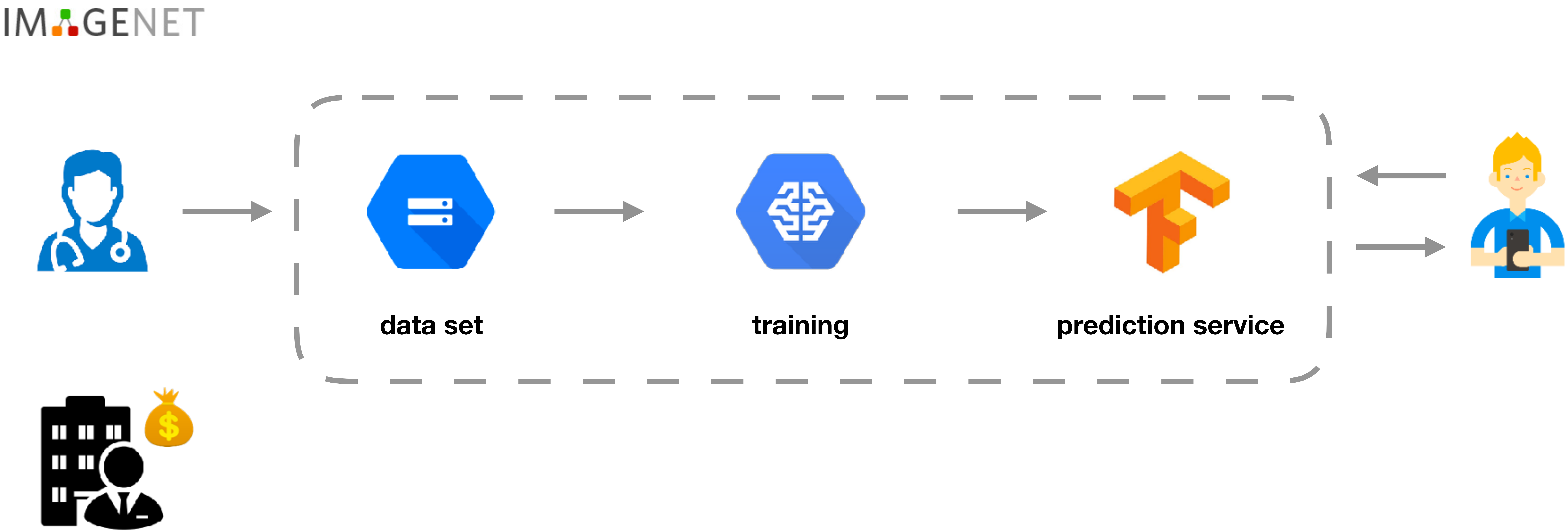
Technologies for Private Machine Learning

Morten Dahl

IFIP Summer School, August 2018

Why?

Machine Learning Process

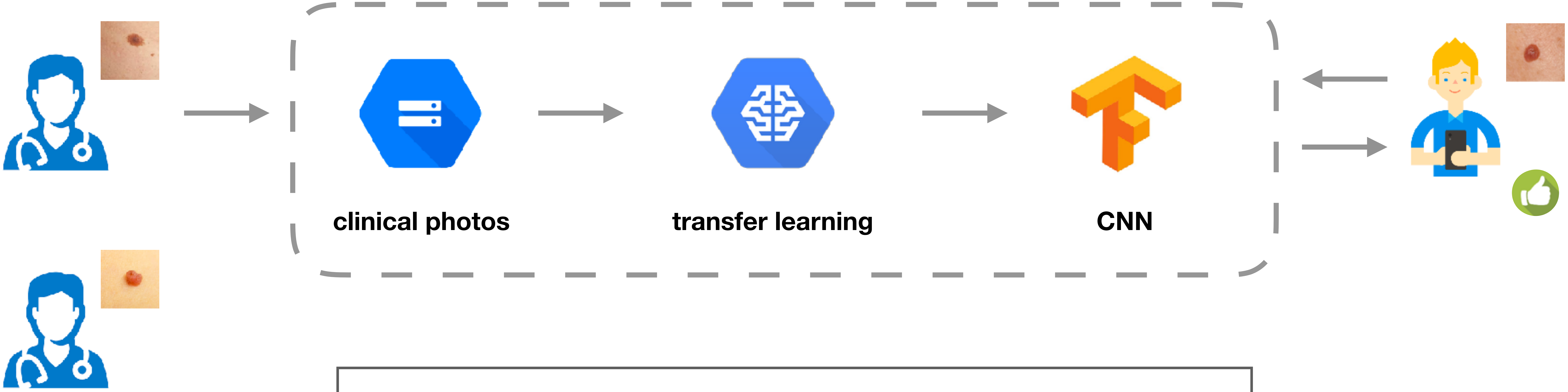


Skin Cancer Image Classification

Brett Kuprel

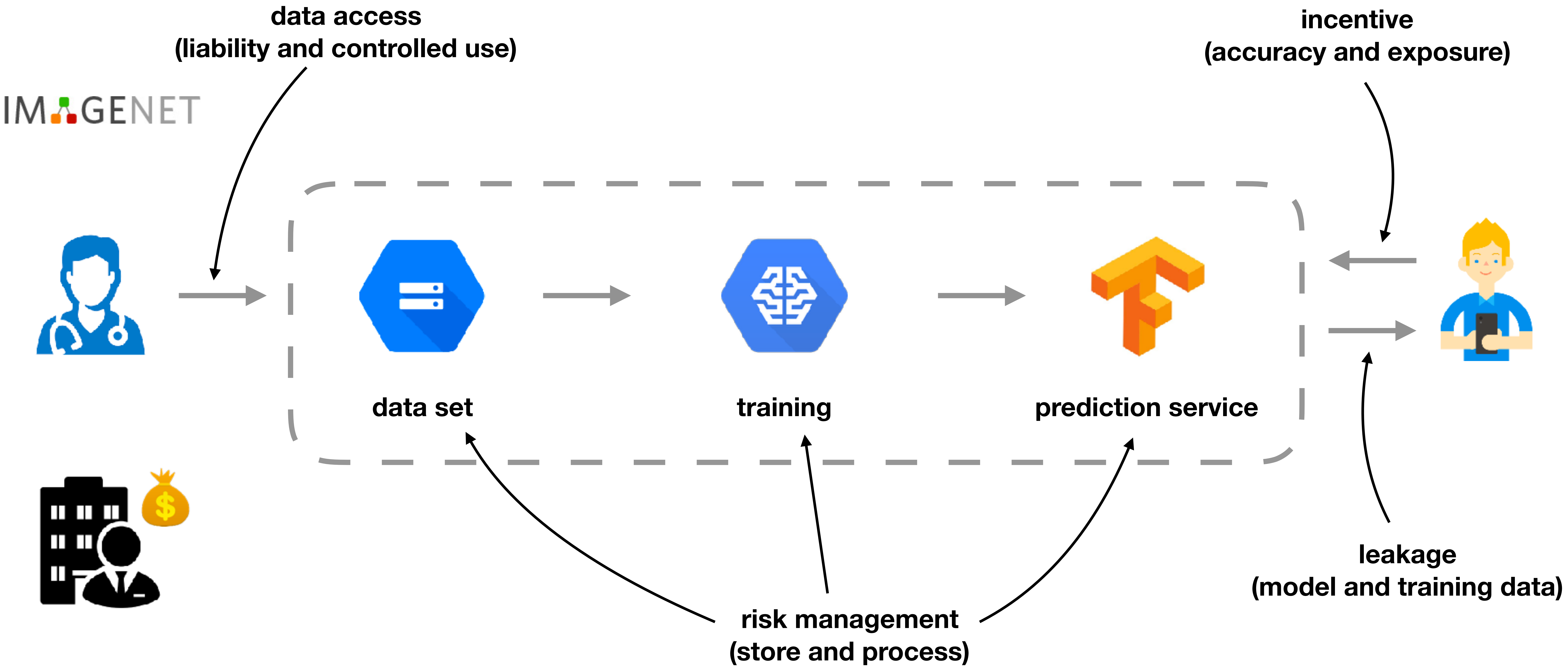
12:30-12:40pm

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.

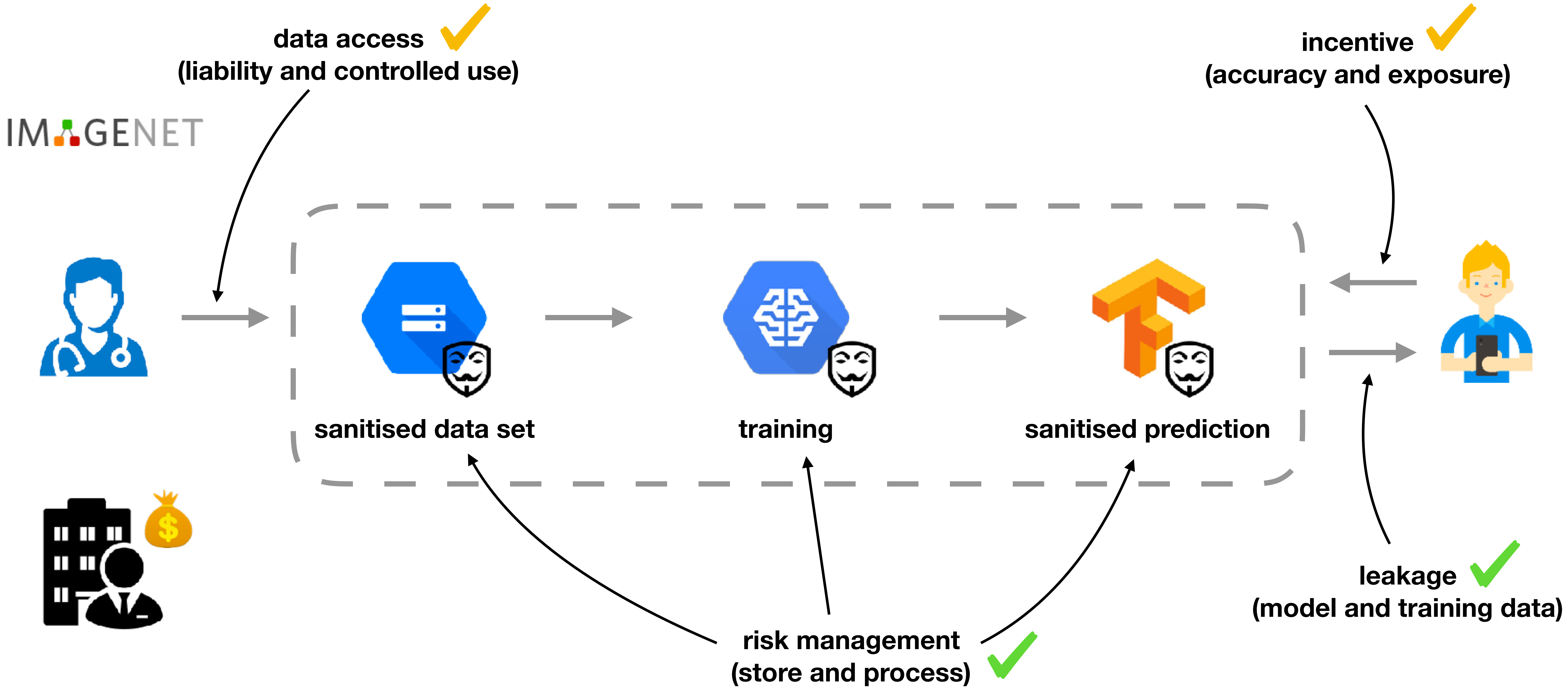


machine learning on sensitive information holds potential for big impact

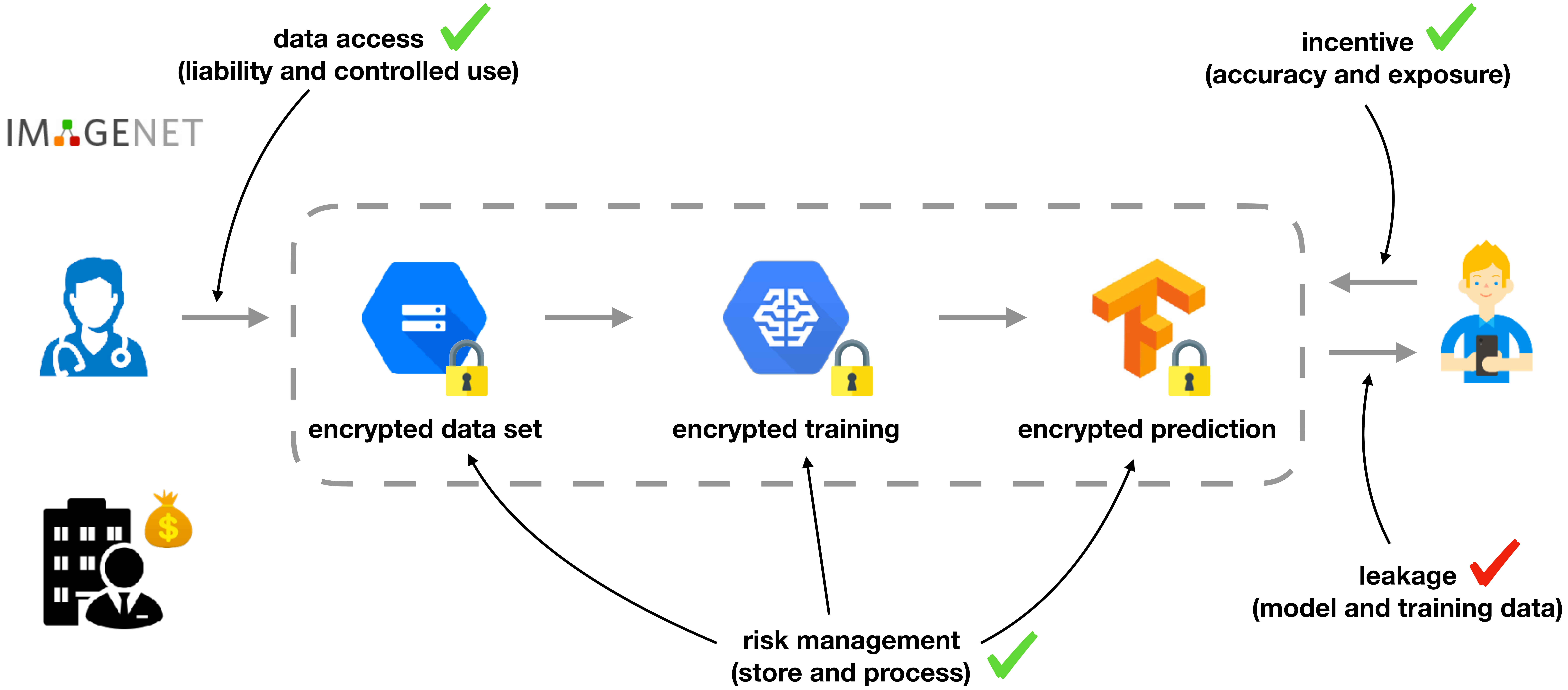
Potential Bottlenecks



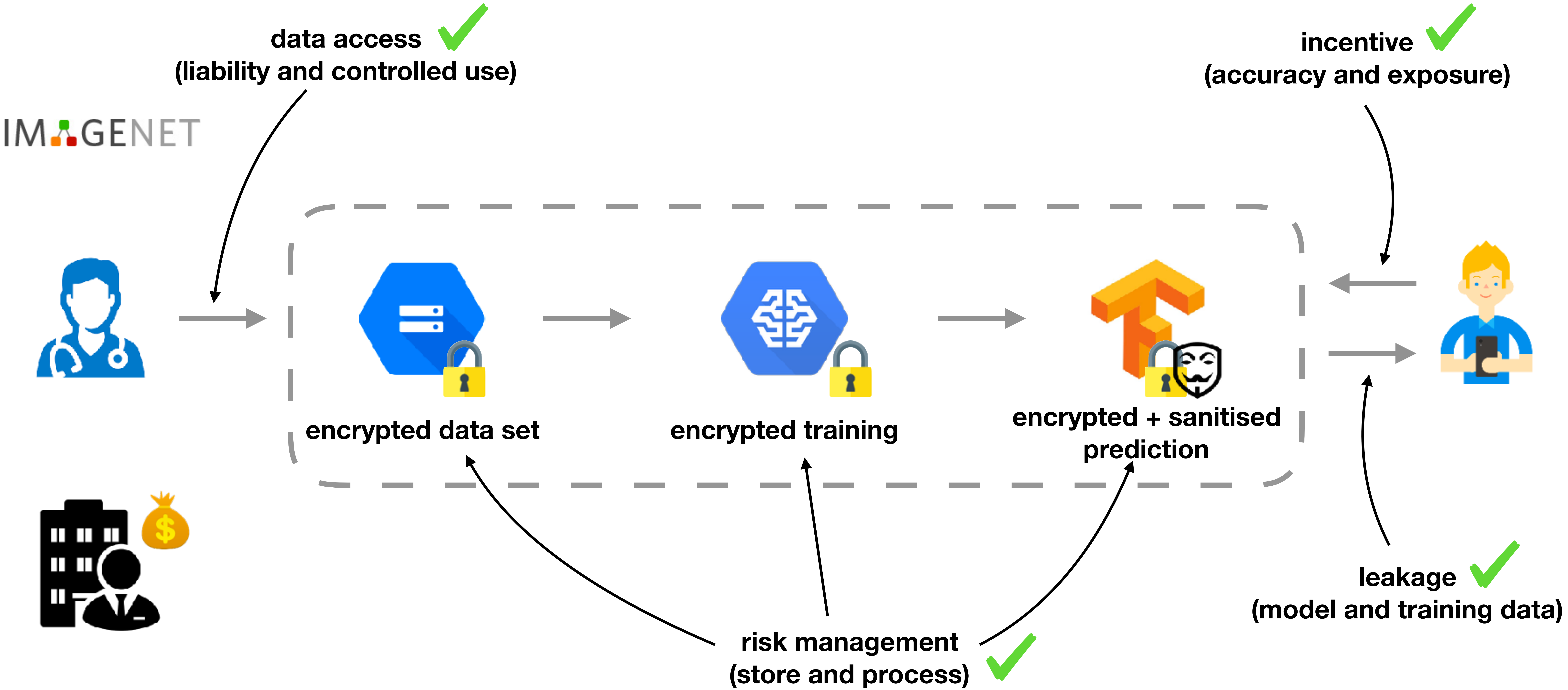
Differential Privacy



Secure Computation



Hybrid


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

Secure Computation

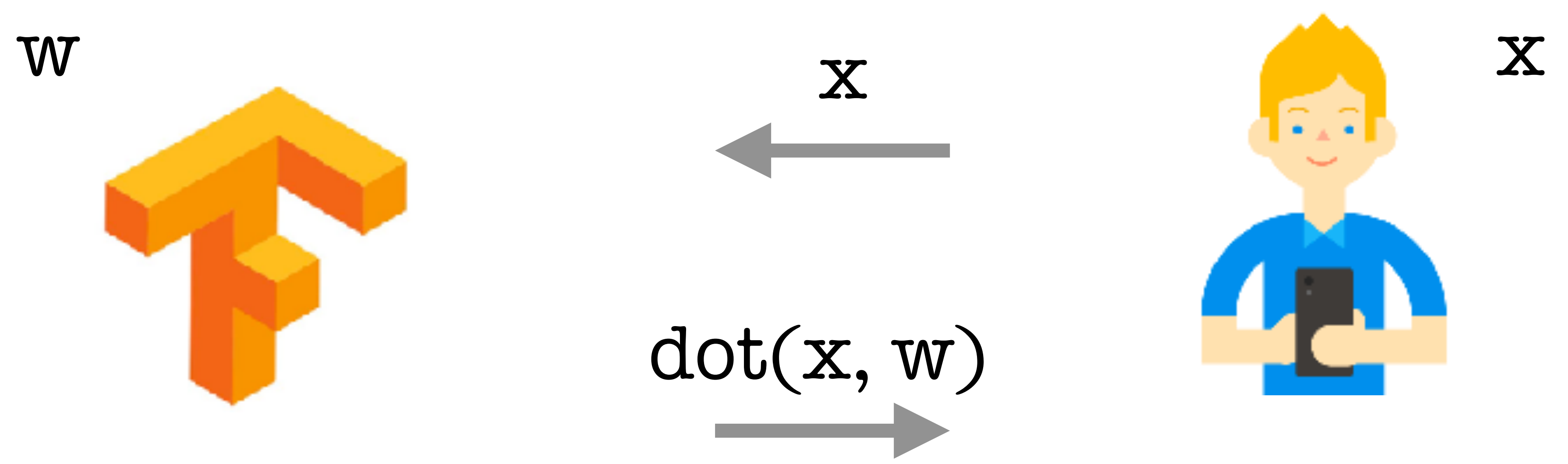
Technologies

	Homomorphic Encryption	Secret Sharing	Garbled Circuits
Computation	*heavy*	*light*	*medium*
Communication	*light*	*heavy*	*medium*
Encoding	*number + binary*	*number + binary*	*binary only*
Optimised operations	*few*	*several*	*some*

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

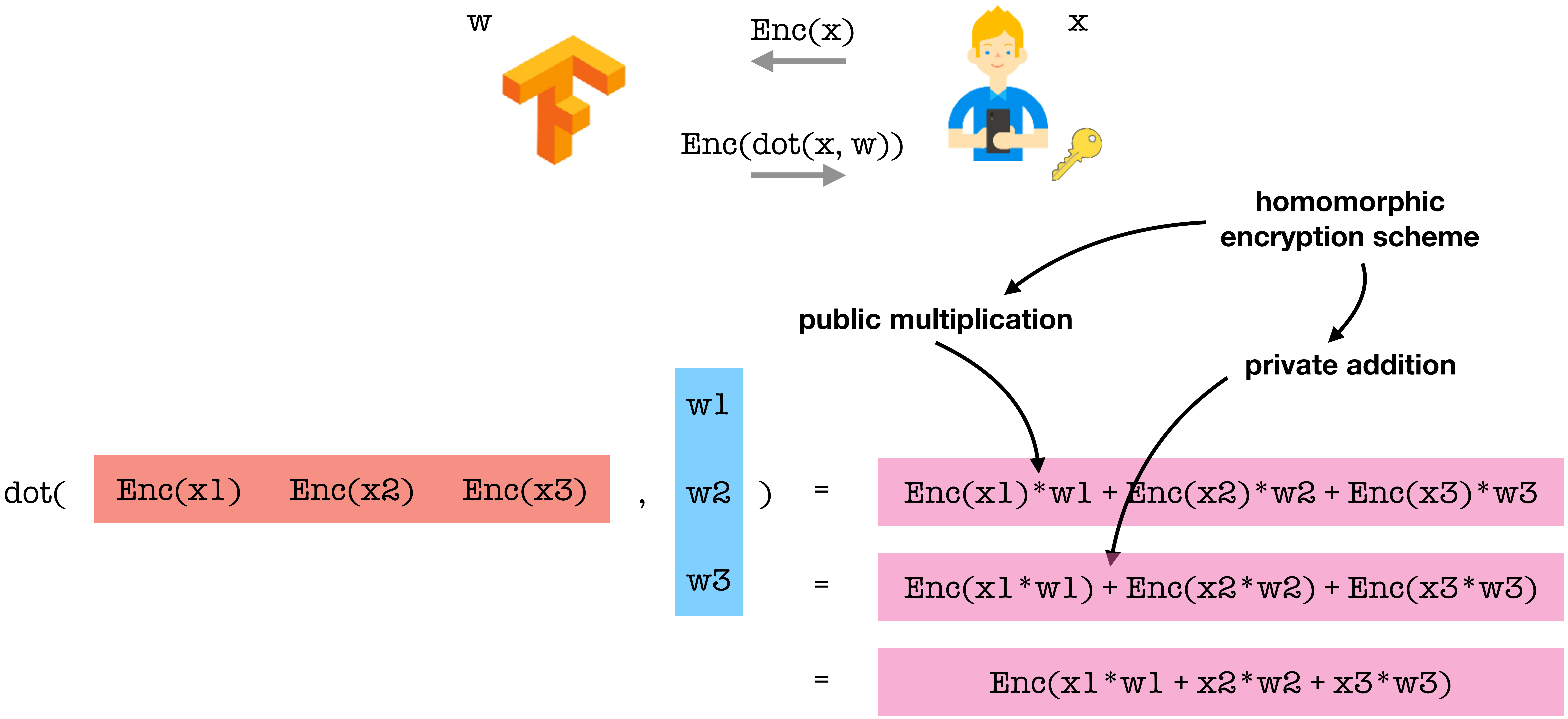
Linear Regression

Prediction with Linear Regression Model

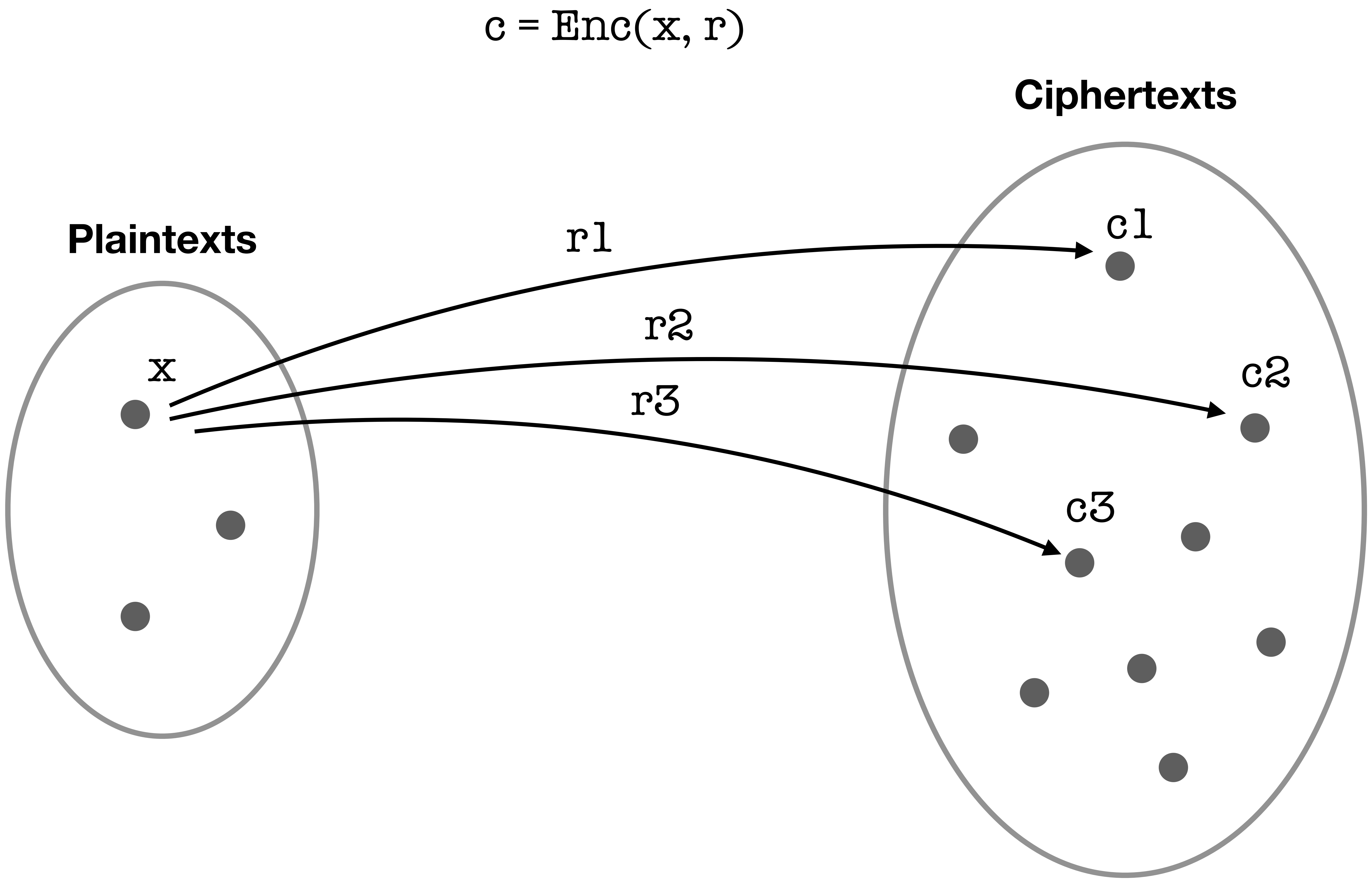


$$\mathrm{dot}\left(\begin{bmatrix} x1 & x2 & x3 \end{bmatrix}, \begin{bmatrix} w1 \\ w2 \\ w3 \end{bmatrix} \right) = x1*w1 + x2*w2 + x3*w3$$

Using Homomorphic Encryption



Paillier Homomorphic Encryption



Paillier Homomorphic Encryption

typically ~4000 bits:
computation is significantly
more expensive

public encryption key

c = Enc(x, r) = g^x * r^n mod n^2

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

$$\mathrm{Enc}(x, r) * \mathrm{Enc}(y, s)$$

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

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

$$= \mathrm{Enc}(x+y, r*s)$$

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

Public Multiplication in Paillier

$$\mathrm{Enc}(x, r) \wedge w$$

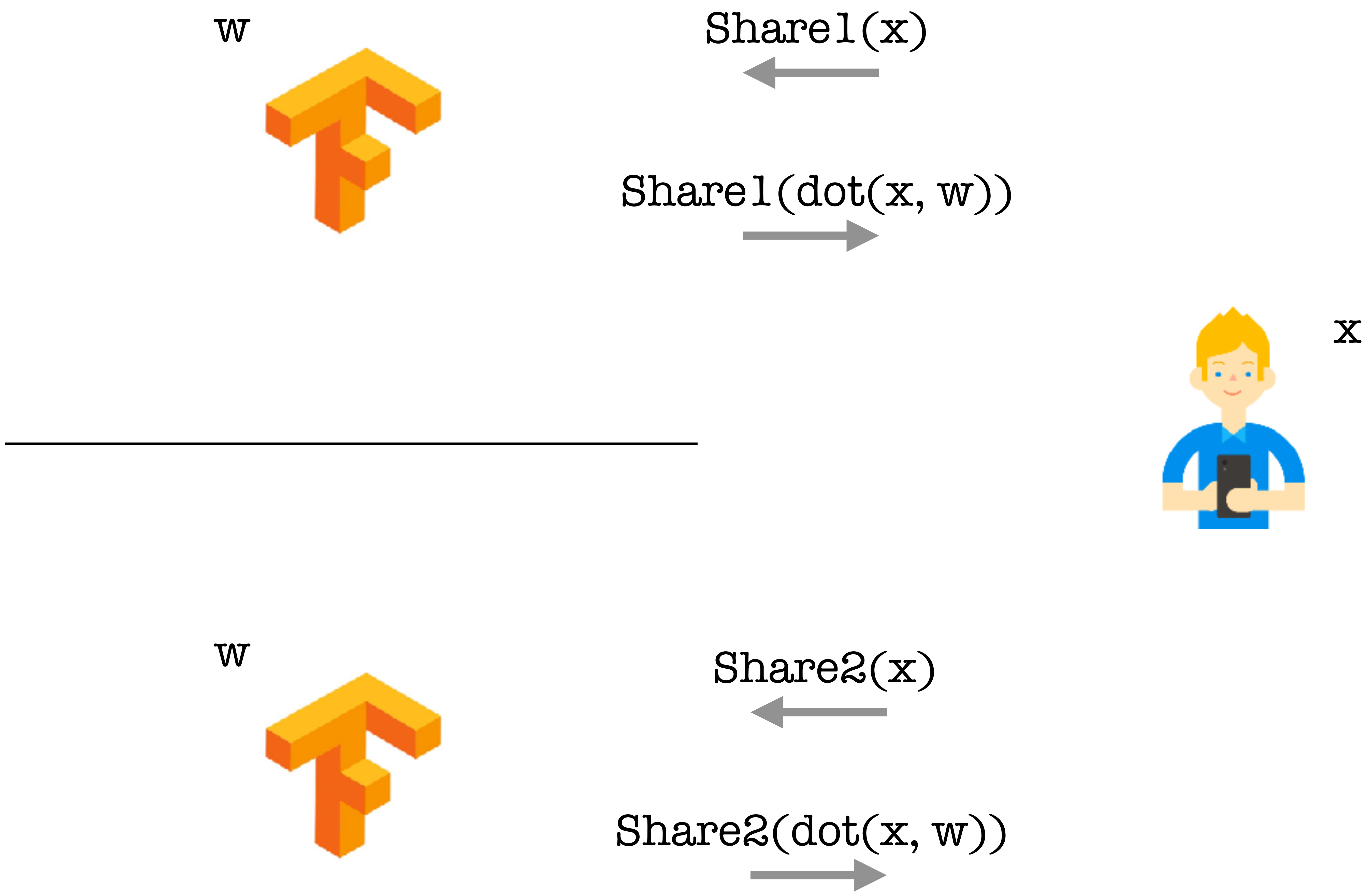
$$= (g \wedge x * r \wedge n \bmod n \wedge 2) \wedge w$$

$$= g \wedge (x * w) * (r \wedge w) \wedge n \bmod n \wedge 2$$

$$= \mathrm{Enc}(x * w, r \wedge w)$$

Enc(5, 2) ^ 2
= 718 * 718
= 1024
= 36^10 * 4^35
= Enc(10, 4)

Using Secret Sharing



Using Secret Sharing

w

Share1(x)

homomorphic secret sharing

public multiplication

private addition

dot(Share1(x0) Share1(x1) Share1(x2) , w0 w1 w2) = Share1(x0)*w0 + Share1(x1)*w1 + Share1(x2)*w2

= Share1(x0*w0) + Share1(x1*w1) + Share1(x2*w2)

= Share1(x0*w0 + x1*w1 + x2*w2)

SPDZ Secret Sharing

public parameter

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

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

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

$m = 10$

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

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

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

Private Addition in SPDZ

s1 t1 u1 = s1 + t1

s2 t2 u2 = s2 + t2

x = s1 + s2

y = t1 + t2

x + y
= (s1 + s2) + (t1 + t2)
= (s1 + t1) + (s2 + t2)
= u1 + u2

Public Multiplication in SPDZ

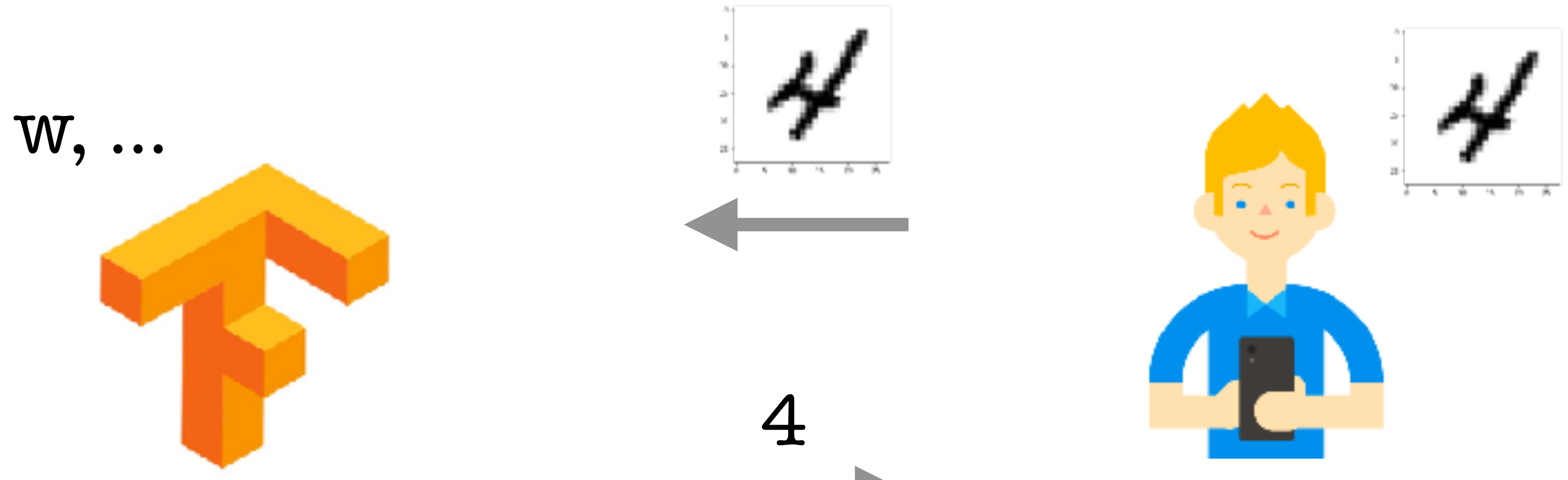
s1	w	u1 = s1 * w
s2	w	u2 = s2 * w

x = s1 + s2

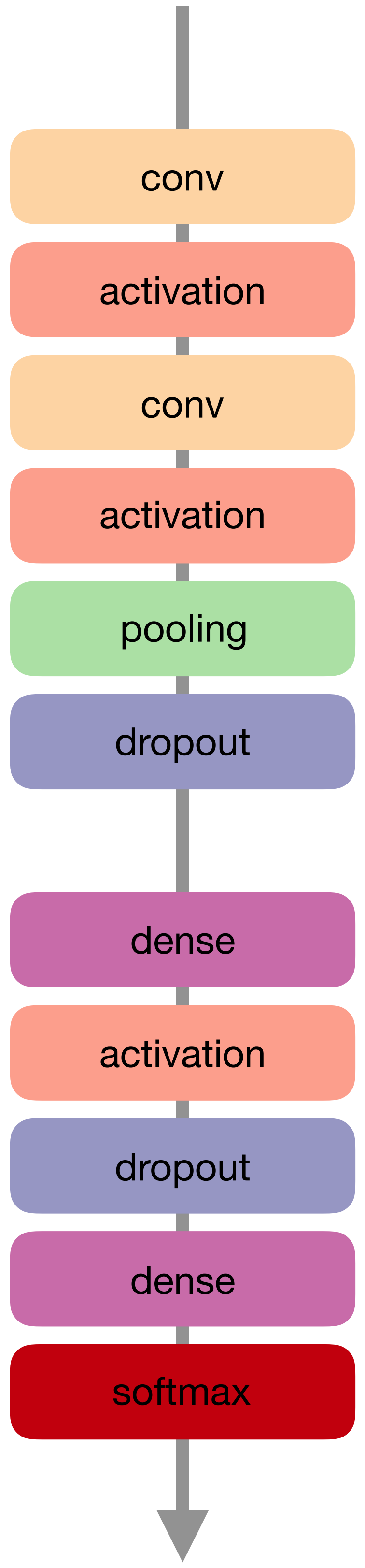
x * w
= (s1 + s2) * w
= (s1 * w) + (s2 * w)
= u1 + u2

Convolutional Neural Networks

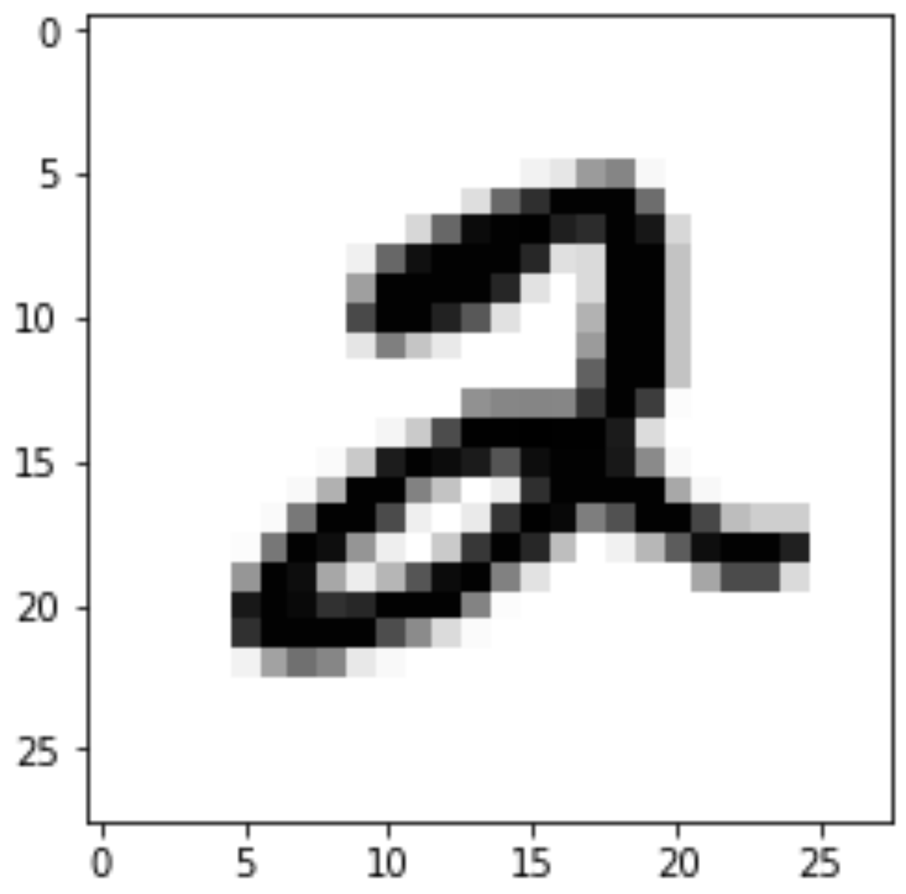
Digit Classification with CNNs




```
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)
```

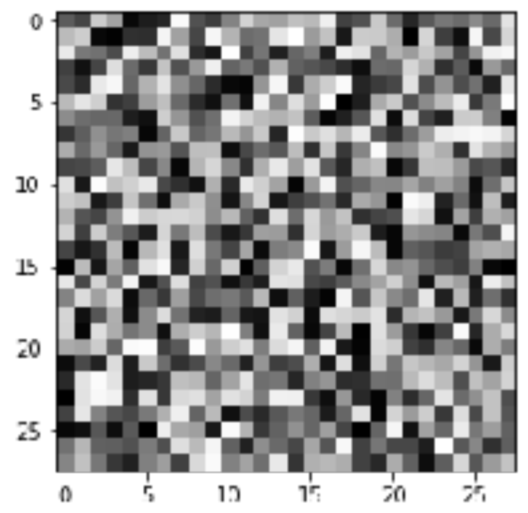


Secret Sharing Images

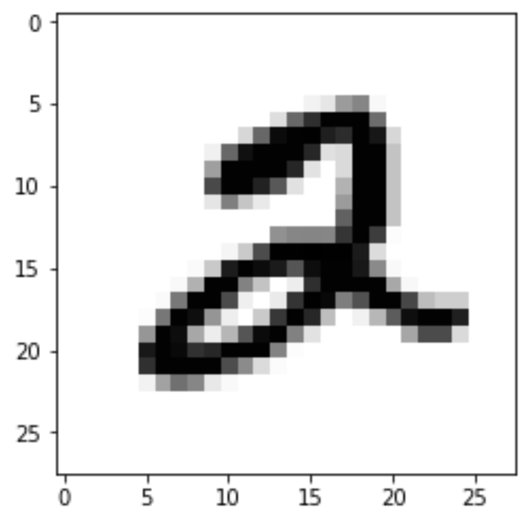



```
array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0, 143, 247, 153,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0, 136, 247, 242,  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, 192, 252, 187,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  62,
        185,  18,   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, 216,
        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, 206,
        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, 131, 251,
        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,  29, 232, 247,
         63,   0,   0,   0, 153, 252, 226,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,  45, 219, 252, 143,
          0,   0,   0, 116, 249, 252, 103,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   4,  96, 253, 255, 253, 200,
        122,   7,  25, 201, 250, 158,   0,   0,   0,   0,   0,   0,   0,
          0,   0]], dtype=uint8)
```

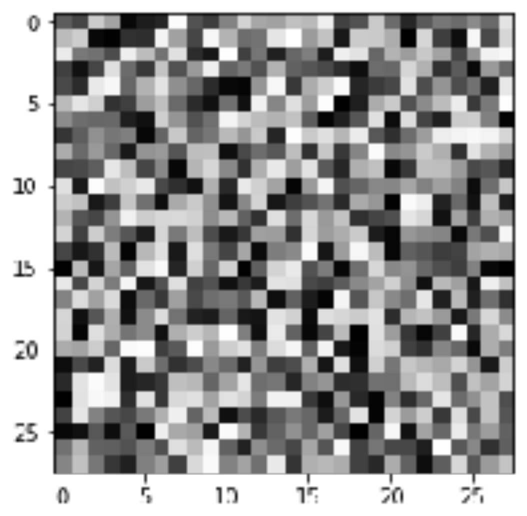
$\mathrm{Share1}(x, r) = r$ =



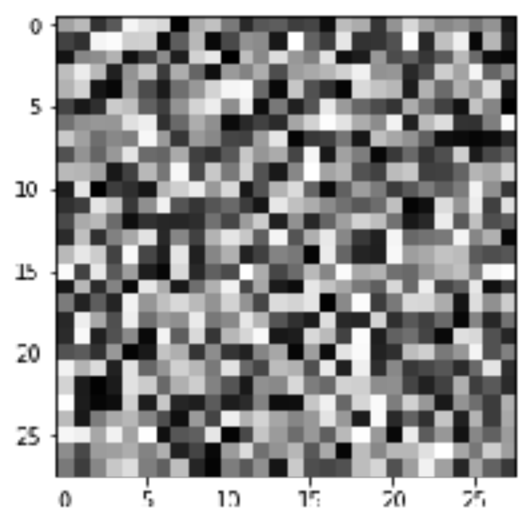
$\mathrm{Share2}(x, r) = x - r$ =



\-

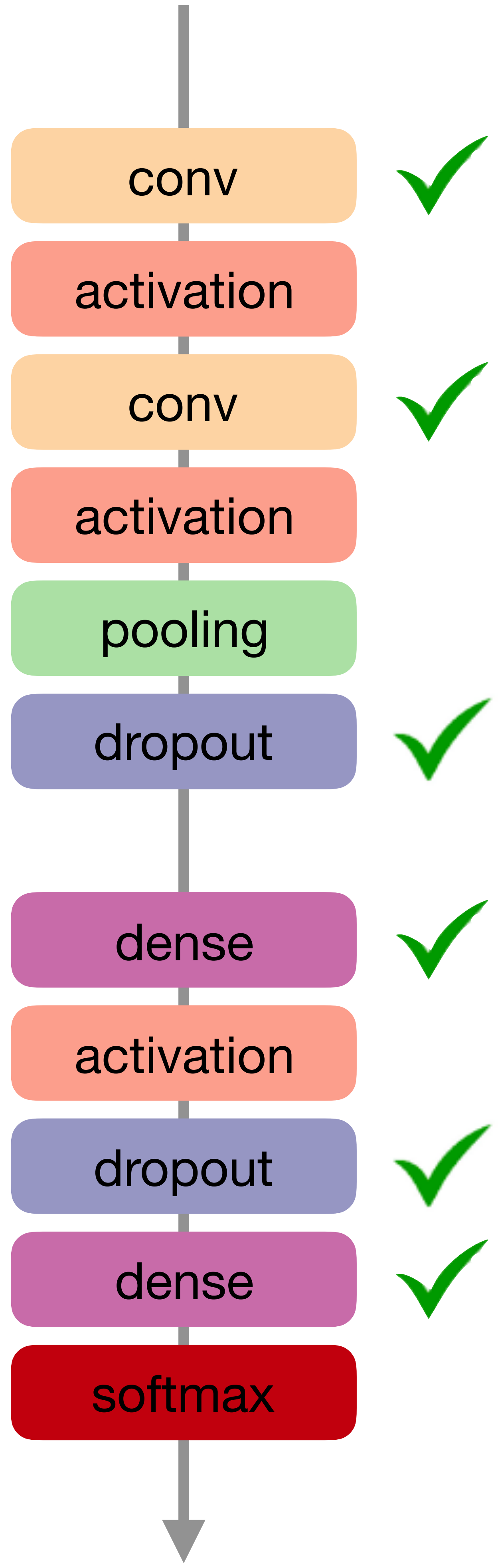


=



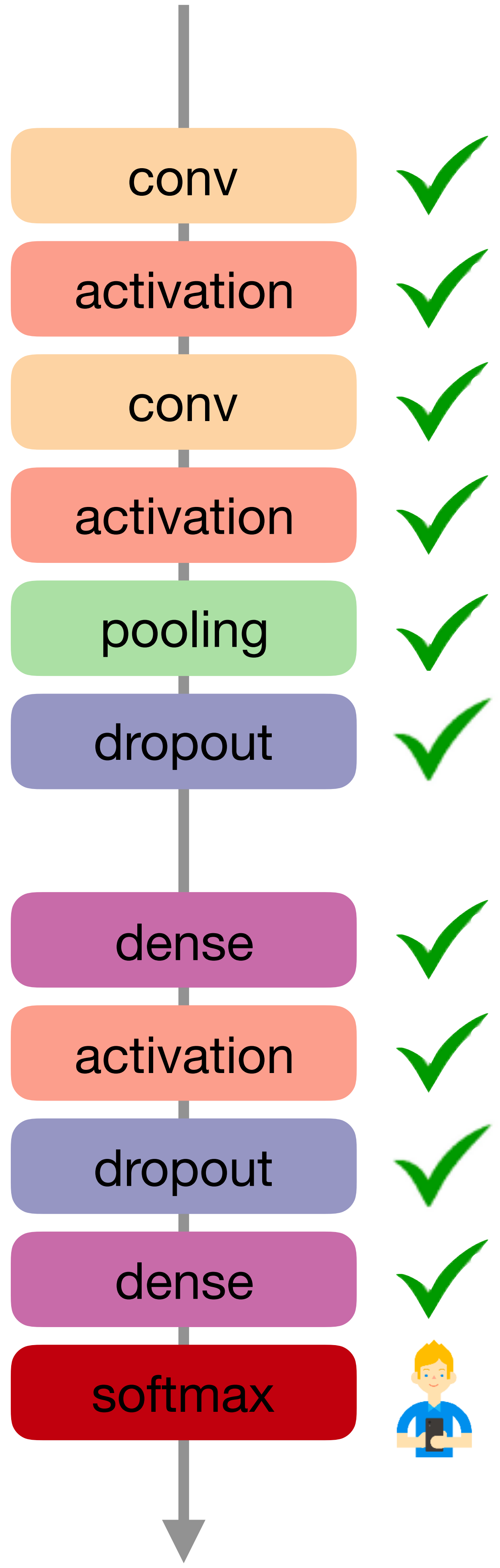
Digit Classification with Secret Sharing

```
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)
```

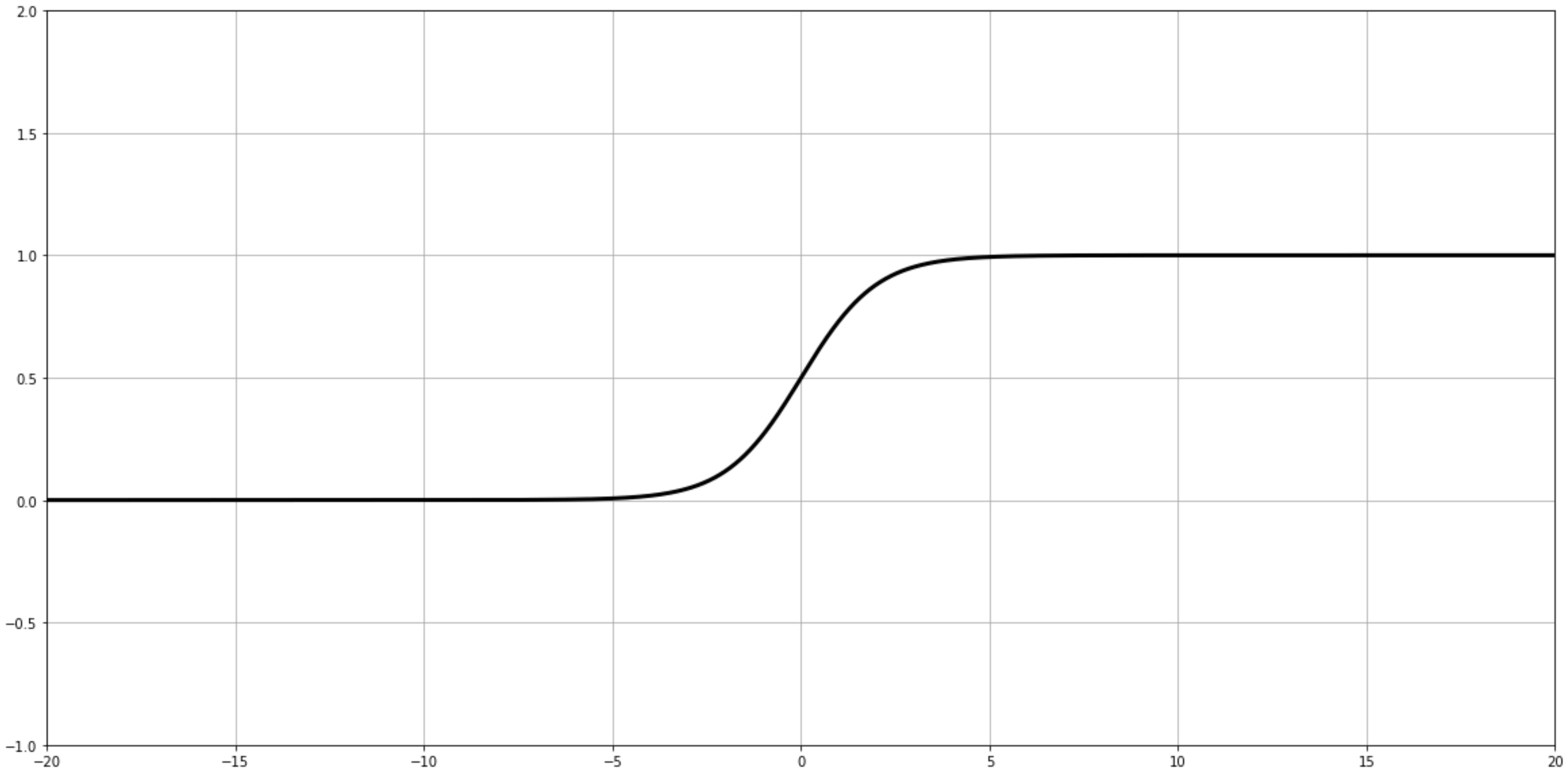


Digit Classification with Secret Sharing

```
feature_layers = [
    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()
]

classification_layers = [
    Dense(128),
    Activation('sigmoid'),
    Dropout(.50),
    Dense(5),
    Activation('softmax')
]

model = Sequential(feature_layers + classification_layers)
```

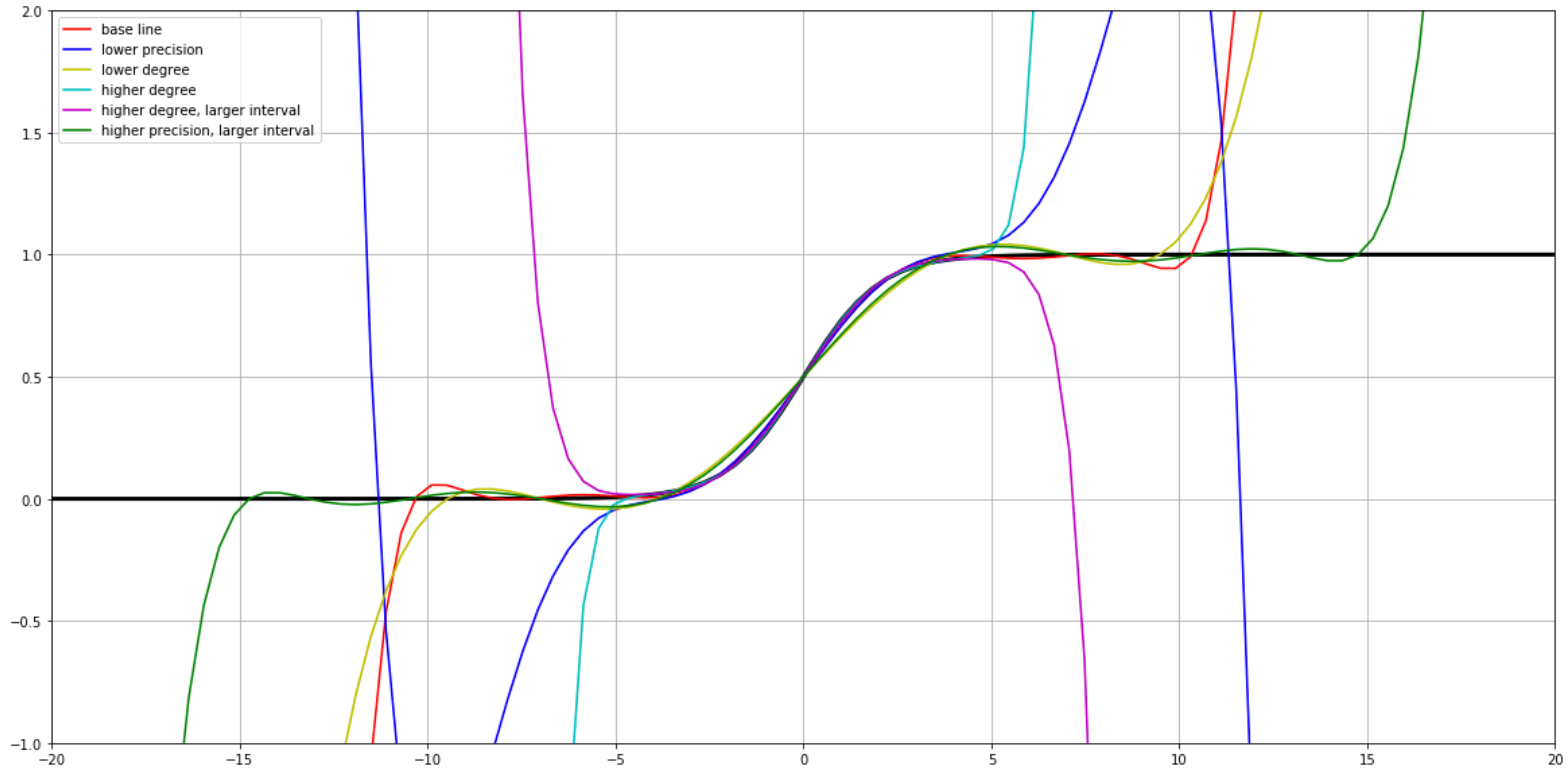


Sigmoid



$$f(x) = \frac{1}{1 + e^{-x}}$$

Sigmoid via Polynomial Approximation



f(x) = c7*x^7 + c5*x^5 + ... + c1*x + c0

Private Multiplication in SPDZ

random triple

(a_1, b_1, c_1) s_1 t_1 alpha beta

$$u_1 = \text{alpha} * \text{beta} + \text{alpha} * b_1 + \text{beta} * a_1 + c_1$$

(a_2, b_2, c_2) s_2 t_2 alpha beta

$$u_2 = \text{alpha} * b_2 + \text{beta} * a_2 + c_2$$

$a = a_1 + a_2$

$b = b_1 + b_2$

$c = a * b = c_1 + c_2$

$x = s_1 + s_2$

$y = t_1 + t_2$

$\text{alpha} = x - a$

$\text{beta} = y - b$

$$x * y = \ldots = u_1 + u_2$$

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.

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

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

Specialised projects

***OpenMined** (https://openmined.org)*

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

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)

OpenMined

OpenMined is a community focused on
researching, developing, and spreading tools for
secure, privacy-preserving, and value-aligned artificial intelligence

Slides originally by Andrew Trask


Secure Computation +
Federated Learning
AI Inc.
Model
Jane's
Data
Jack's
Data
Joe's
Data
Federated Learning for Safe AI

tf-encrypted

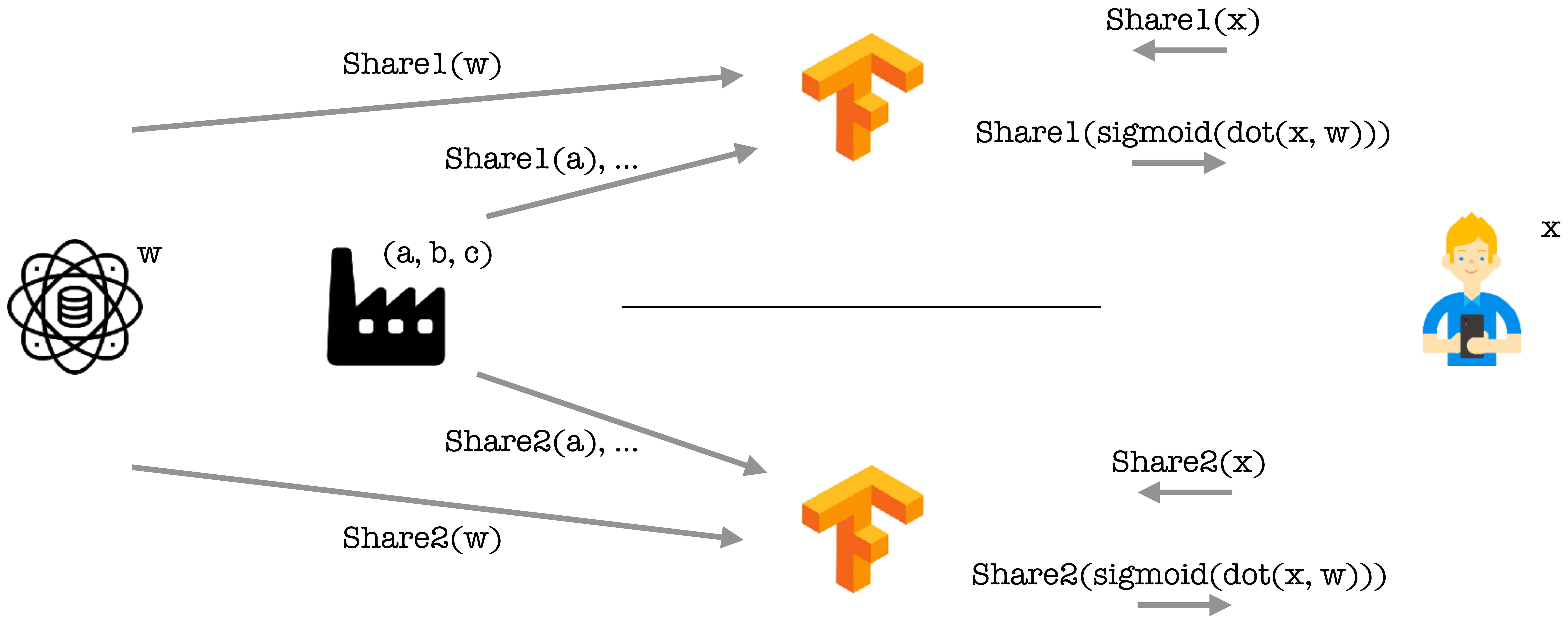
platform on top of TensorFlow for experimenting
with private machine learning

integrates with TensorFlow
for easy mix with existing functionality

aims at being extendable by both
cryptographers and machine learners

benefits from TensorFlow's interface and backend
for ease and efficiency

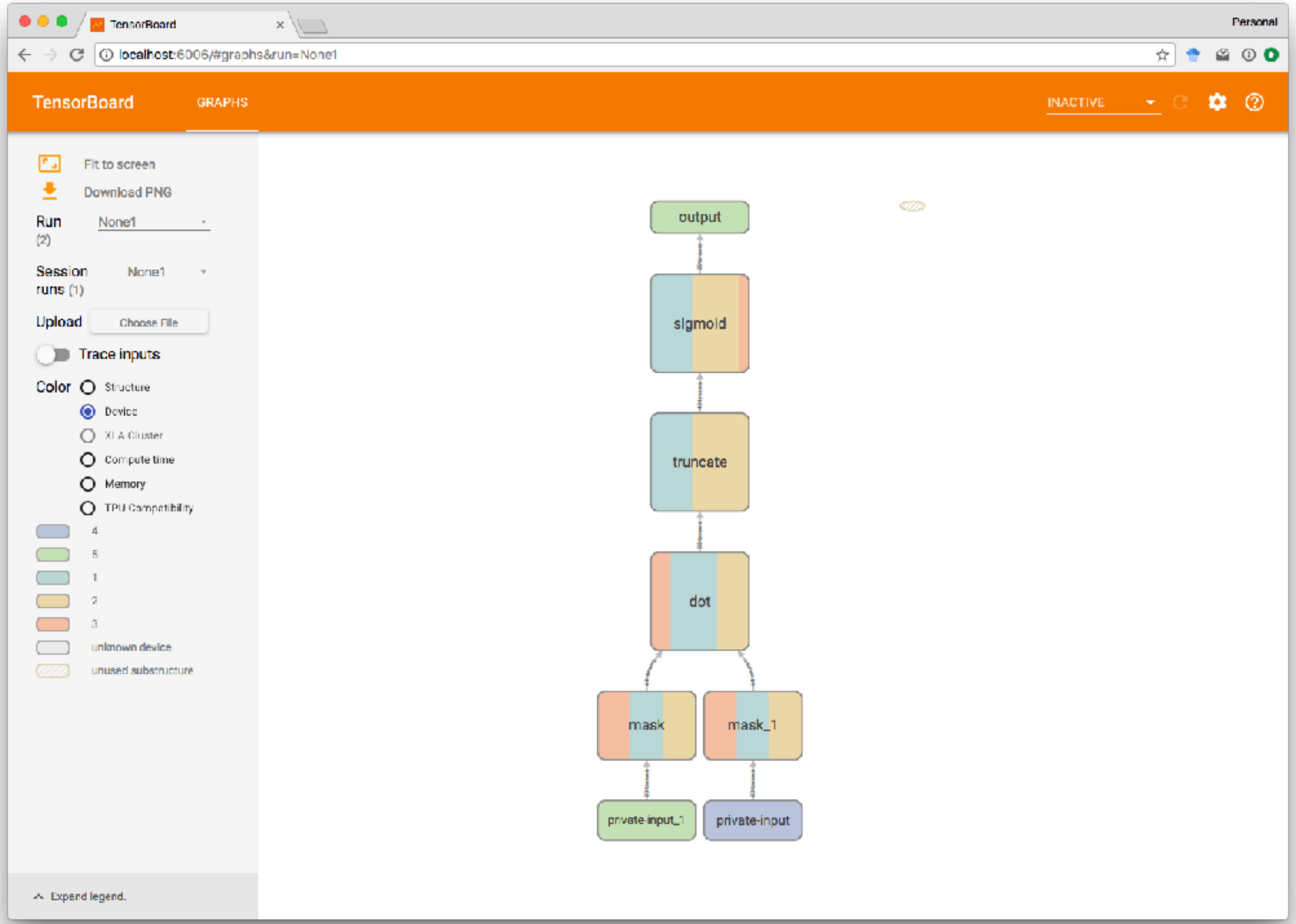
Logistic Regression



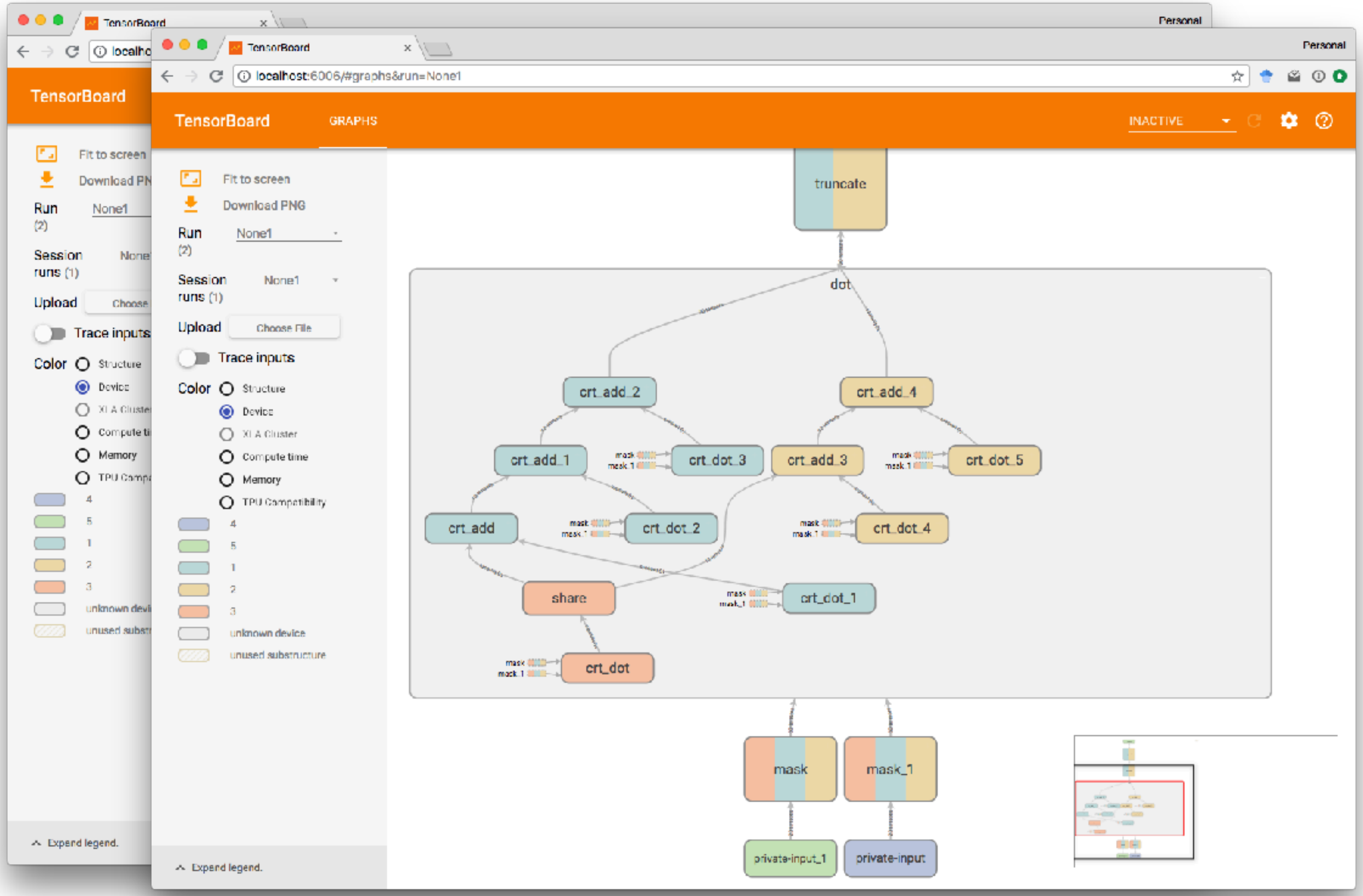
Easy Expression

```python
with tfe.protocol.Pond(server0, server1, crypto_producer) as prot:

    w = prot.define_private_input(weights_input)
    x = prot.define_private_input(prediction_input)

    # compute prediction
    y = prot.sigmoid(prot.dot(x, w))

    prediction_op = prot.define_output(y, prediction_output)

    with config.session() as sess:
        # init
        tfe.run(sess, tf.global_variables_initializer())

        # run encrypted prediction
        tfe.run(sess, prediction_op)
```

Compilation to Graphs



Compilation to Graphs



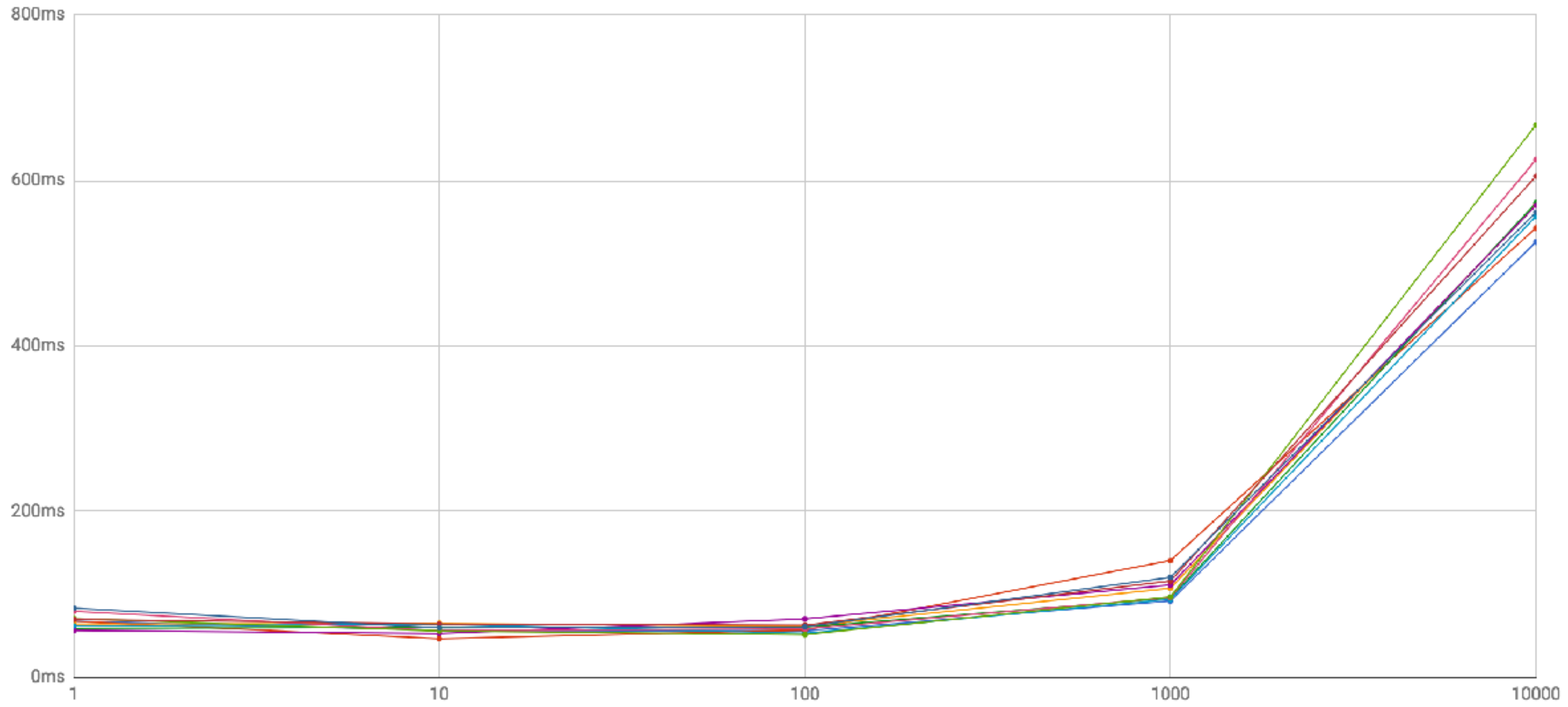
Input Integration

```
class WeightsInputProvider(tfe.io.InputProvider):
    def provide_input(self) -> tf.Tensor:
        raw_w = np.array([1, 2, 3, 4]).reshape((2,2))
        return tf.constant(raw_w)
```

```
class PredictionInputProvider(tfe.io.InputProvider):
    def provide_input(self) -> tf.Tensor:
        raw_x = np.array([5, 5, 5, 5]).reshape((2,2))
        return tf.constant(raw_x)
```

Performance

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



Thank you!

mortendahl.github.io

@mortendahlcs