing

public parameter

$$\text{Share1}(x, r) = r \bmod m$$

$$\text{Share2}(x, r) = x - r \bmod m$$

$$x = \text{Share1}(x, r) + \text{Share2}(x, r) \bmod m$$

$m = 10$

$$\text{Share1}(5, 7) = 7 \bmod 10 = 7$$

$$\text{Share2}(5, 7) = 5 - 7 \bmod 10 = 8$$

$$7 + 8 = 15 = 5 \bmod 10$$

Private Addition in SPDZ

x_1 y_1 $z_1 = x_1 + y_1$

x_2 y_2 $z_2 = x_2 + y_2$

$x = x_1 + x_2$ $y = y_1 + y_2$

$$
\begin{aligned}
x + y &= (x_1 + x_2) + (y_1 + y_2) \\
&= (x_1 + y_1) + (x_2 + y_2) \\
&= z_1 + z_2
\end{aligned}
$$

Public Multiplication in SPDZ

`x1` `w` `z1 = x1 * w`

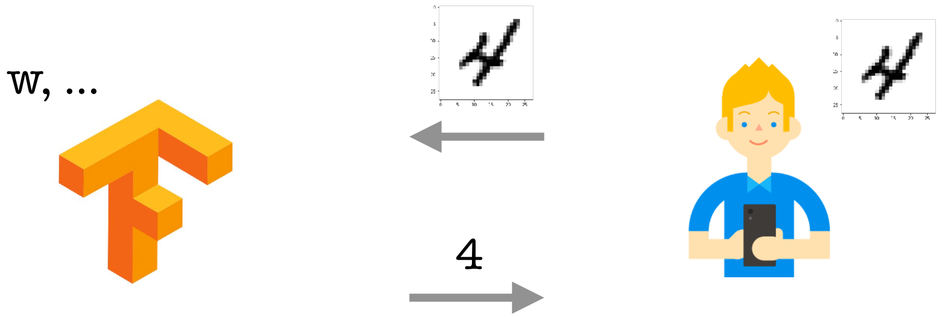
`x2` `w` `z2 = x2 * w`

`x = x1 + x2`

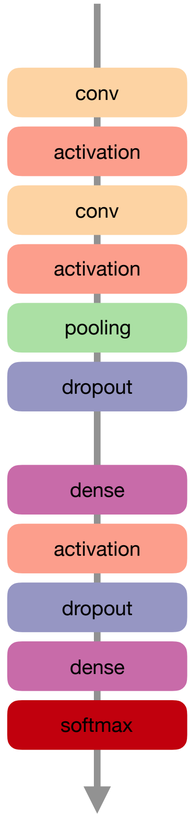
```
x * w
  = (x1 + x2) * w
  = (x1 * w) + (x2 * w)
  = z1 + z2
```

Convolutional Neural Networks

Digit Classification with CNNs

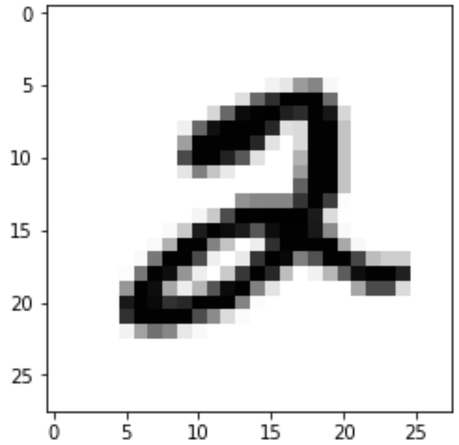



```
feature_layers = [
    Conv2D(32, (3, 3), padding='same', input_shape=(28, 28, 1)),
    Activation('relu'),
    Conv2D(32, (3, 3), padding='same'),
    Activation('relu'),
    MaxPooling2D(pool_size=(2,2)),
    Dropout(.25),
    Flatten()
]

classification_layers = [
    Dense(128),
    Activation('relu'),
    Dropout(.50),
    Dense(NUM_CLASSES),
    Activation('softmax')
]

model = Sequential(feature_layers + classification_layers)
```



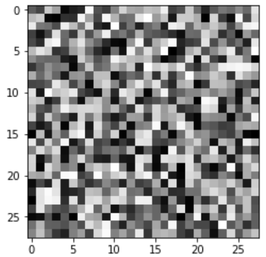
Secret Sharing Images

Secret Sharing Images

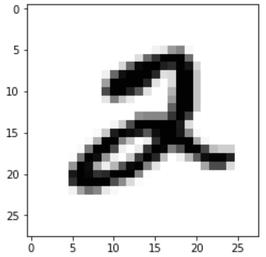


```
array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0, 143, 247, 153,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0, 136, 247, 242,  86,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0, 192, 252, 187,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        185,  18,   0,   0,   0,   0,   0,  89, 236, 217,  47,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        253,  60,   0,   0,   0,   0, 212, 255,  81,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        252,  68,   0,   0,   0,  48, 242, 253,  89,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        212,  21,   0,   0,  11, 167, 252, 197,   5,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  29,
         63,   0,   0,   0, 153, 252, 226,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  45, 219, 252,
          0,   0,   0, 116, 249, 252, 103,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   4,  96, 253, 255, 253,
        122,   7,  25, 201, 250, 158,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0]], dtype=uint8)
```

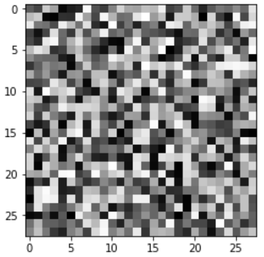
Share1(x, r) = r =



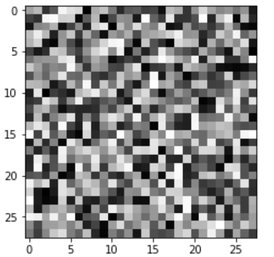
Share2(x, r) = x - r =



-

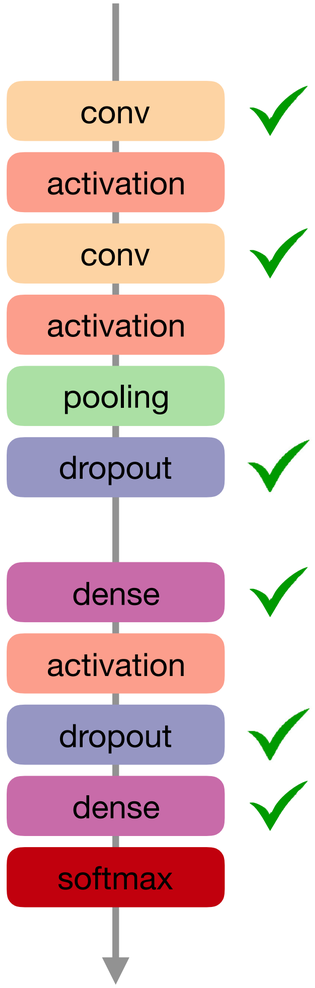


=



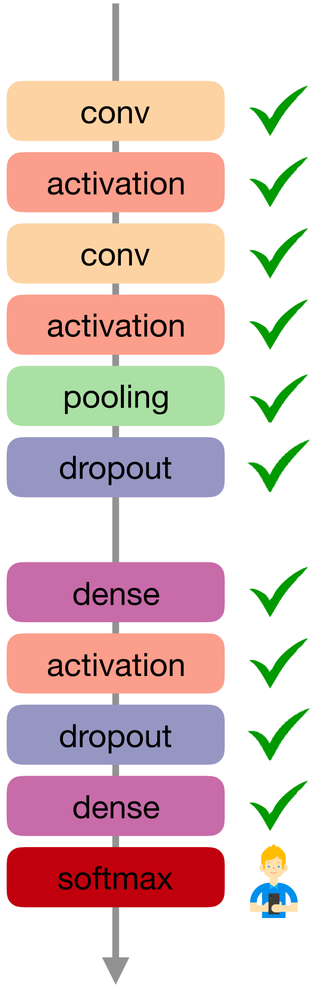
Digit Classification with Secret Sharing

```
feature_layers = [
    Conv2D(32, (3, 3), padding='same', input_shape=(28, 28, 1)),
    Activation('relu'),
    Conv2D(32, (3, 3), padding='same'),
    Activation('relu'),
    MaxPooling2D(pool_size=(2,2)),
    Dropout(.25),
    Flatten()
]

classification_layers = [
    Dense(128),
    Activation('relu'),
    Dropout(.50),
    Dense(NUM_CLASSES),
    Activation('softmax')
]

model = Sequential(feature_layers + classification_layers)
```



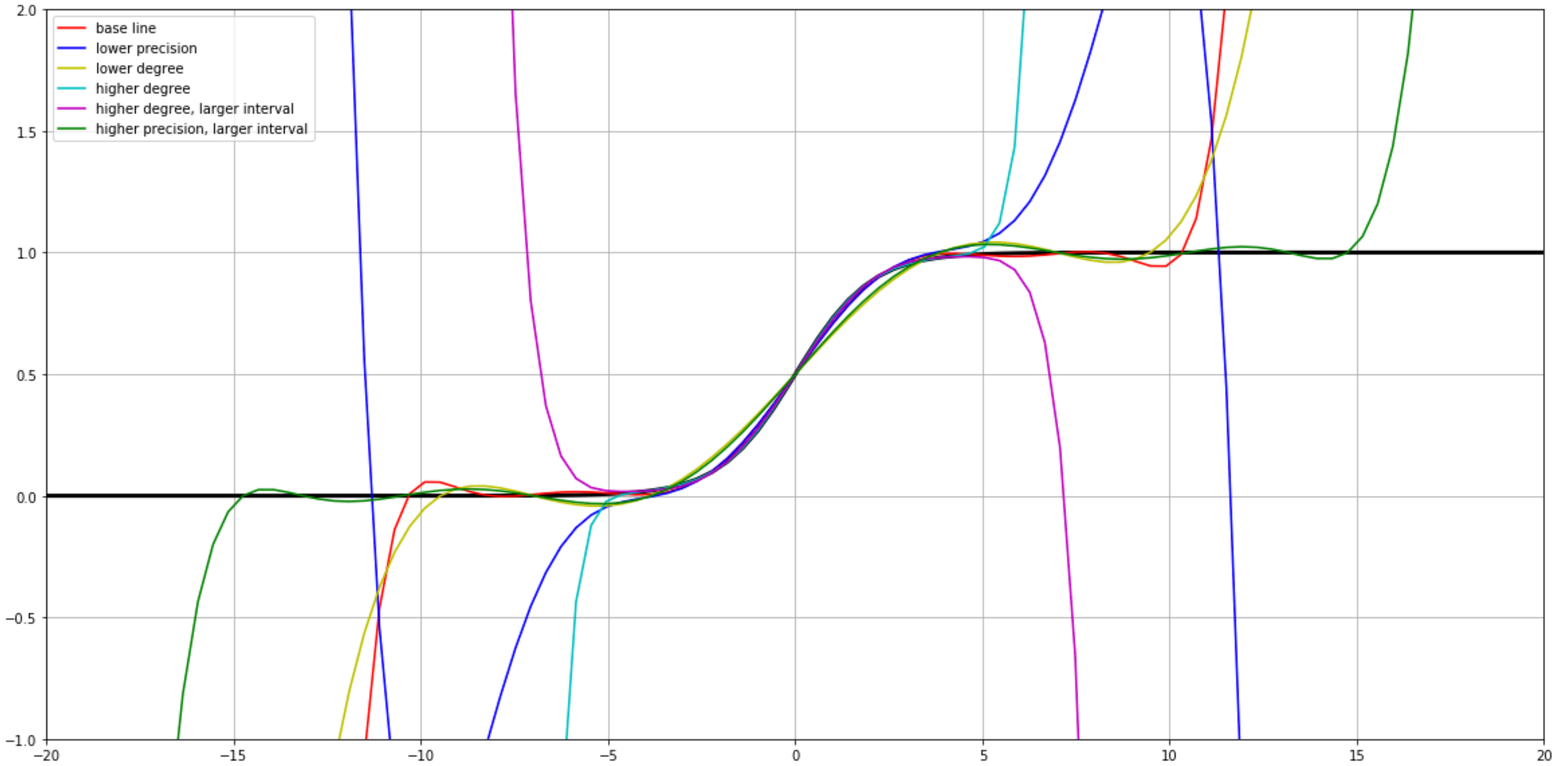
Digit Classification with Secret Sharing

```
feature_layers = [
   feature_layers = [
       Conv2D(32, (3, 3), padding='same', input_shape=(28, 28, 1)),
       Activation('sigmoid'),
       Conv2D(32, (3, 3), padding='same'),
       Activation('sigmoid'),
       AveragePooling2D(pool_size=(2,2)),
       Dropout(.25),
]      Flatten()
   ]
cl
   classification_layers = [
       Dense(128),
       Activation('sigmoid'),
       Dropout(.50),
       Dense(5),
]      Activation('softmax')
   ]
mo
   model = Sequential(feature_layers + classification_layers)
```



model adaptation can improve performance significantly

Sigmoid via Polynomial Approximation



$$f(x) = c7*x^7 + c5*x^5 + \ldots + c1*x + c0$$

Private Multiplication in SPDZ

random triple

(a1 , b1 , c1)

x1

y1

alpha

beta

z1 = alpha*beta
+ alpha*b1
+ beta*a1
+ c1

(a2 , b2 , c2)

x2

y2

alpha

beta

z2 = alpha*b2
+ beta*a2
+ c2

a = a1 + a2

b = b1 + b2

c = a*b = c1 + c2

x = x1 + x2

y = y1 + y2

alpha = x - a

beta = y - b

x * y
= ...
= z1 + z2

Training

Participants

Share1(x, y)

Share2(x, y)

SoftMax Alternatives

$$f(x_i) = \frac{e^{-x_i}}{\sum_j e^{-x_j}}$$

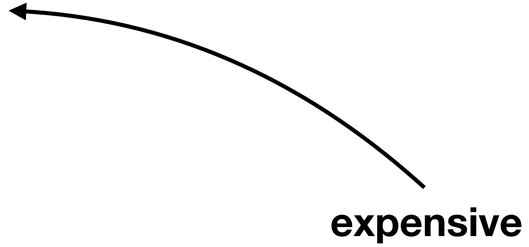
$$g(x_i) = \frac{\mathsf{ReLU}(x_i)}{\sum_j \mathsf{ReLU}(x_j)}$$

SecureML: A System for Scalable Privacy-Preserving Machine Learning, Mohassel and Zhang ‘16

Simpler Optimisation Strategies

RMSProp:

$$w = w - \frac{\eta}{\sqrt{v(w,t)}} \nabla Q(w)$$

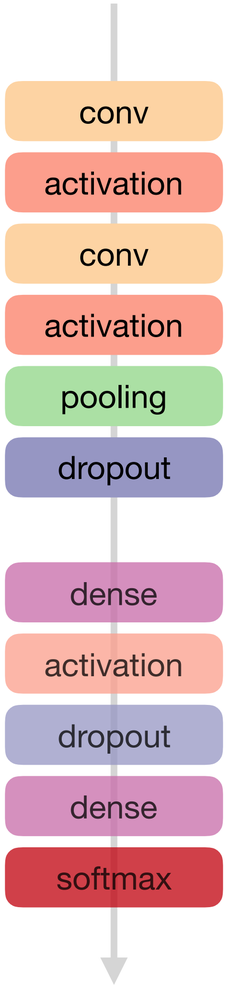


SGD:

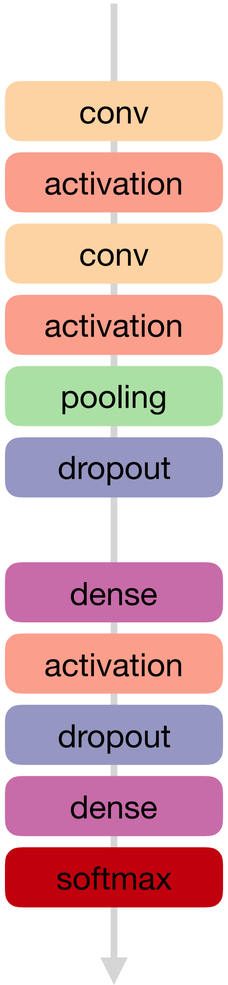
$$w = w - \eta \nabla Q(w)$$

Transfer Learning

IMAGENET



Transfer Learning



Making It Accessible

Projects and Literature

Recent research papers using secure computation

***CryptoNets**: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy*, Dowlin et al.

***SecureML**: A System for Scalable Privacy-Preserving Machine Learning*, Mohassel and Zhang

***DeepSecure**: Scalable Provably-Secure Deep Learning*, Rouhani et al.

***Gazelle**: A Low Latency Framework for Secure Neural Network Inference*, Juvekar et al.

***ABY3**: A Mixed Protocol Framework for Machine Learning*, Mohassel and Rindal

***SecureNN**: Efficient and Private Neural Network Training*, Wagh et al.

(great summary in https://eprint.iacr.org/2017/1190)

Specialised projects

tf-encrypted *(https://github.com/mortendahl/tf-encrypted)*

PySyft *(https://github.com/OpenMined/PySyft)*

Secure computation frameworks

SCALE-MAMBA *(https://homes.esat.kuleuven.be/~nsmart/SCALE/)*

ABY *(https://github.com/encryptogroup/ABY)*

OblivC *(http://oblivc.org/)*

(much more at https://github.com/rdragos/awesome-mpc)

Multidisciplinary Challenge

Data science
(use-cases, workflow, monitoring)

Cryptography
(techniques, protocols, guarantees)

Machine learning
(models, approx, quantisation)

Engineering
(distributed, multi-core, readability)

useful to have common framework

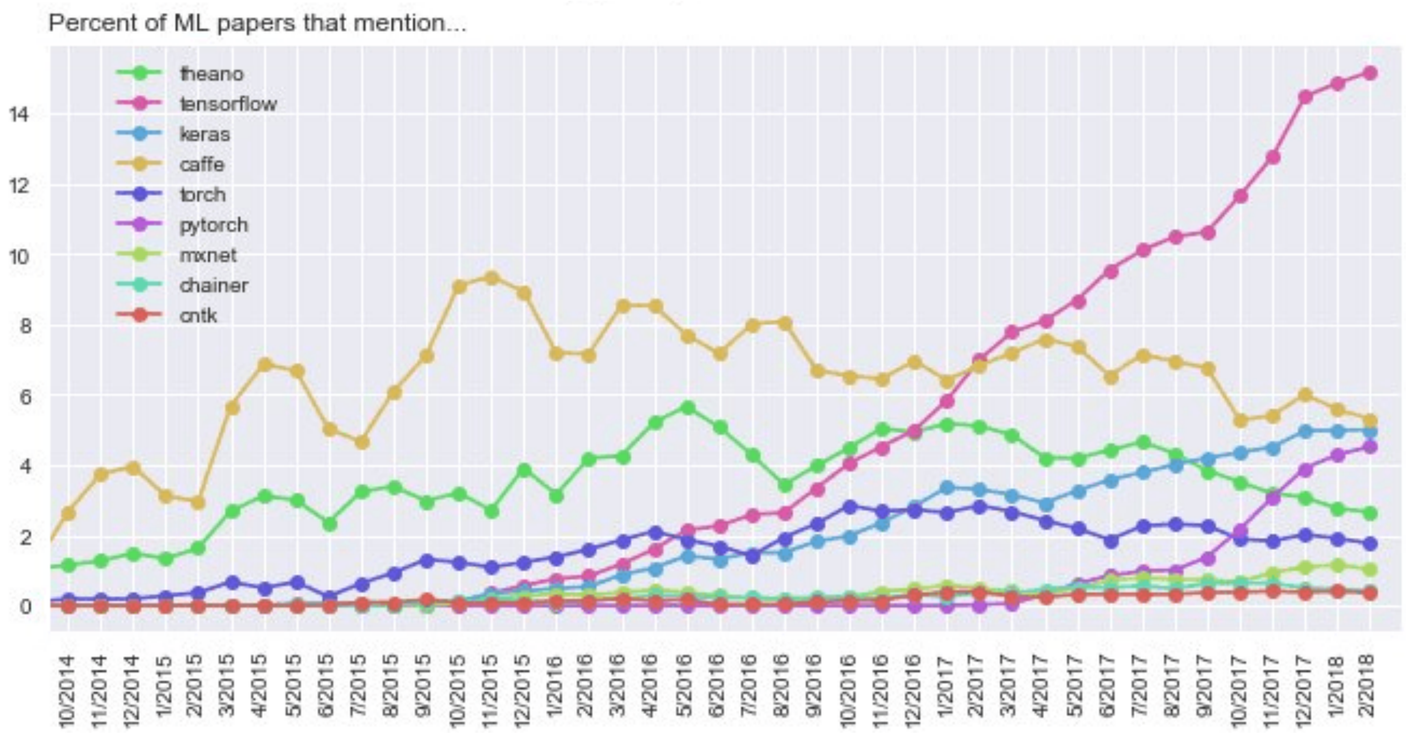
tf-encrypted

open source project for exploring and experimenting with
privacy-preserving machine learning in TensorFlow

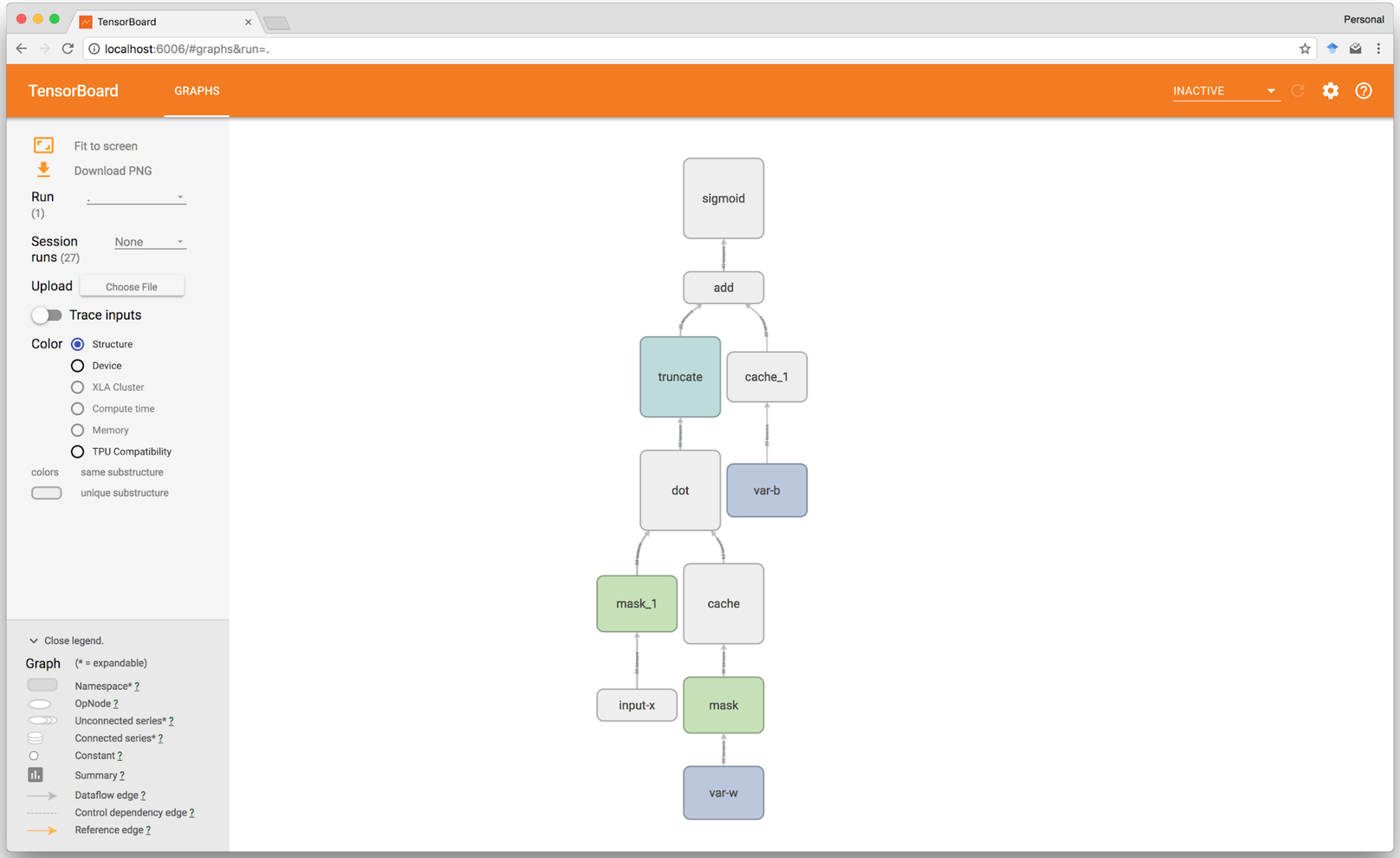
TensorFlow

platform for production-level training and deployment of models

popular and backed by Google



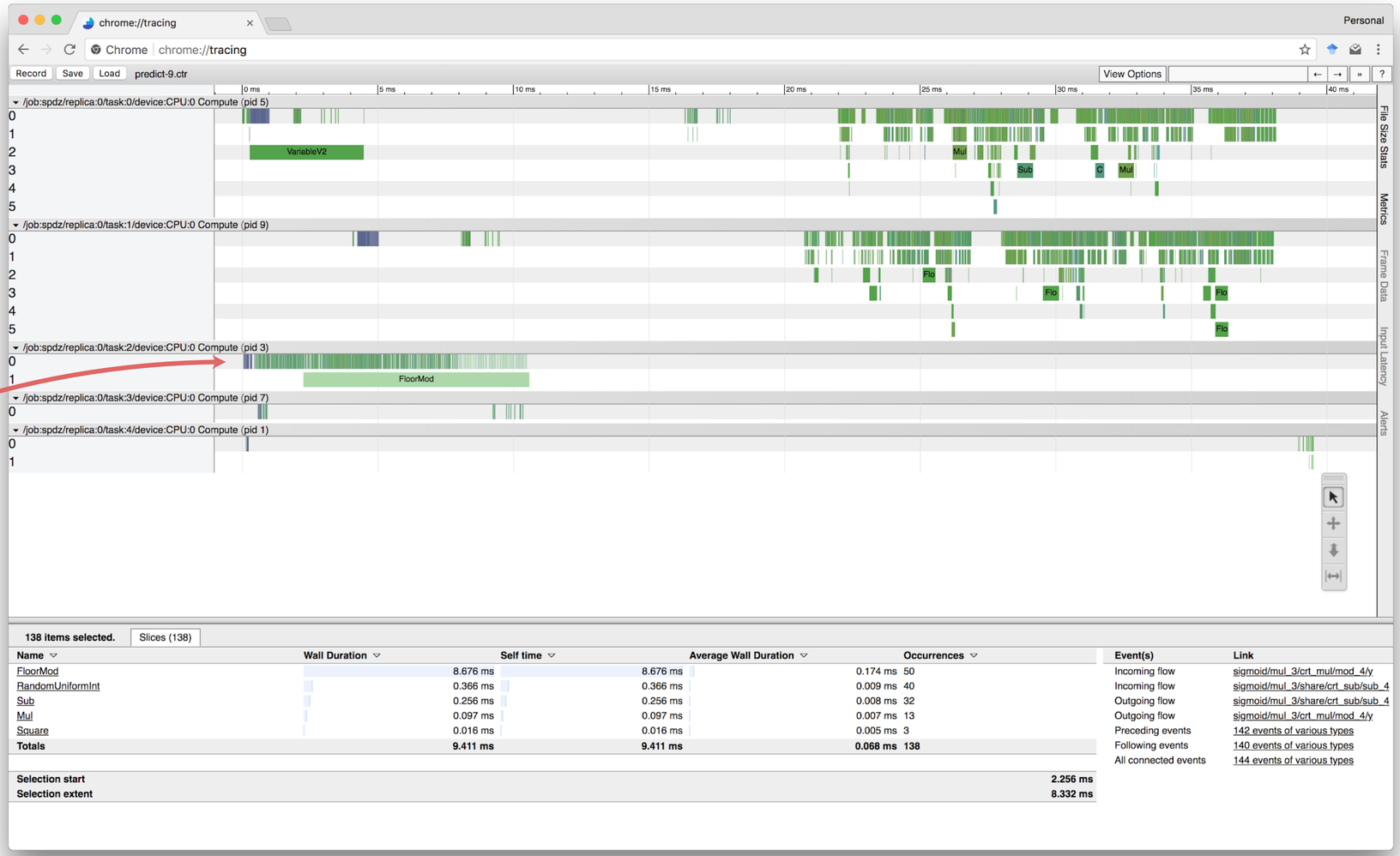
Dataflow Graphs



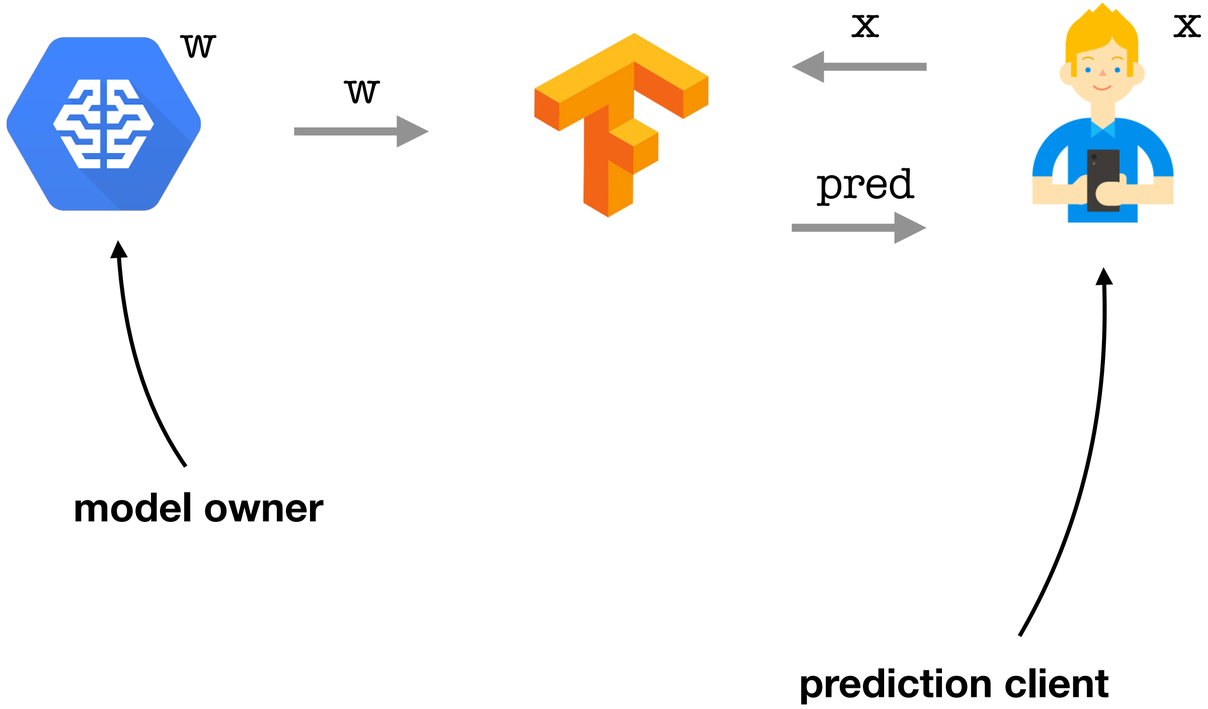
Optimized Distributed Execution

Concurrent

Batching

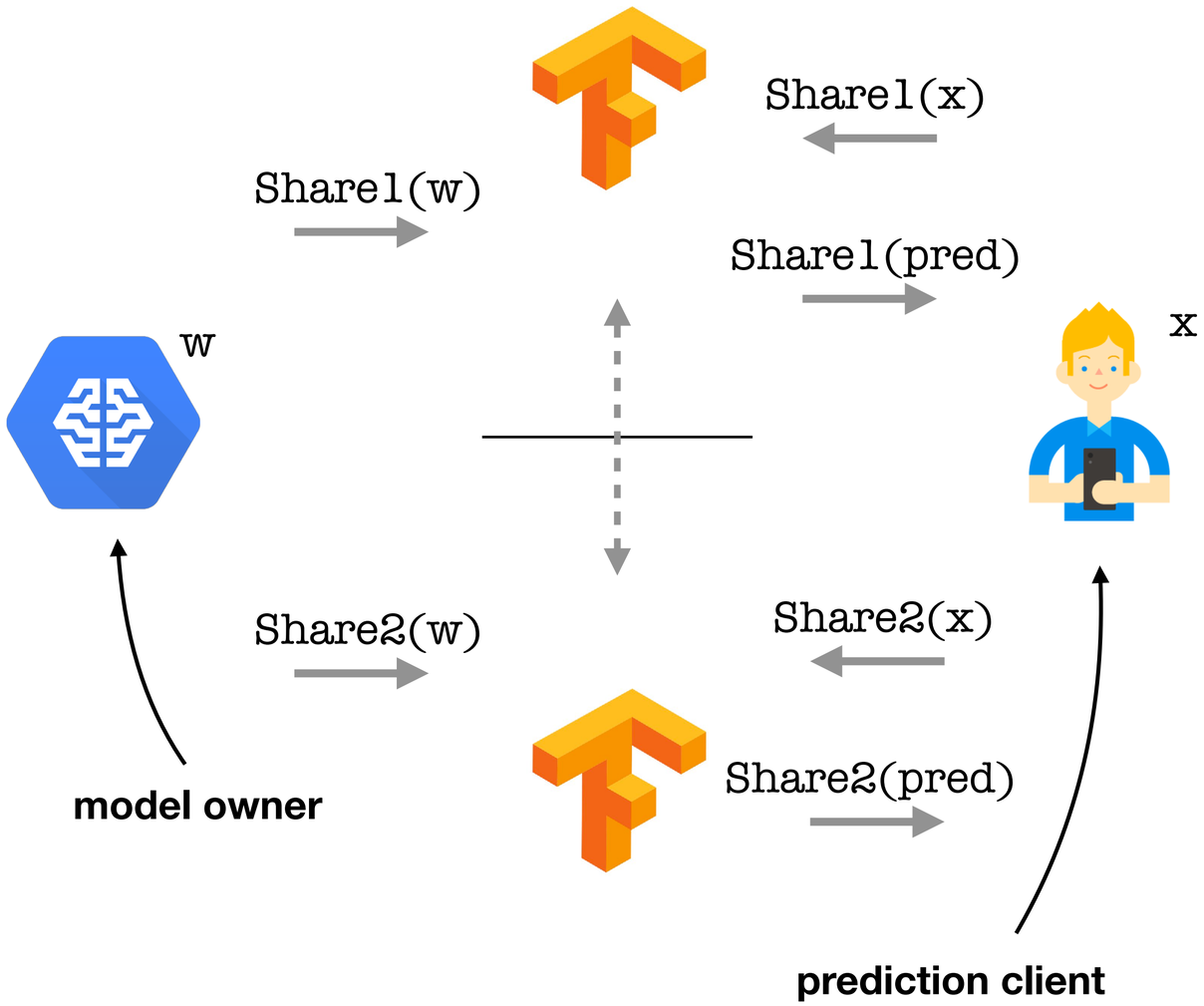


Public Prediction Example (TensorFlow)




```python
import tensorflow as tf

def provide_weights():""" Load from disk """
def provide_input(): """ Pre-process """
def receive_output(logits): return tf.Print([], [tf.argmax(logits)])

# get model weights
w0, b0, w1, b1, w2, b2 = provide_weights()

# get prediction input
x = provide_input()

# compute prediction
layer0 = tf.nn.relu((tf.matmul(x, w0) + b0))
layer1 = tf.nn.relu((tf.matmul(layer0, w1) + b1))
logits = tf.matmul(layer2, w2) + b2

# process result of prediction
prediction_op = receive_output(logits)

# run graph execution in a tf.Session
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    sess.run(prediction_op)
```

Private Prediction Example (tf-encrypted)




```python
import tensorflow as tf
import tf_encrypted as tfe

def provide_weights():""" Load from disk """
def provide_input(): """ Pre-process """
def receive_output(logits): return tf.Print([], [tf.argmax(logits)])

# get model weights as private tensors from owner
w0, b0, w1, b1, w2, b2 = tfe.define_private_input("model-owner", provide_weights)

# get prediction input as private tensors from client
x = tfe.define_private_input("prediction-client", provide_input)

# compute private prediction on servers
layer0 = tfe.relu((tfe.matmul(x, w0) + b0))
layer1 = tfe.relu((tfe.matmul(layer0, w1) + b1))
logits = tfe.matmul(layer1, w2) + b2

# process result of prediction on client
prediction_op = tfe.define_output("prediction-client", logits, receive_output)

# run secure graph execution in a tf.Session
with tfe.Session() as sess:
    sess.run(tf.global_variables_initializer())
    sess.run(prediction_op)
```

Wrap-Up

Privacy-preserving technologies are
enablers for accessing sensitive information

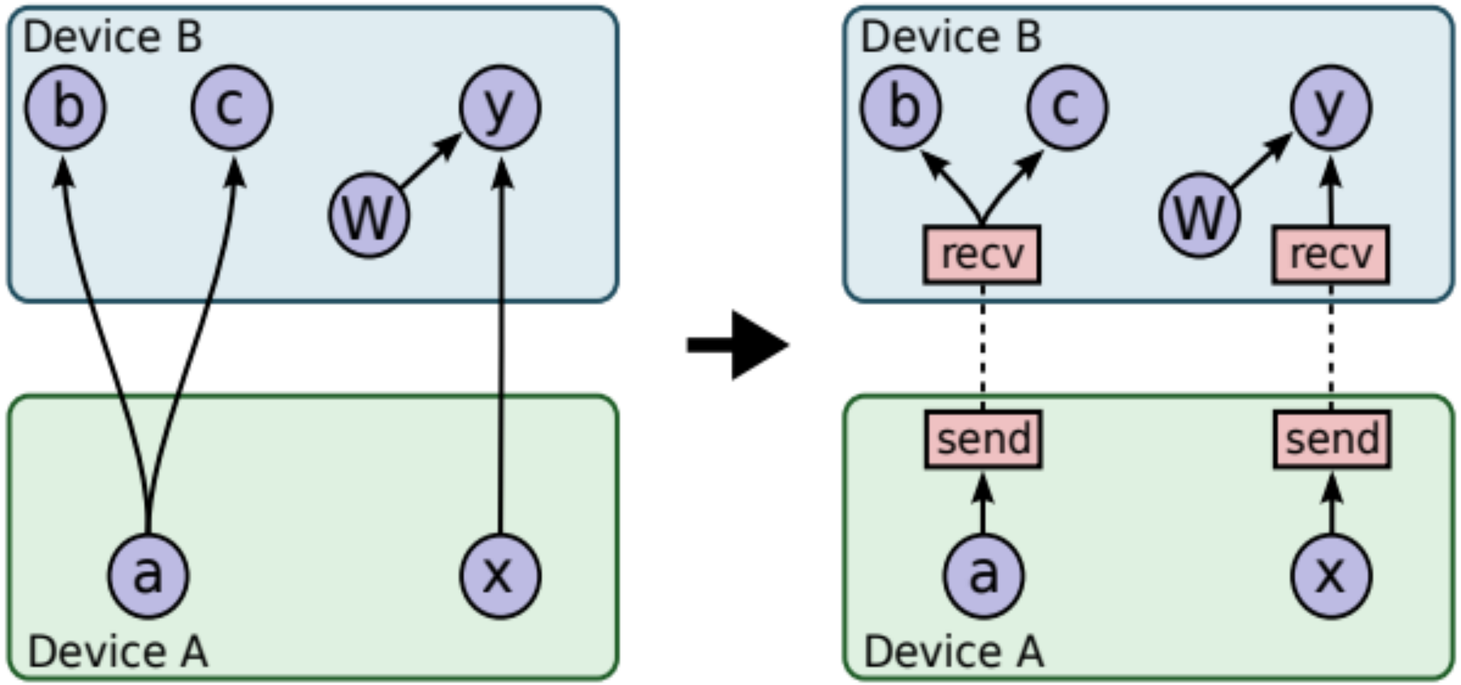
Secure computation is complementary,
not a replacement

Multidisciplinary field benefitting
from adaptations on both sides

Bonus:
some interest from safety (governance)
and fairness (validation)

Thank you!

Distributed TensorFlow



TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems, Abadi et al. ‘16

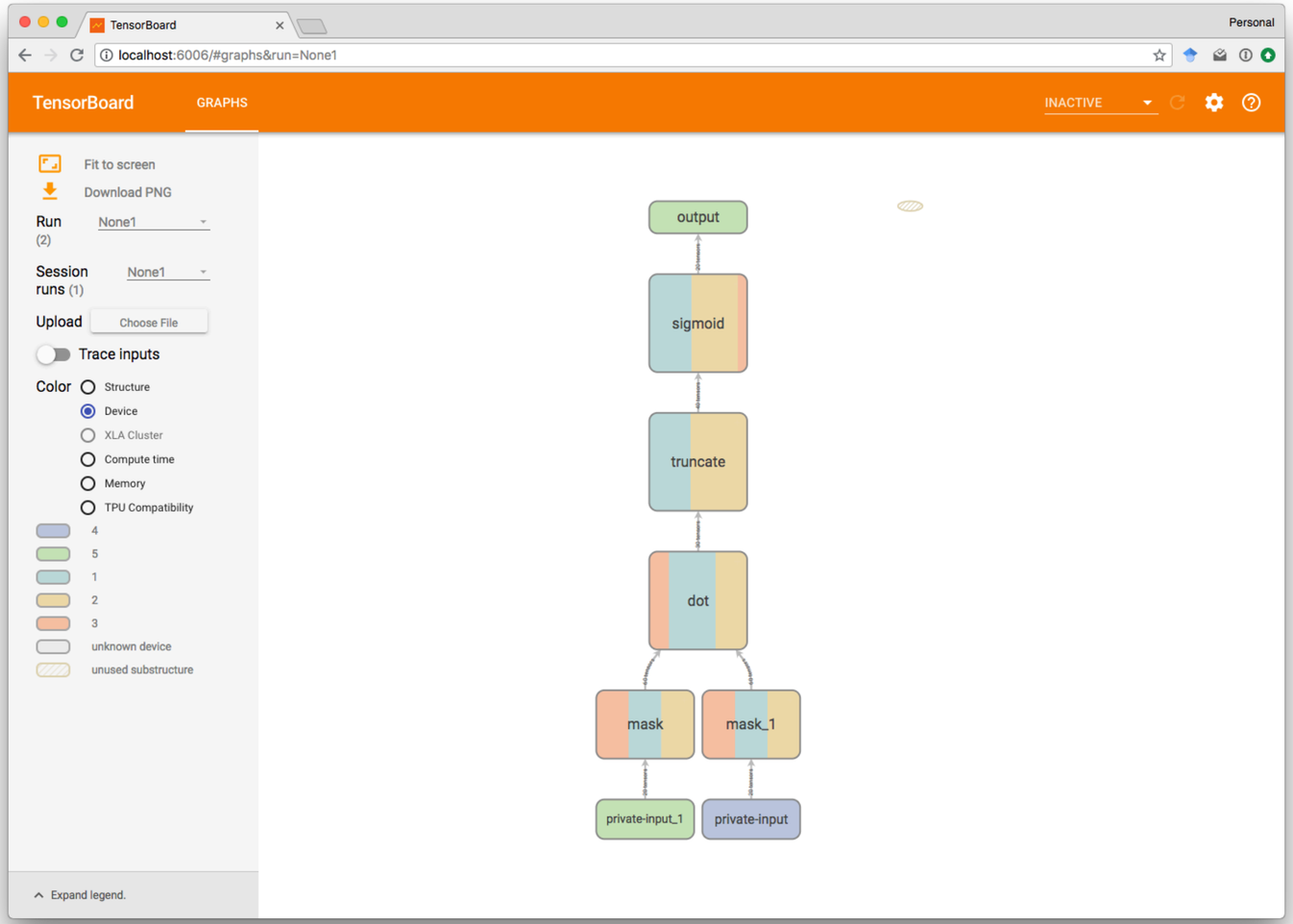
Protocols in TensorFlow

```
1   def _matmul_masked_masked(prot, x, y):
2       a, a0, a1, alpha_on_0, alpha_on_1 = x.unwrapped
3       b, b0, b1, beta_on_0, beta_on_1 = y.unwrapped
4
5       with tf.name_scope('matmul'):
6
7           with tf.device(prot.crypto_producer.device_name):
8               ab = a.matmul(b)
9               ab0, ab1 = prot._share(ab)
10
11          with tf.device(prot.server_0.device_name):
12              z0 = ab0 + a0.matmul(beta) + alpha.matmul(b0) + alpha.matmul(beta)
13
14          with tf.device(prot.server_1.device_name):
15              z1 = ab1 + a1.matmul(beta) + alpha.matmul(b1)
16
17          return PondPrivateTensor(prot, z0, z1)
```

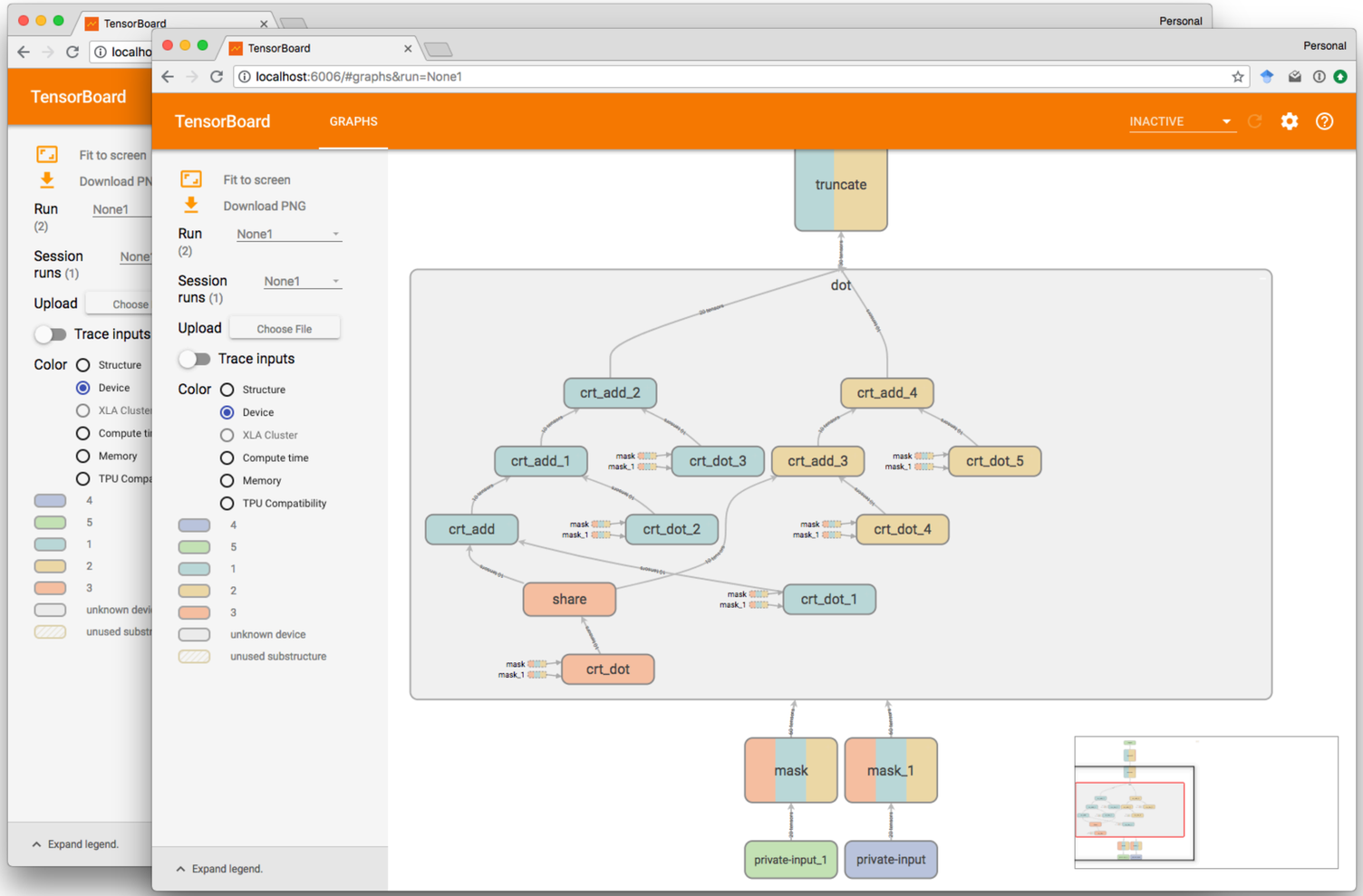
Protocols in TensorFlow

```
1   def _matmul_masked_masked(prot, x, y):
2       a, a0, a1, alpha_on_0, alpha_on_1 = x.unwrapped
3       b, b0, b1, beta_on_0, beta_on_1 = y.unwrapped
4
5       with tf.name_scope('matmul'):
6
7           with tf.device(prot.crypto_producer.device_name):
8               ab = a.matmul(b)
9               ab0, ab1 = prot._share(ab)
10
11          with tf.device(prot.server_0.device_name):
12              z0 = ab0 + a0.matmul(beta) + alpha.matmul(b0) + alpha.matmul(beta)
13
14          with tf.device(prot.server_1.device_name):
15              z1 = ab1 + a1.matmul(beta) + alpha.matmul(b1)
16
17          return PondPrivateTensor(prot, z0, z1)
```

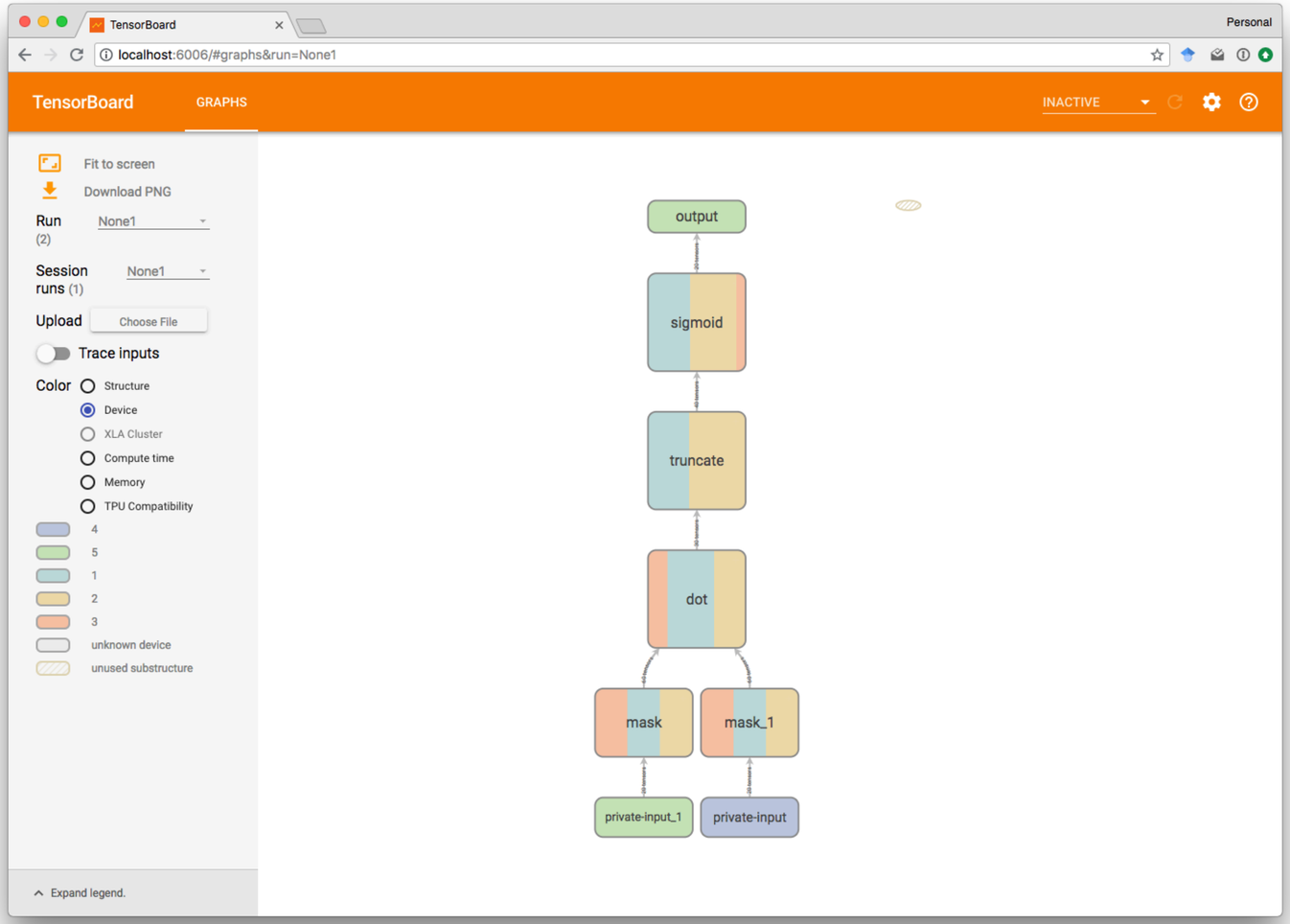
Compilation to Graphs



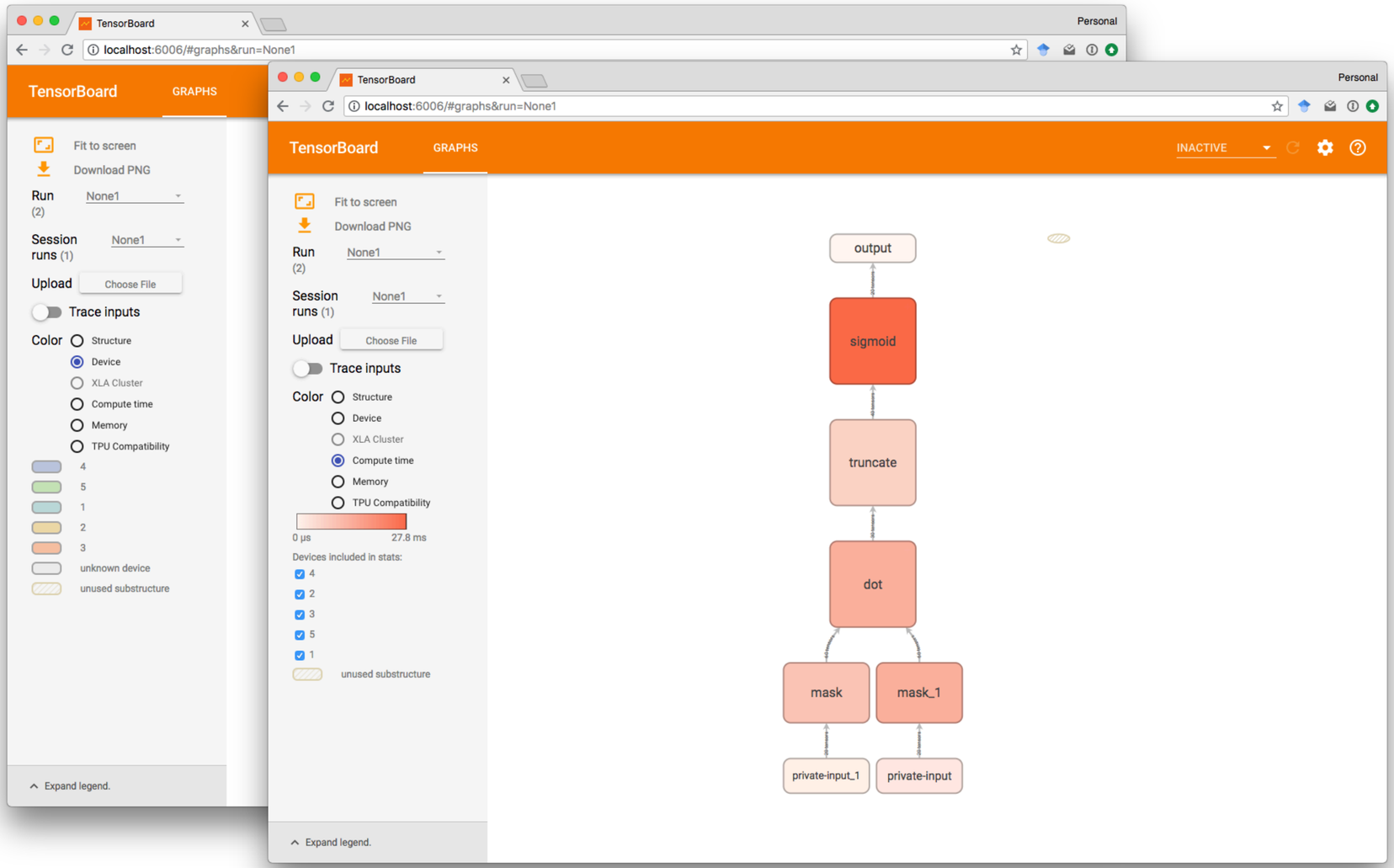
Compilation to Graphs



Profiling

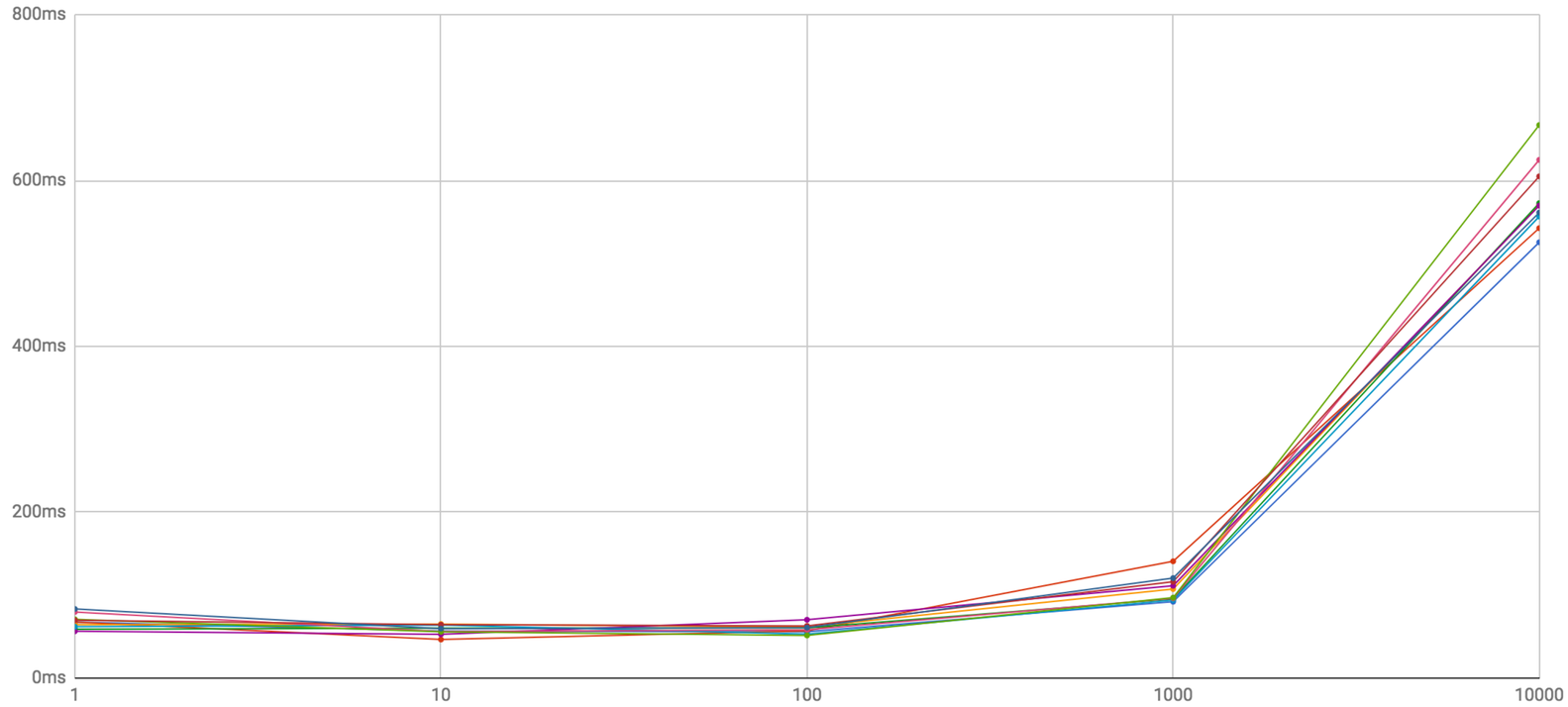

TensorBoard
Personal
localhost:6006/#graphs&run=None1
TensorBoard
GRAPHS
INACTIVE
Fit to screen
Download PNG
Run (2)
None1
Session runs (1)
None1
Upload
Choose File
Trace inputs
Color
Structure
Device
XLA Cluster
Compute time
Memory
TPU Compatibility
4
5
1
2
3
unknown device
unused substructure
Expand legend.
output
sigmoid
truncate
dot
mask
mask_1
private-input_1
private-input

Profiling


TensorBoard
Personal
localhost:6006/#graphs&run=None1
TensorBoard
GRAPHS
INACTIVE
Fit to screen
Download PNG
Run (2)
None1
Session runs (1)
None1
Upload
Choose File
Trace inputs
Color
Structure
Device
XLA Cluster
Compute time
Memory
TPU Compatibility
0 µs
27.8 ms
Devices included in stats:
4
2
3
5
1
unused substructure
Expand legend.
output
sigmoid
truncate
dot
mask
mask_1
private-input_1
private-input

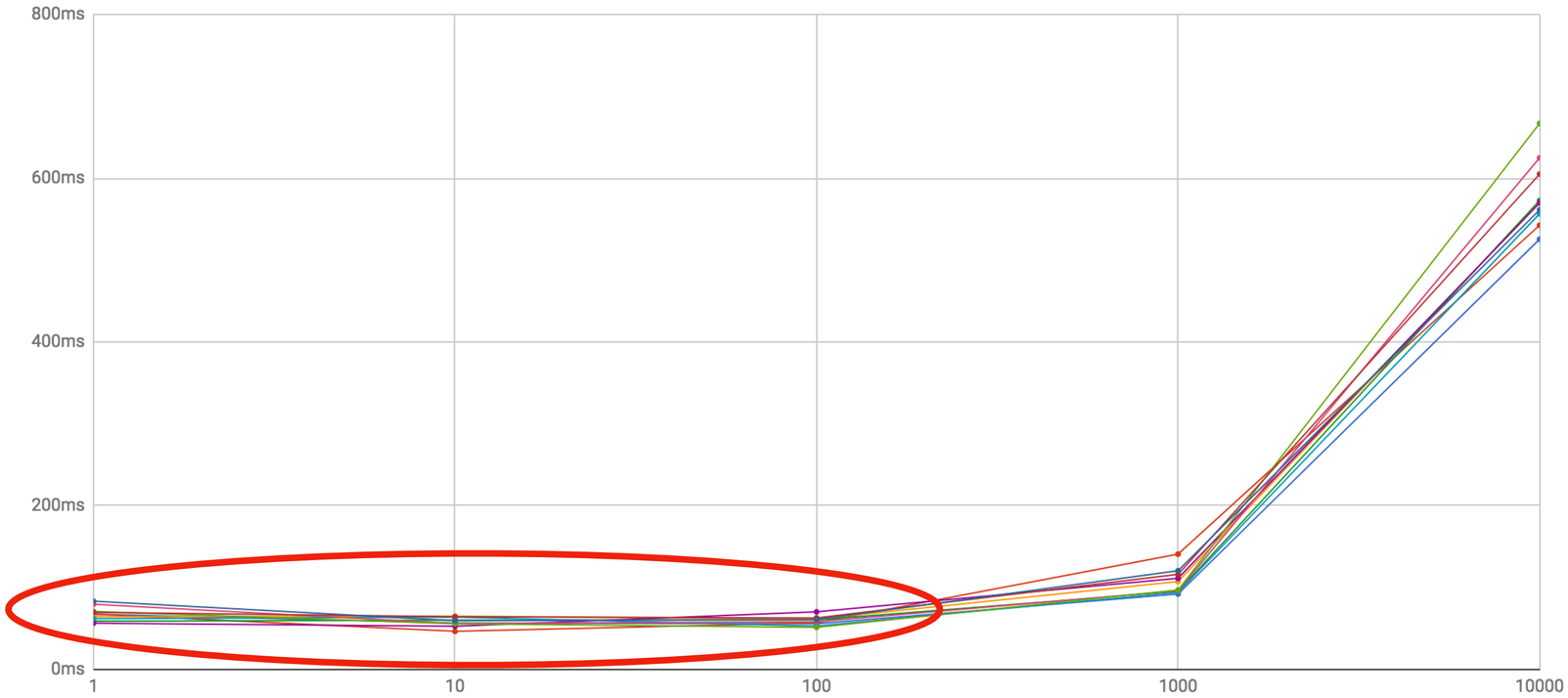
Performance

100 features, servers on Google cloud (2 vCPU, 10 GB)



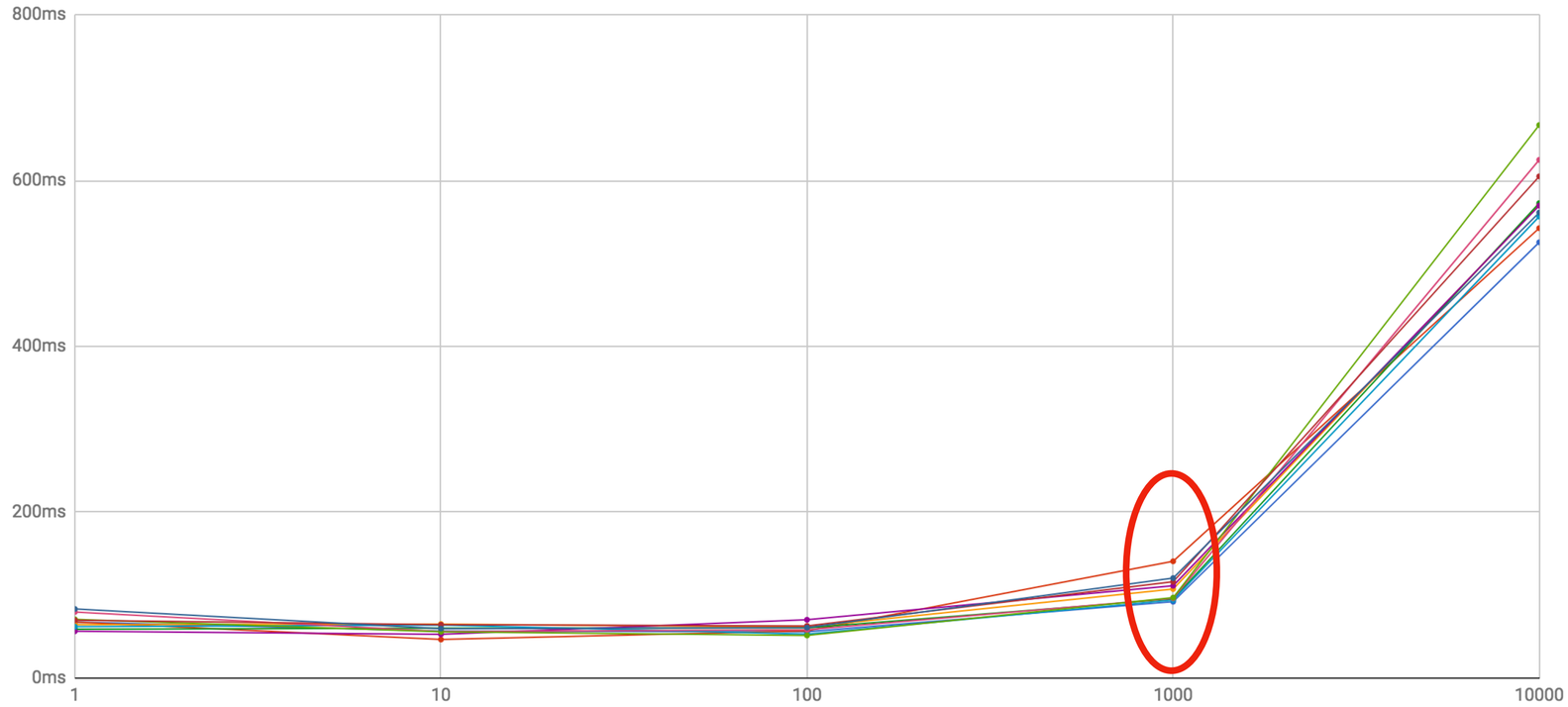
Performance

100 features, servers on Google cloud (2 vCPU, 10 GB)



Performance

100 features, servers on Google cloud (2 vCPU, 10 GB)



Performance

100 features, servers on Google cloud (2 vCPU, 10 GB)