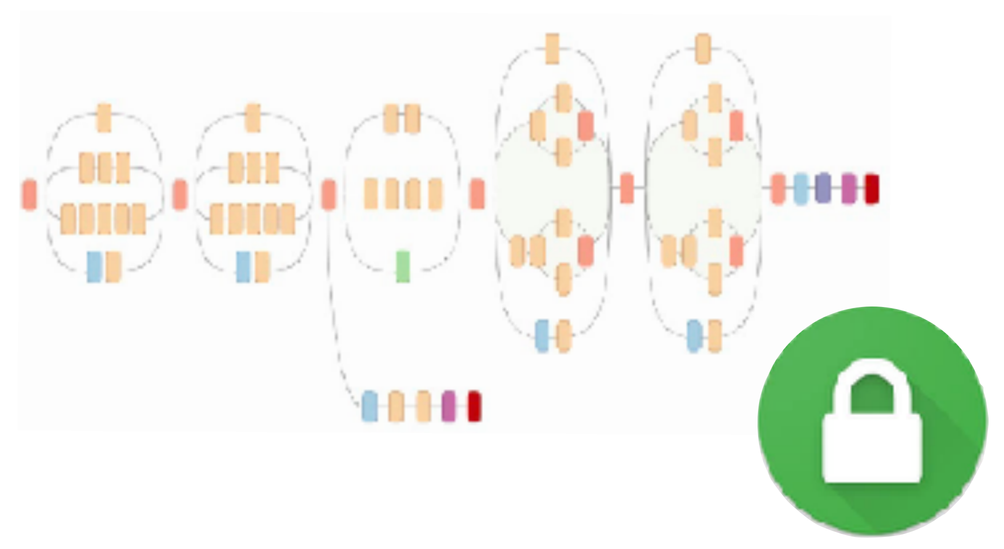
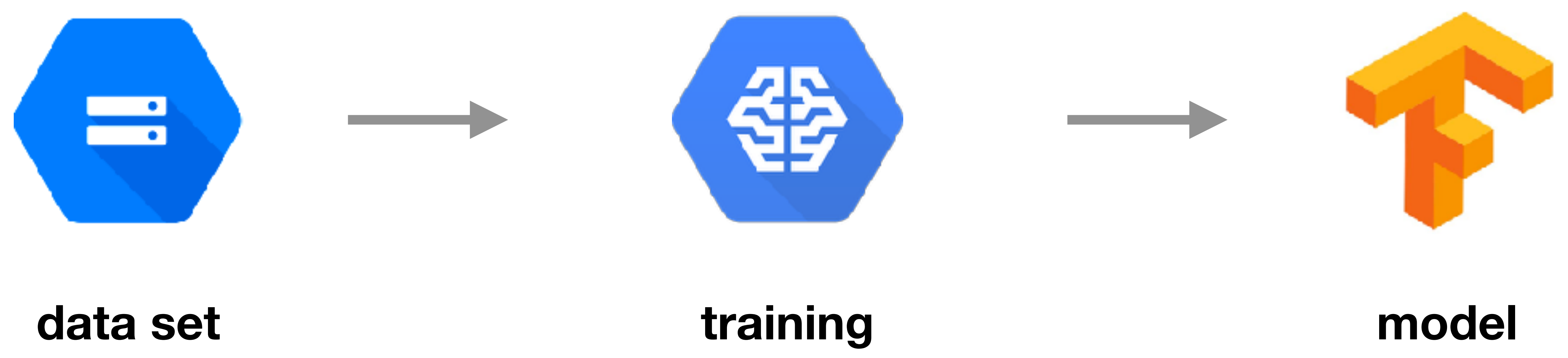
Private Machine Learning



Morten Dahl

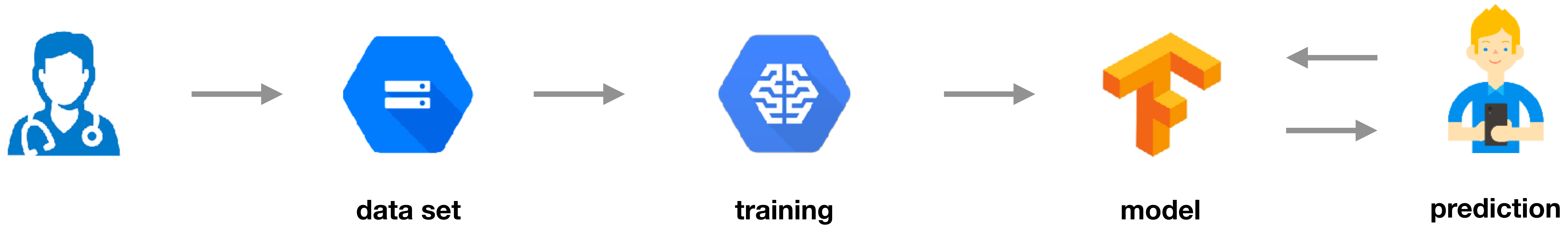
Paris Machine Learning meetup, December 2017

Machine Learning Process



Machine Learning Process

IMAGENET

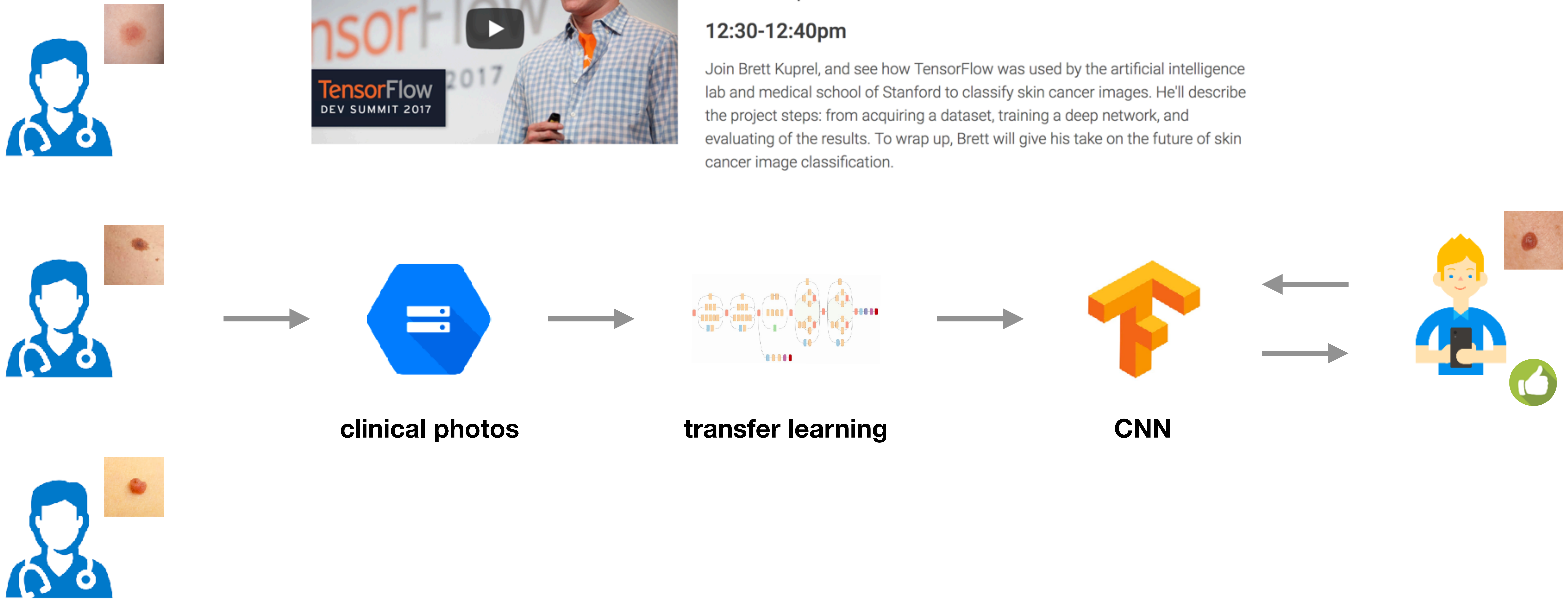


Skin Cancer Image Classification

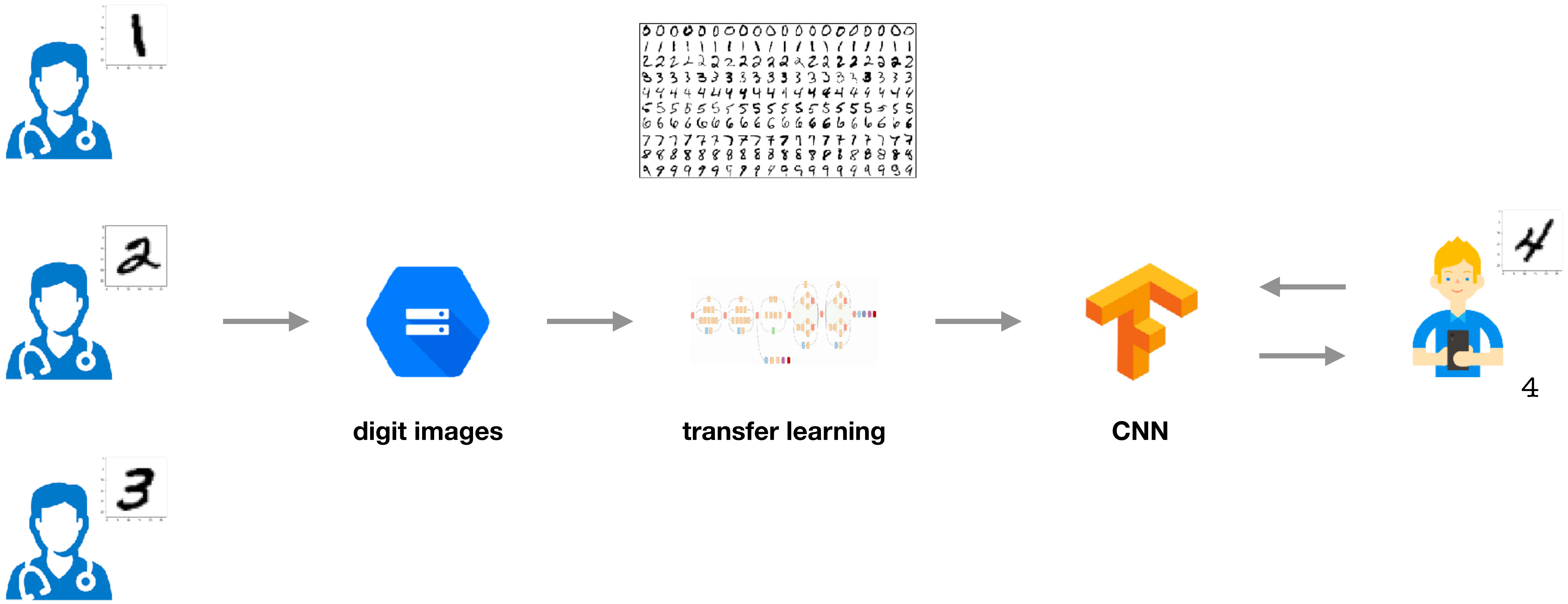
Brett Kuprel

12:30-12:40pm

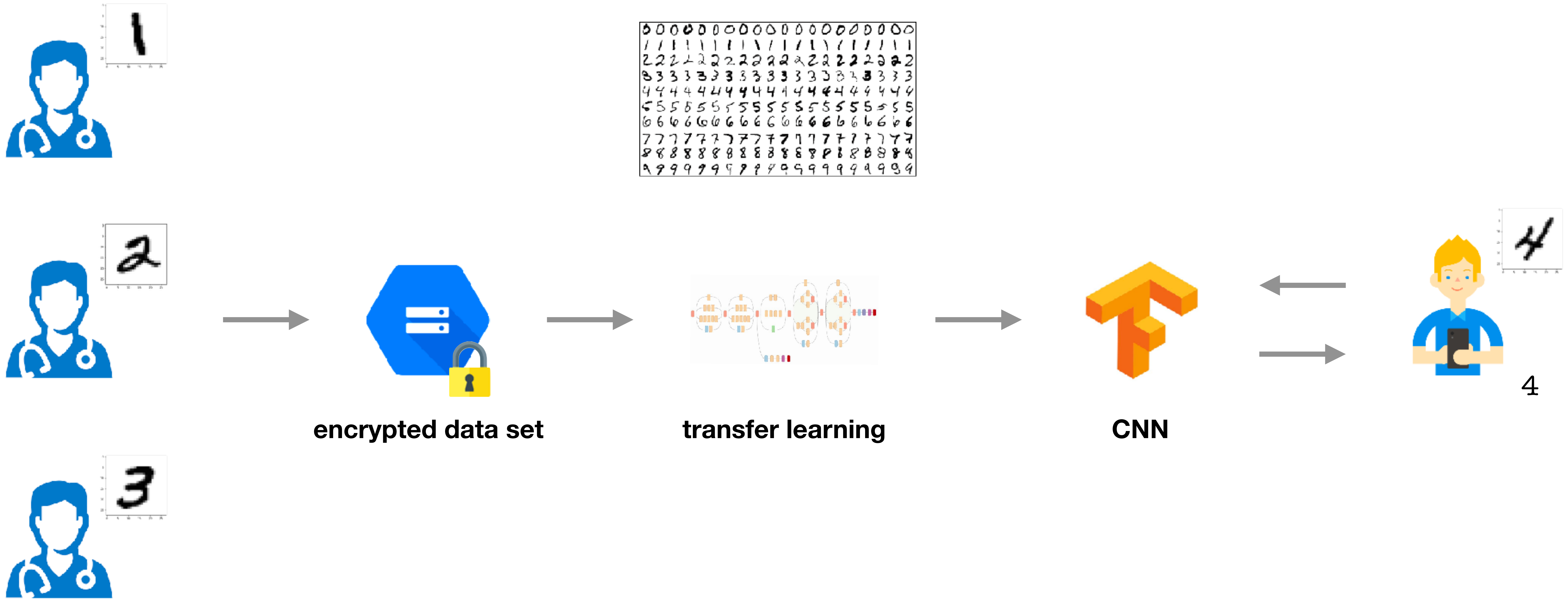
Join Brett Kuprel, and see how TensorFlow was used by the artificial intelligence lab and medical school of Stanford to classify skin cancer images. He'll describe the project steps: from acquiring a dataset, training a deep network, and evaluating of the results. To wrap up, Brett will give his take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



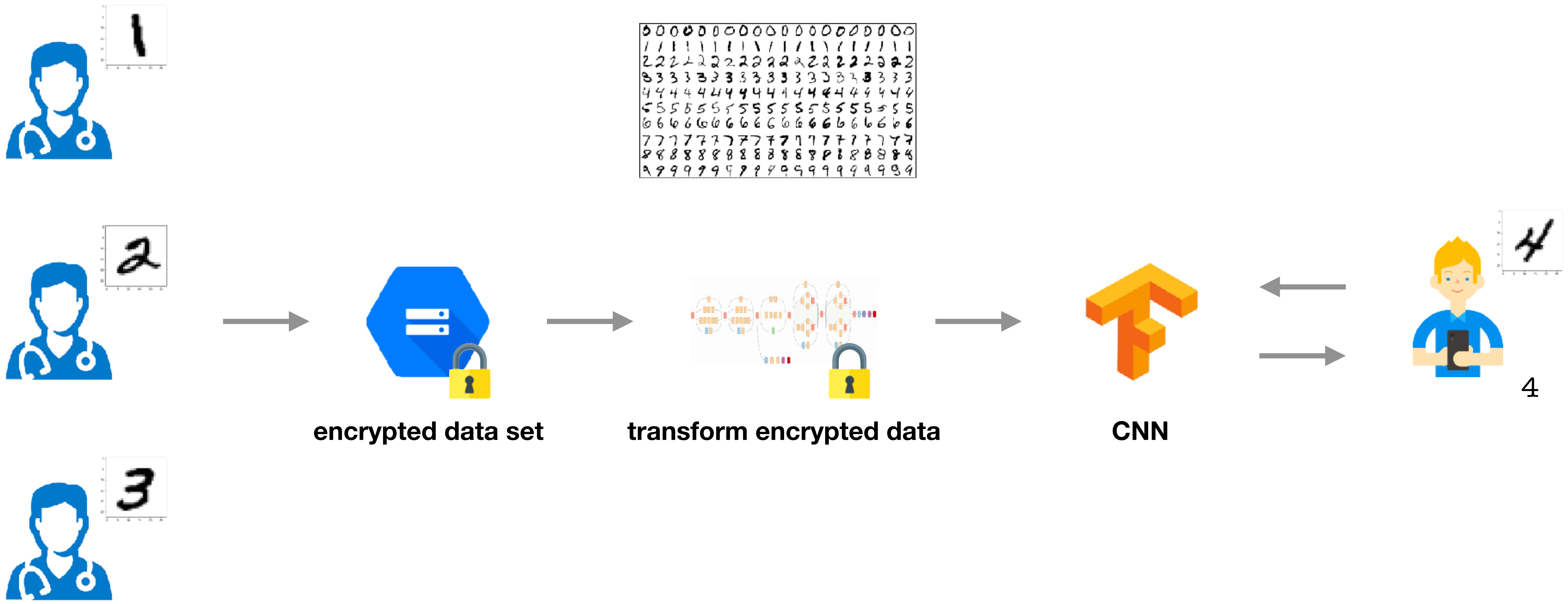
MNIST



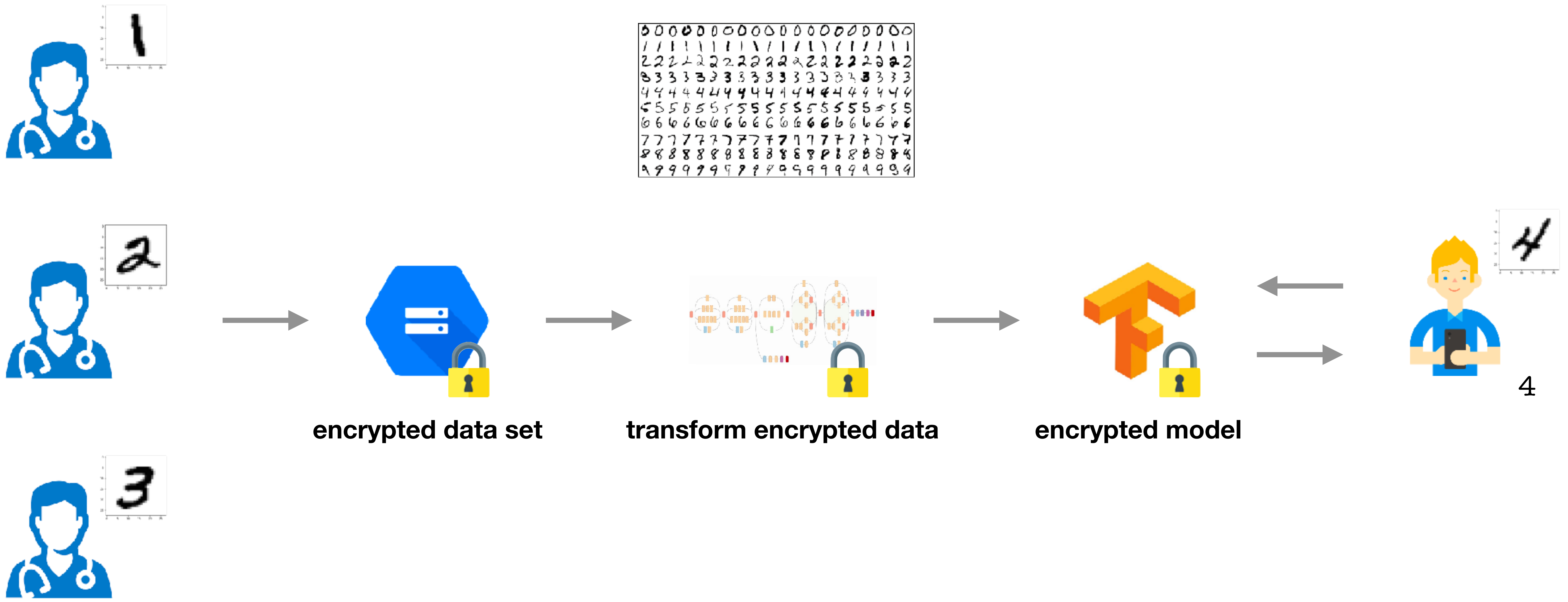
MNIST



MNIST

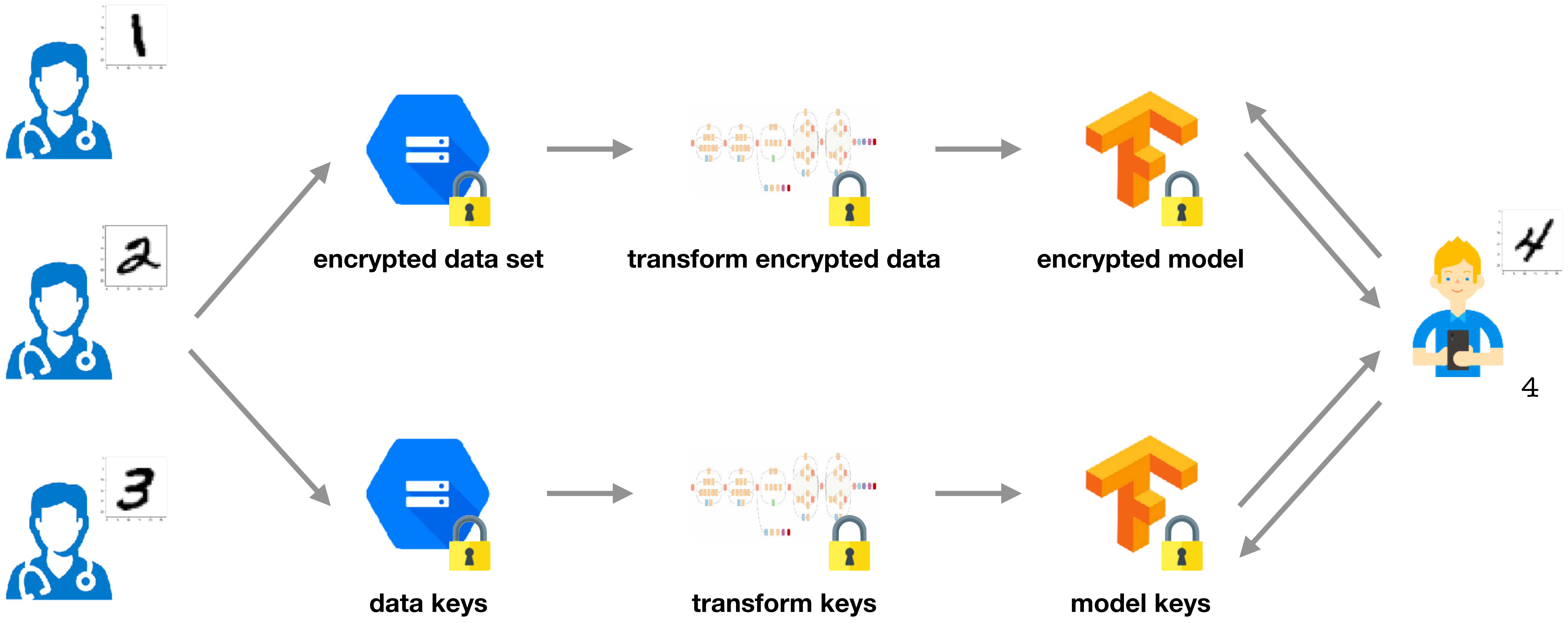


MNIST



encrypted data set
transform encrypted data
encrypted model
4

Two Servers



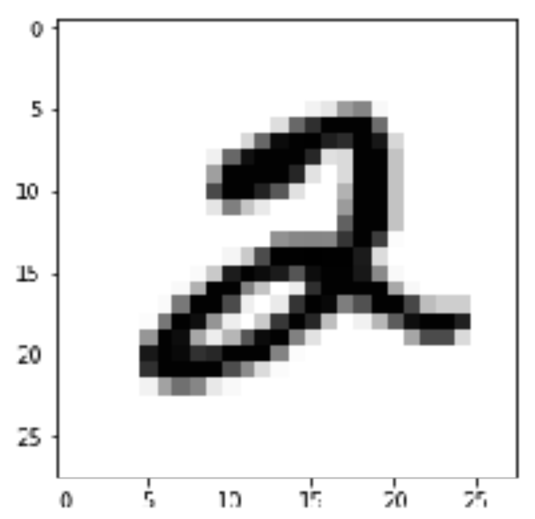
1
2
3
encrypted data set
transform encrypted data
encrypted model
data keys
transform keys
model keys
4
4

Encrypted Data

```
x = (x1 + x2) % M
```

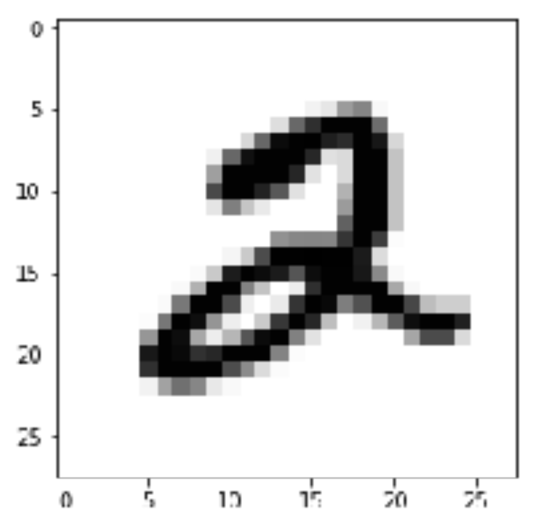
Encrypted Data

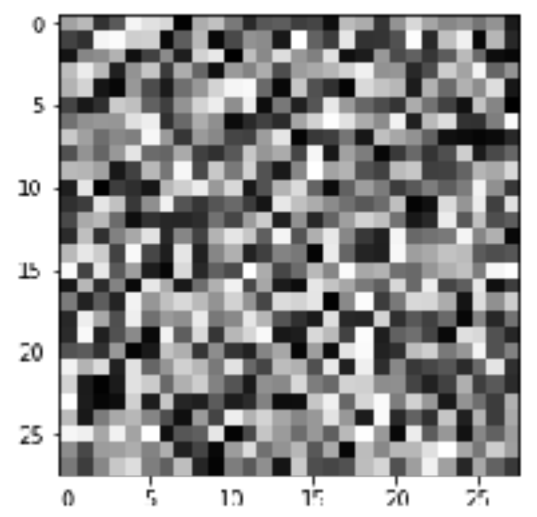
$x = (x1 + x2) \% M$



Encrypted Data

`x = (x1 + x2) % M`

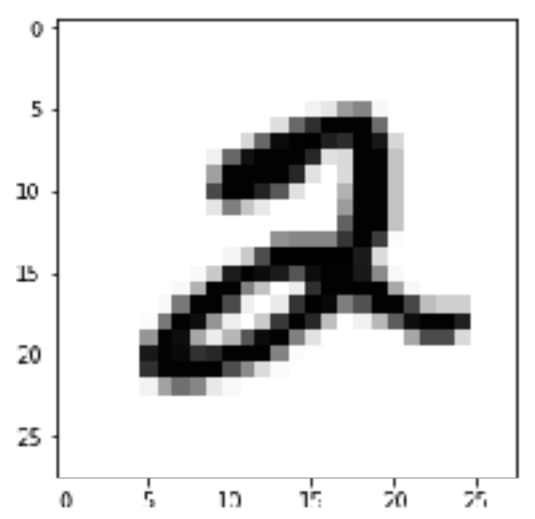




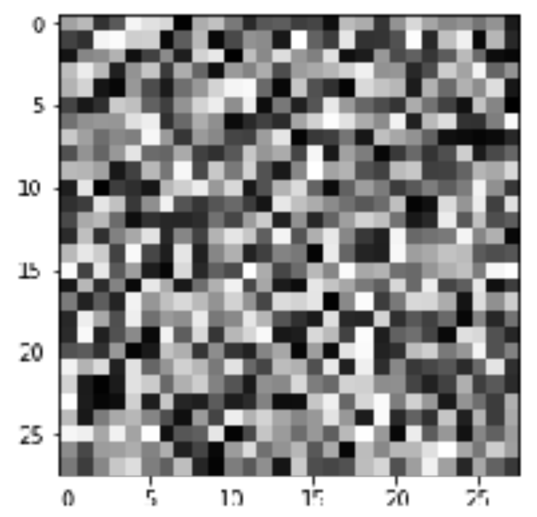
mask

Encrypted Data

x = (x1 + x2) % M

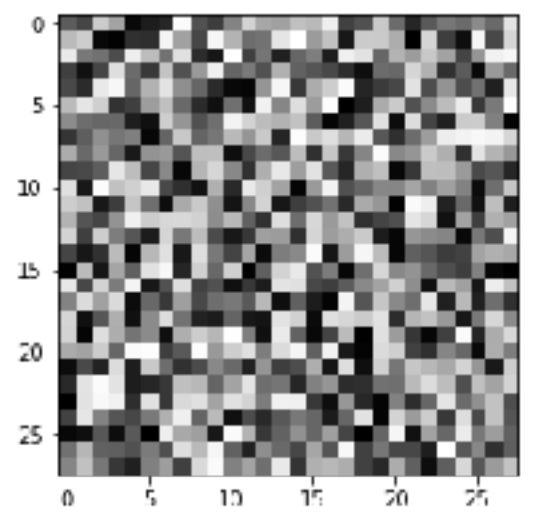


-



mask

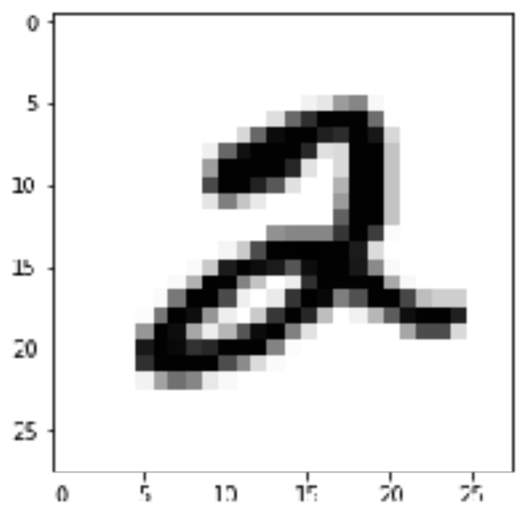
=



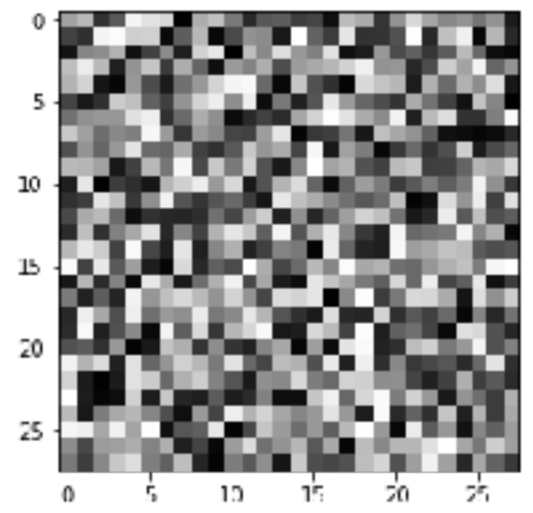
masked image

Encrypted Data

$x = (x1 + x2) \% M$

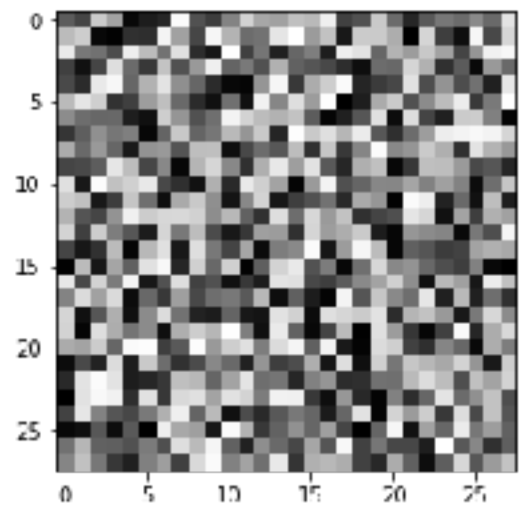


-

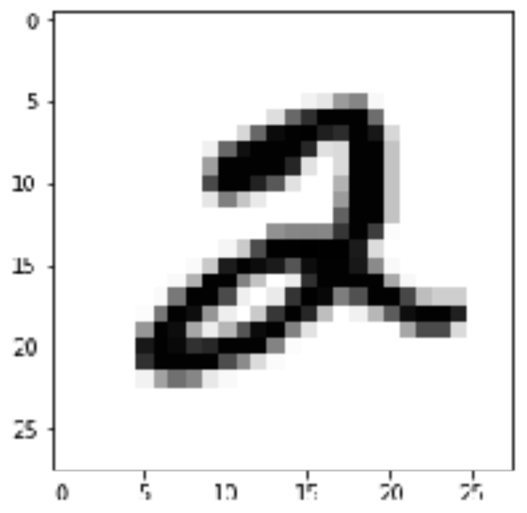


mask

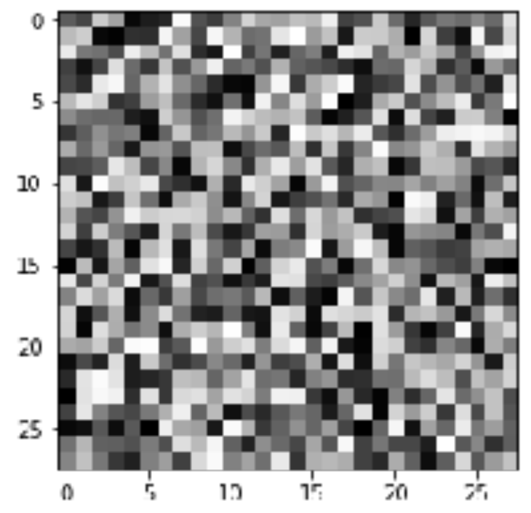
=



masked image

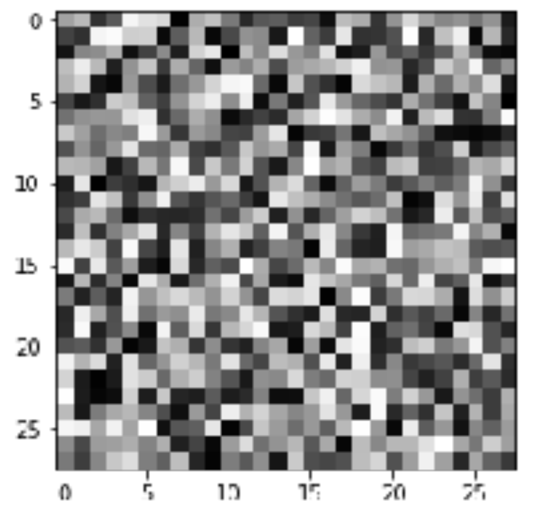


=



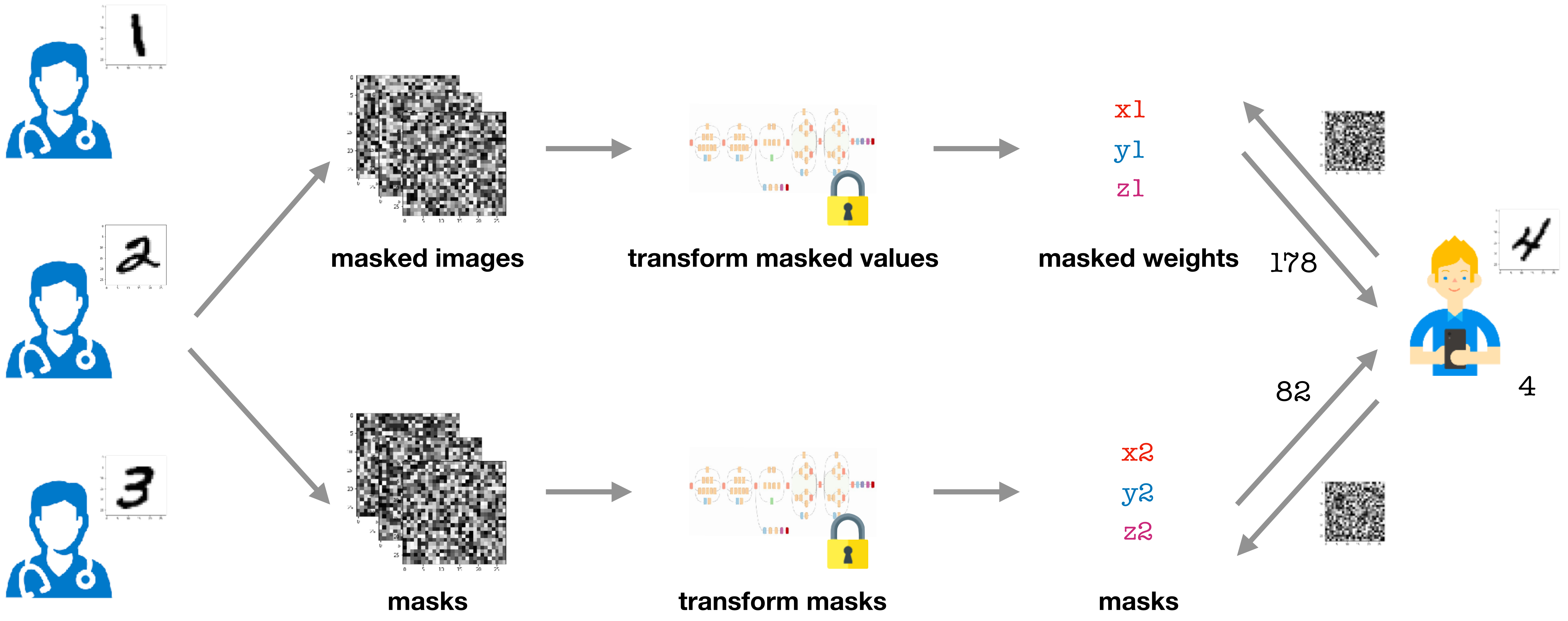
masked image

+

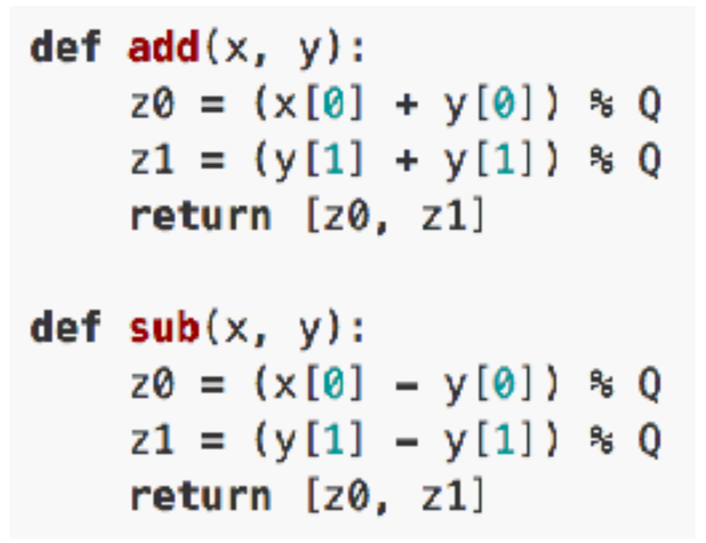


mask

Encrypted MNIST



Addition



```
def add(x, y):
    z0 = (x[0] + y[0]) % Q
    z1 = (y[1] + y[1]) % Q
    return [z0, z1]

def sub(x, y):
    z0 = (x[0] - y[0]) % Q
    z1 = (y[1] - y[1]) % Q
    return [z0, z1]
```

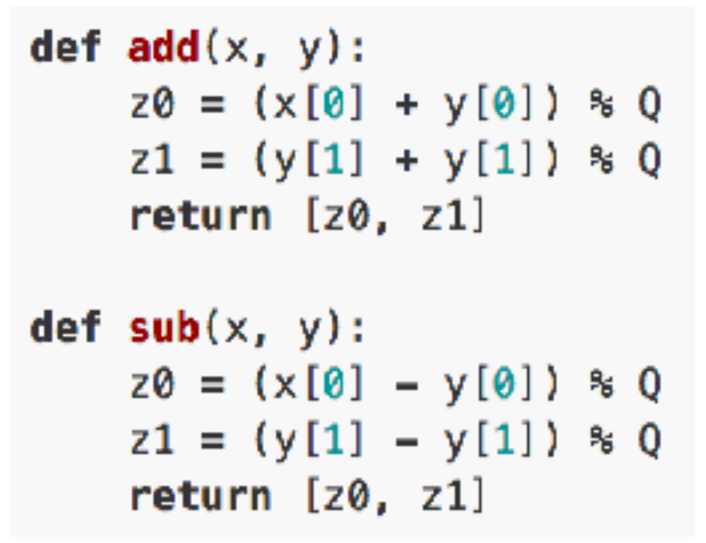
x1 y1

---

x2 y2

x = x1 + x2 y = y1 + y2

Addition



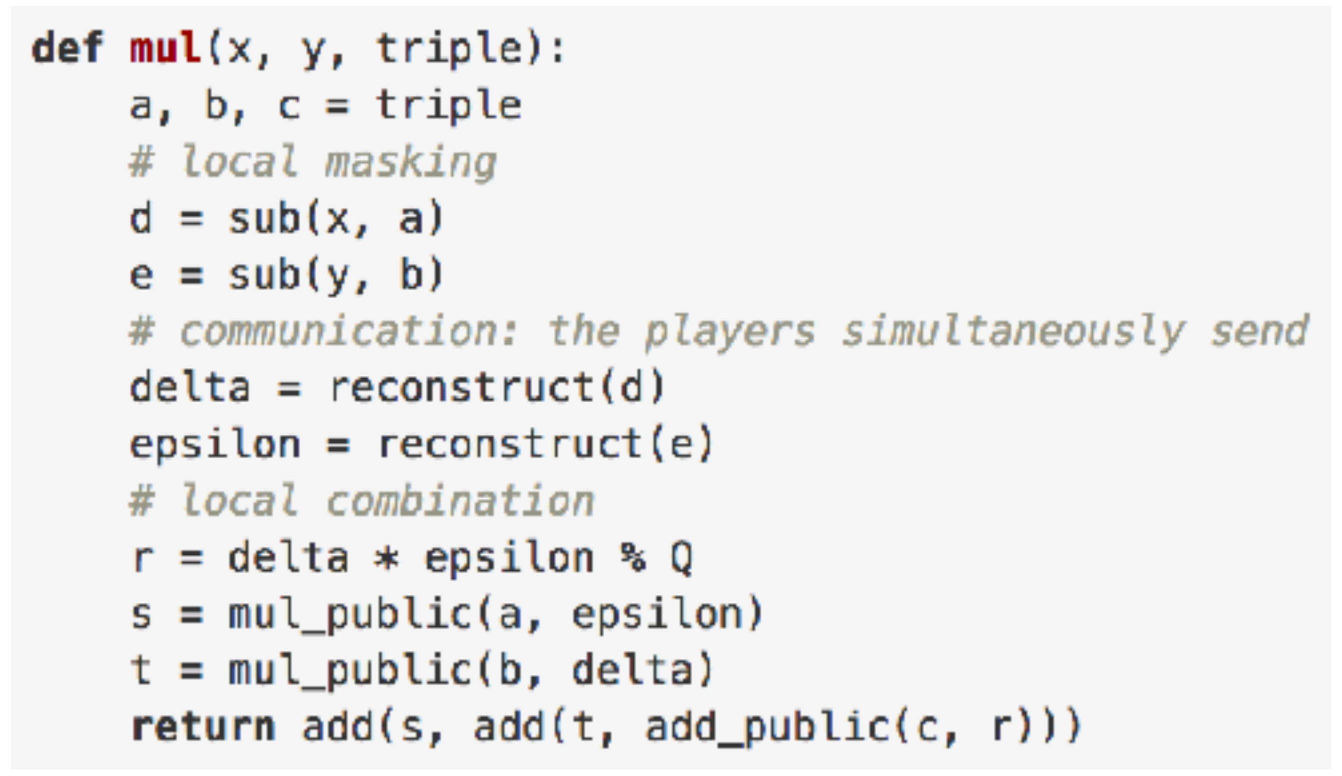
```python
def add(x, y):
    z0 = (x[0] + y[0]) % Q
    z1 = (y[1] + y[1]) % Q
    return [z0, z1]

def sub(x, y):
    z0 = (x[0] - y[0]) % Q
    z1 = (y[1] - y[1]) % Q
    return [z0, z1]
```

$x1$	$y1$	$z1 = x1 + y1$
$x2$	$y2$	$z2 = x2 + y2$

$x = x1 + x2$

$y = y1 + y2$

$$\begin{aligned} z &= z1 + z2 \\ &= (x1 + y1) + (x2 + y2) \\ &= (x1 + x2) + (y1 + y2) \\ &= x + y \end{aligned}$$



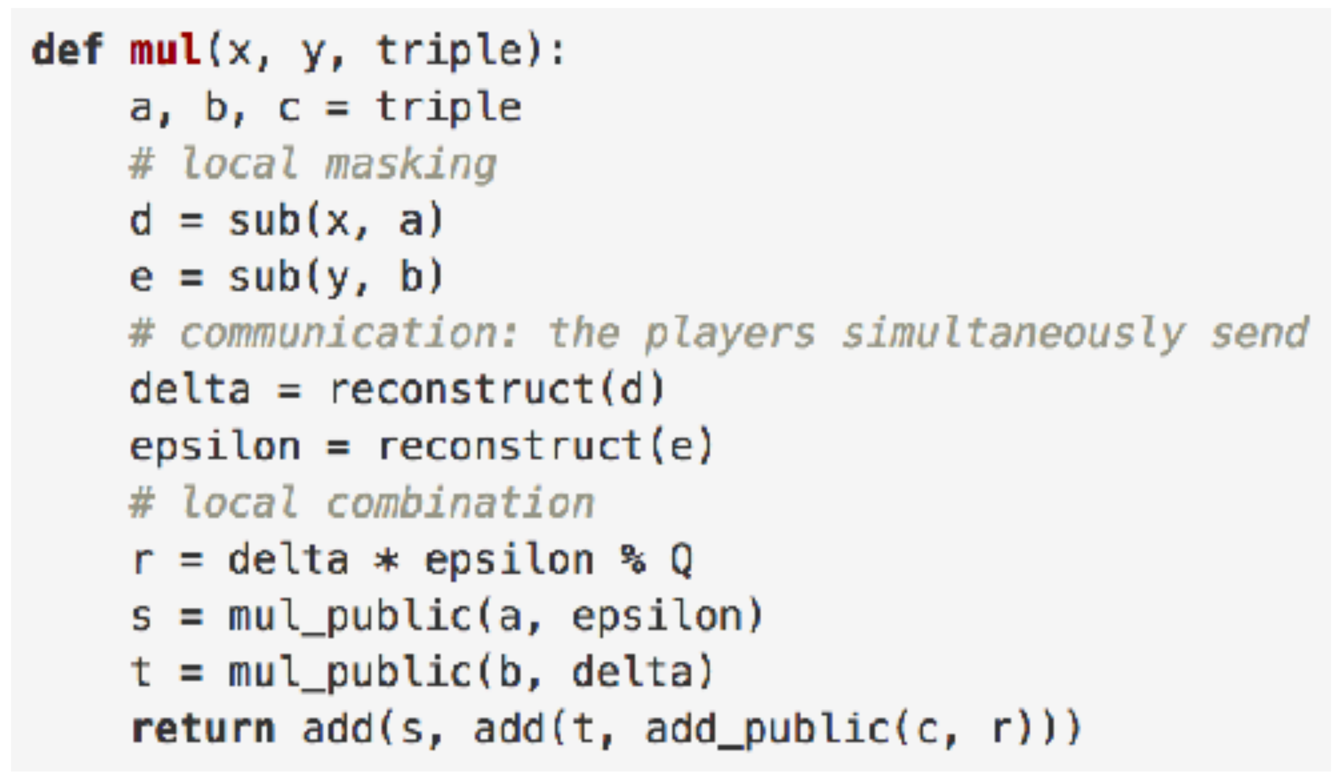
```python
def mul(x, y, triple):
    a, b, c = triple
    # local masking
    d = sub(x, a)
    e = sub(y, b)
    # communication: the players simultaneously send
    delta = reconstruct(d)
    epsilon = reconstruct(e)
    # local combination
    r = delta * epsilon % Q
    s = mul_public(a, epsilon)
    t = mul_public(b, delta)
    return add(s, add(t, add_public(c, r)))
```

Multiplication

x1 y1

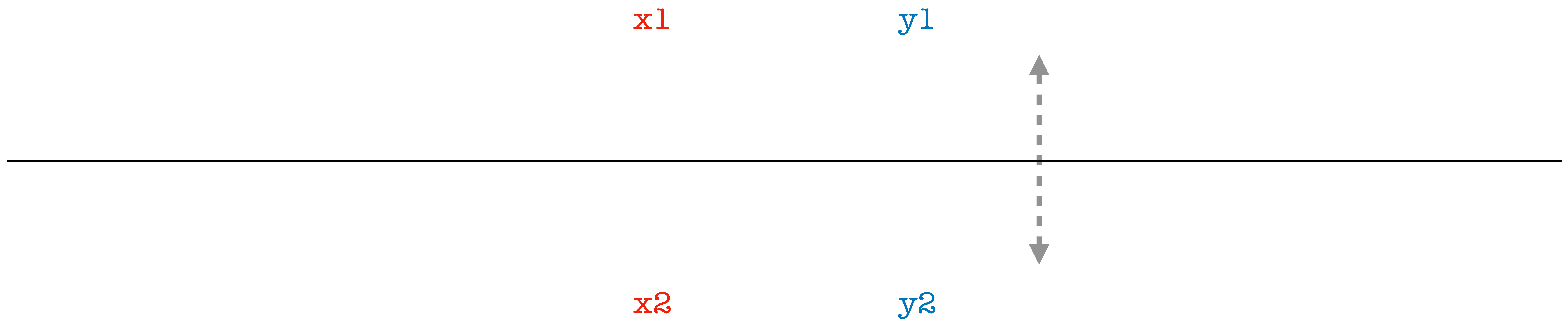
x2 y2

x = x1 + x2 y = y1 + y2



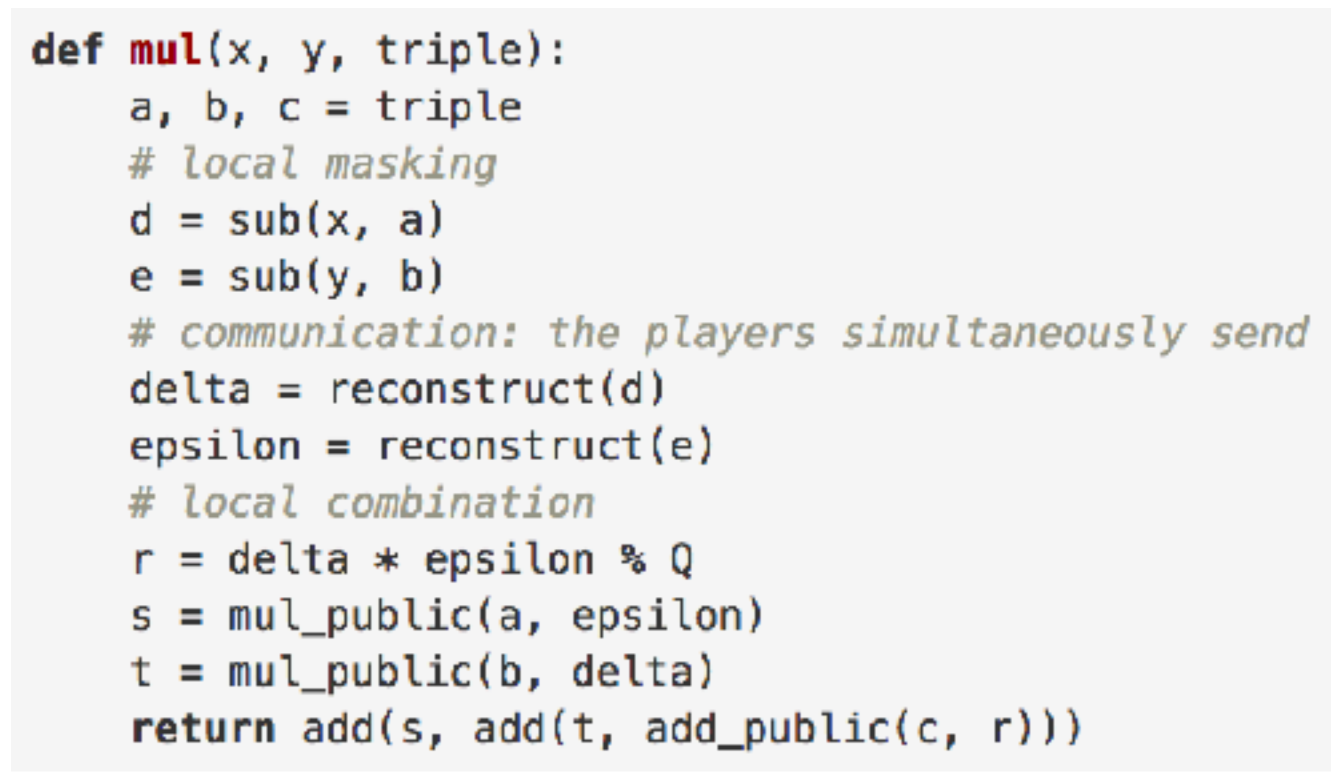
```
def mul(x, y, triple):
    a, b, c = triple
    # local masking
    d = sub(x, a)
    e = sub(y, b)
    # communication: the players simultaneously send
    delta = reconstruct(d)
    epsilon = reconstruct(e)
    # local combination
    r = delta * epsilon % Q
    s = mul_public(a, epsilon)
    t = mul_public(b, delta)
    return add(s, add(t, add_public(c, r)))
```

Multiplication

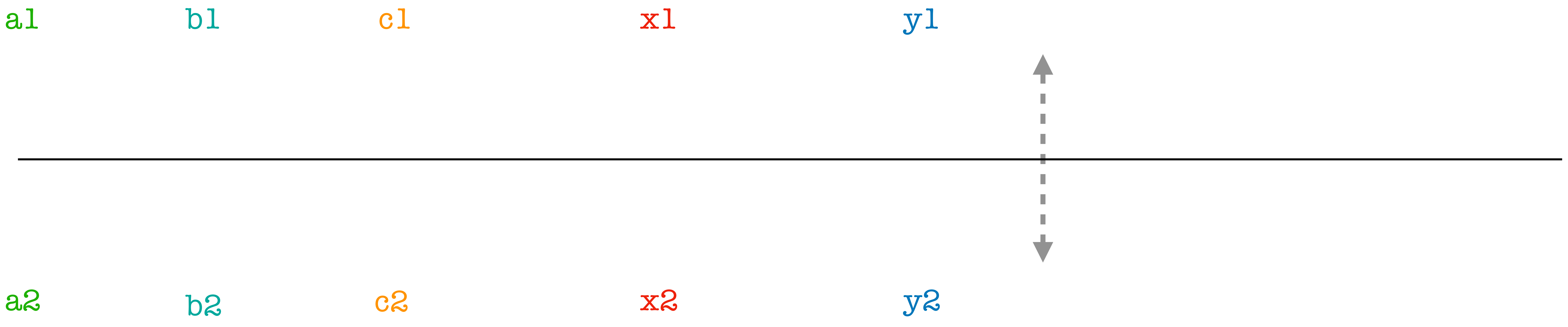


$x = x1 + x2$ $\quad$ $y = y1 + y2$

Multiplication



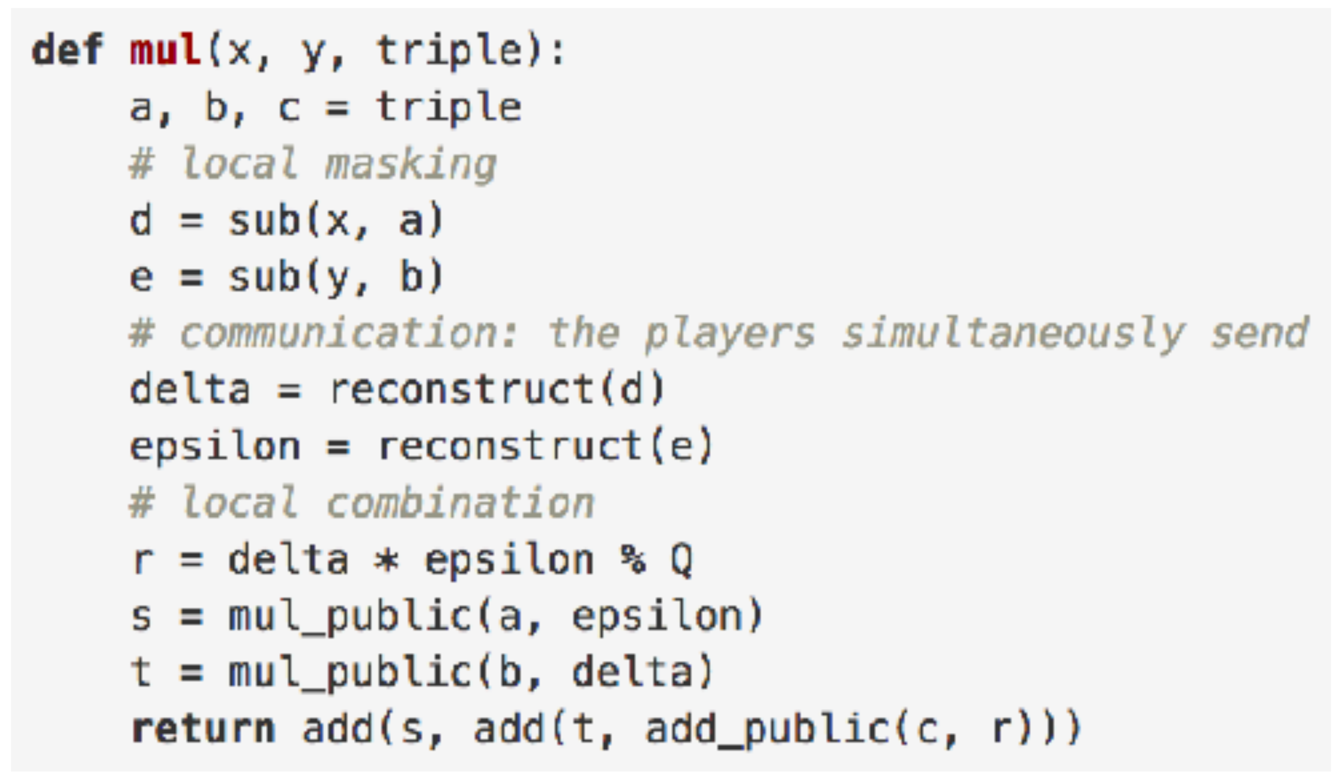
```python
def mul(x, y, triple):
    a, b, c = triple
    # local masking
    d = sub(x, a)
    e = sub(y, b)
    # communication: the players simultaneously send
    delta = reconstruct(d)
    epsilon = reconstruct(e)
    # local combination
    r = delta * epsilon % Q
    s = mul_public(a, epsilon)
    t = mul_public(b, delta)
    return add(s, add(t, add_public(c, r)))
```



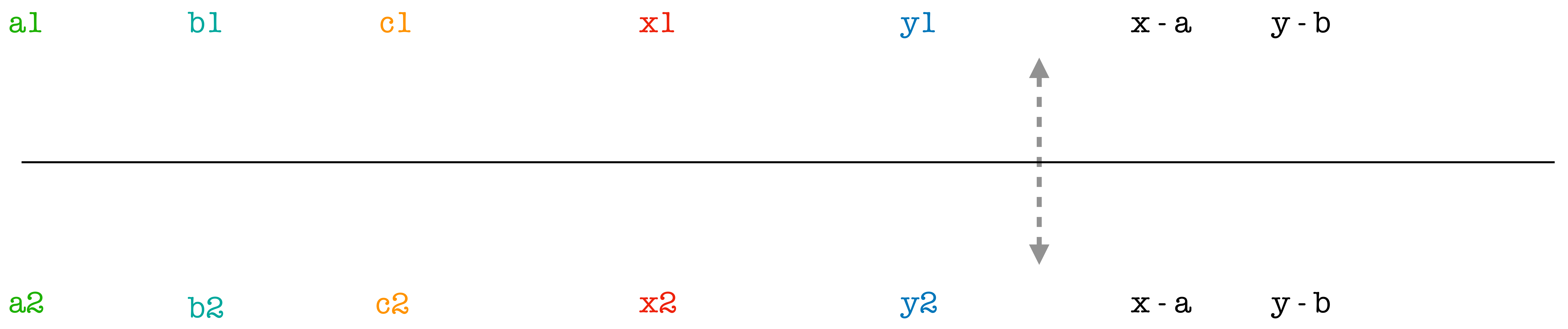
$a = a1 + a2$ $b = b1 + b2$ $c = c1 + c2$
$= a * b$

$x = x1 + x2$ $y = y1 + y2$

Multiplication



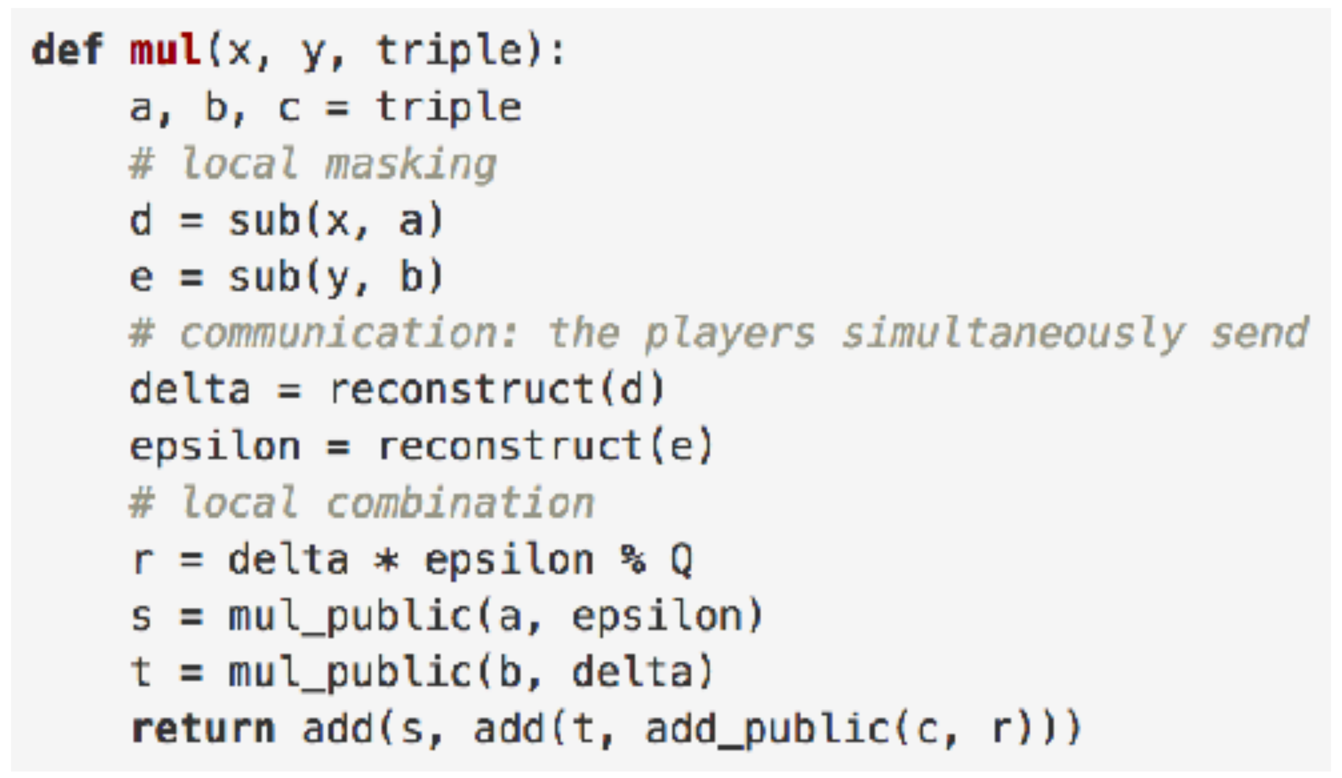
```
def mul(x, y, triple):
    a, b, c = triple
    # local masking
    d = sub(x, a)
    e = sub(y, b)
    # communication: the players simultaneously send
    delta = reconstruct(d)
    epsilon = reconstruct(e)
    # local combination
    r = delta * epsilon % Q
    s = mul_public(a, epsilon)
    t = mul_public(b, delta)
    return add(s, add(t, add_public(c, r)))
```



a = a1 + a2 b = b1 + b2 c = c1 + c2
= a * b

x = x1 + x2 y = y1 + y2

Multiplication



```
def mul(x, y, triple):
    a, b, c = triple
    # local masking
    d = sub(x, a)
    e = sub(y, b)
    # communication: the players simultaneously send
    delta = reconstruct(d)
    epsilon = reconstruct(e)
    # local combination
    r = delta * epsilon % Q
    s = mul_public(a, epsilon)
    t = mul_public(b, delta)
    return add(s, add(t, add_public(c, r)))
```

$a1$ $b1$ $c1$ $x1$ $y1$ $x - a$ $y - b$

$$z1 = (x - a)*(y - b) + (x - a)*b1 + (y - b)*a1 + c1$$

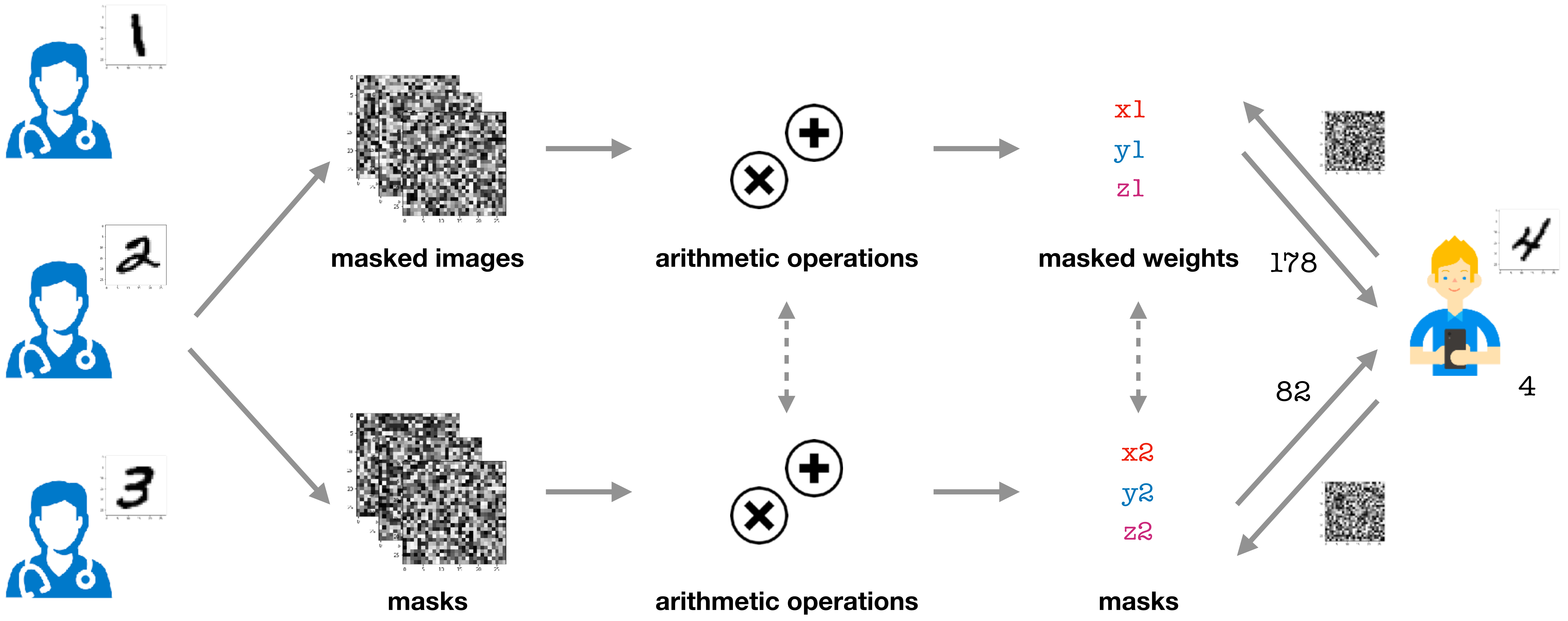
$a2$ $b2$ $c2$ $x2$ $y2$ $x - a$ $y - b$

$$z2 = (x - a)*(y - b) + (x - a)*b2 + (y - b)*a2 + c2$$

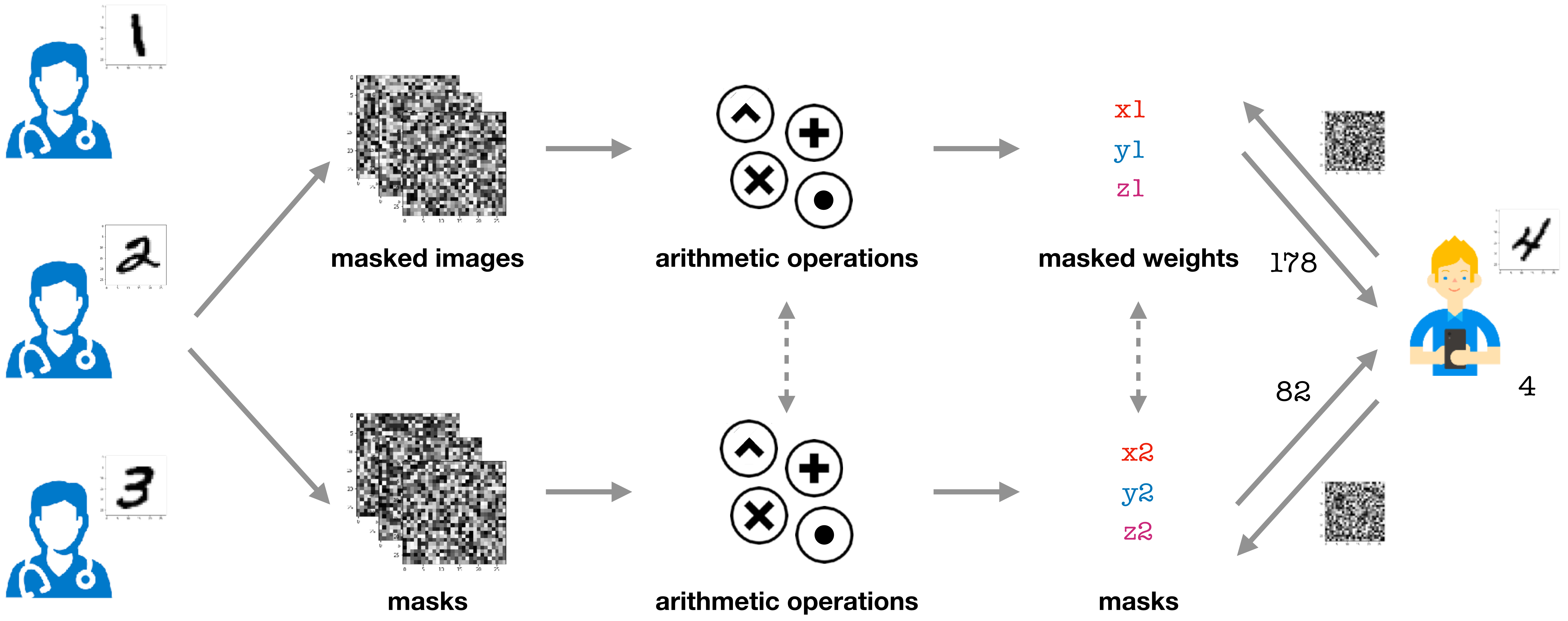
$a = a1 + a2$ $b = b1 + b2$ $c = c1 + c2 = a * b$ $x = x1 + x2$ $y = y1 + y2$

$z = z1 + z2 = \ldots = x * y$

Encrypted MNIST

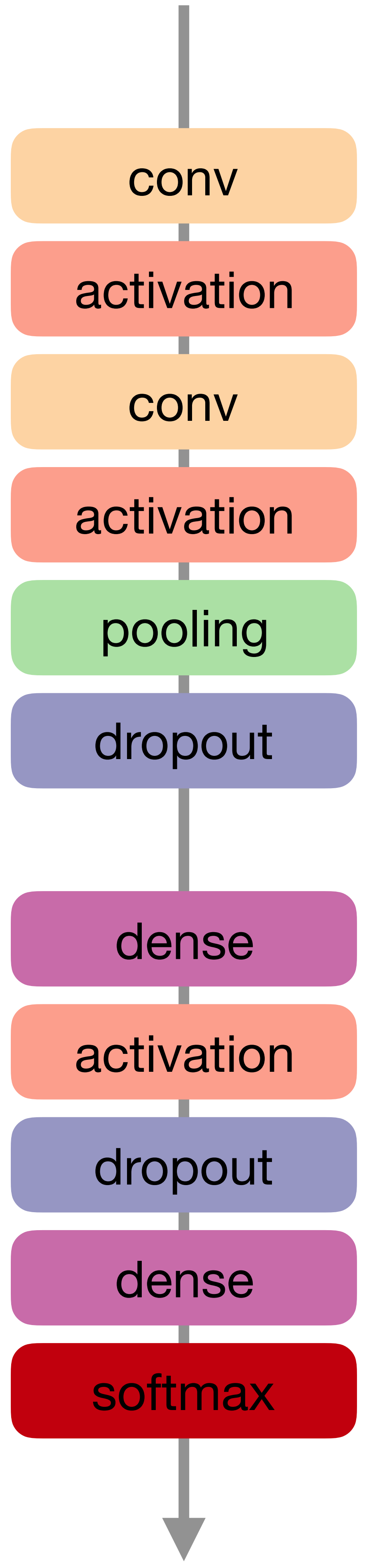


Encrypted MNIST

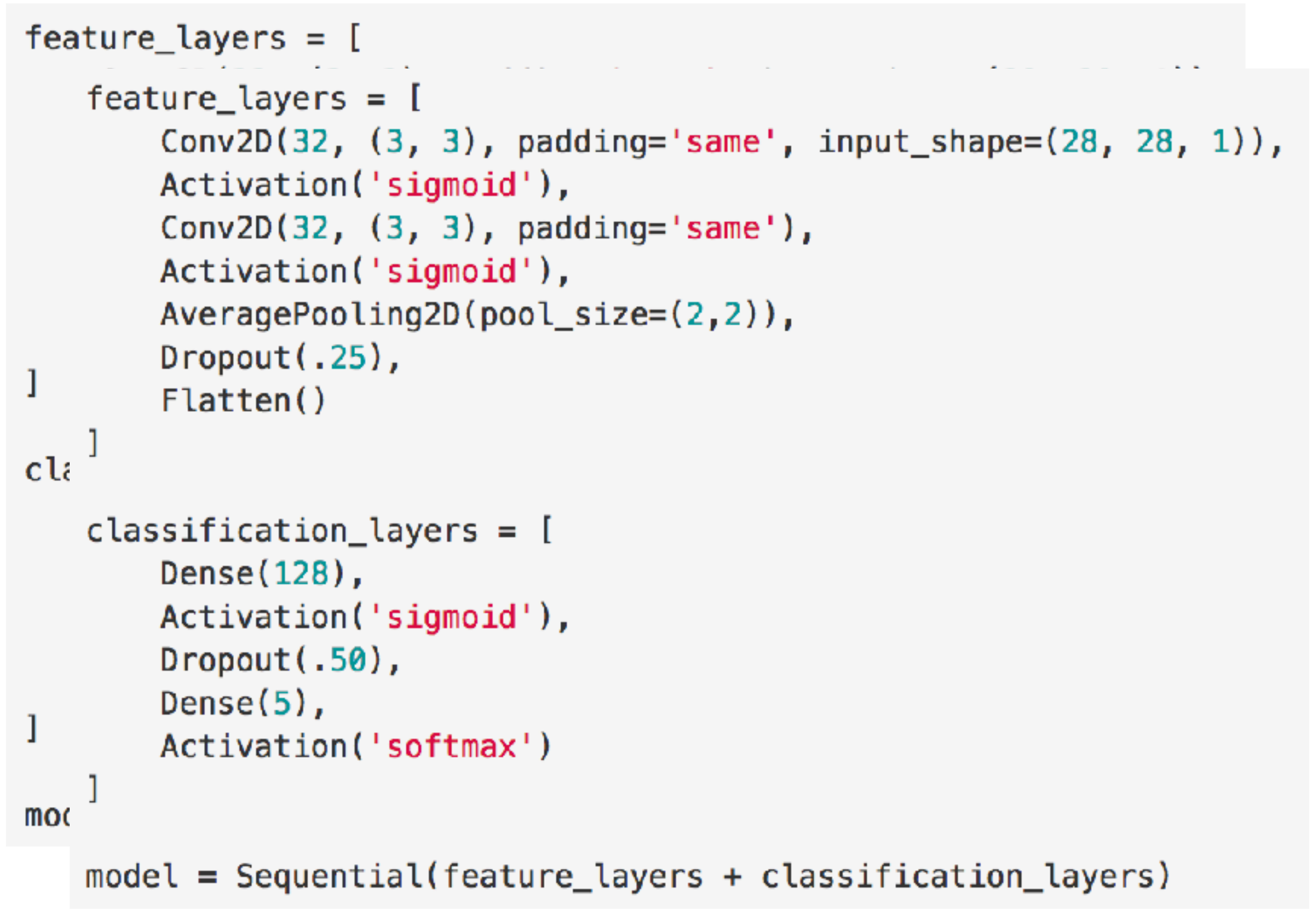


Encrypted Prediction

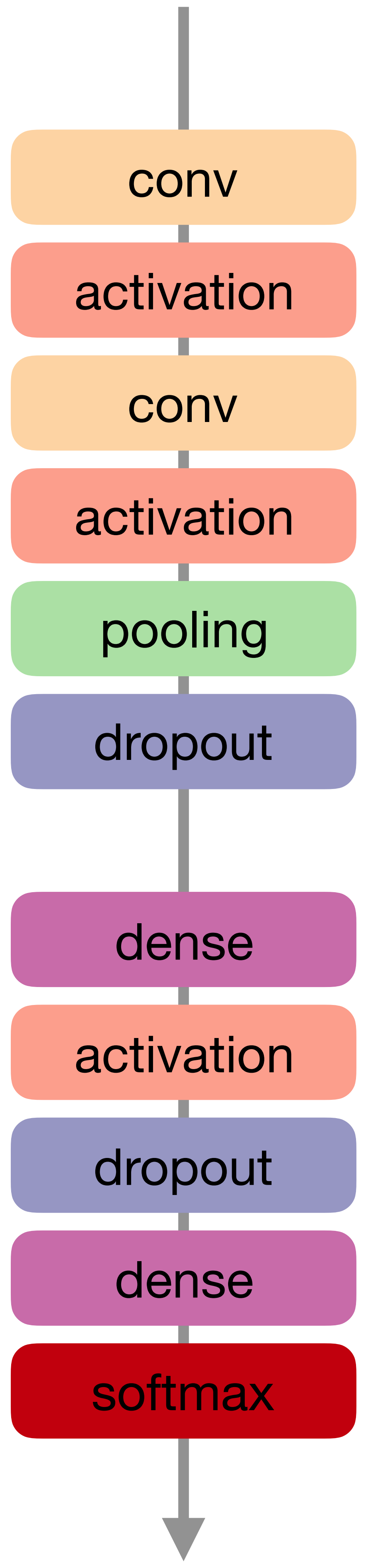
```
feature_layers = [
    Conv2D(32, (3, 3), padding='same', input_shape=(28, 28, 1)),
    Activation('relu'),
    Conv2D(32, (3, 3), padding='same'),
    Activation('relu'),
    MaxPooling2D(pool_size=(2,2)),
    Dropout(.25),
    Flatten()
]

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

model = Sequential(feature_layers + classification_layers)
```



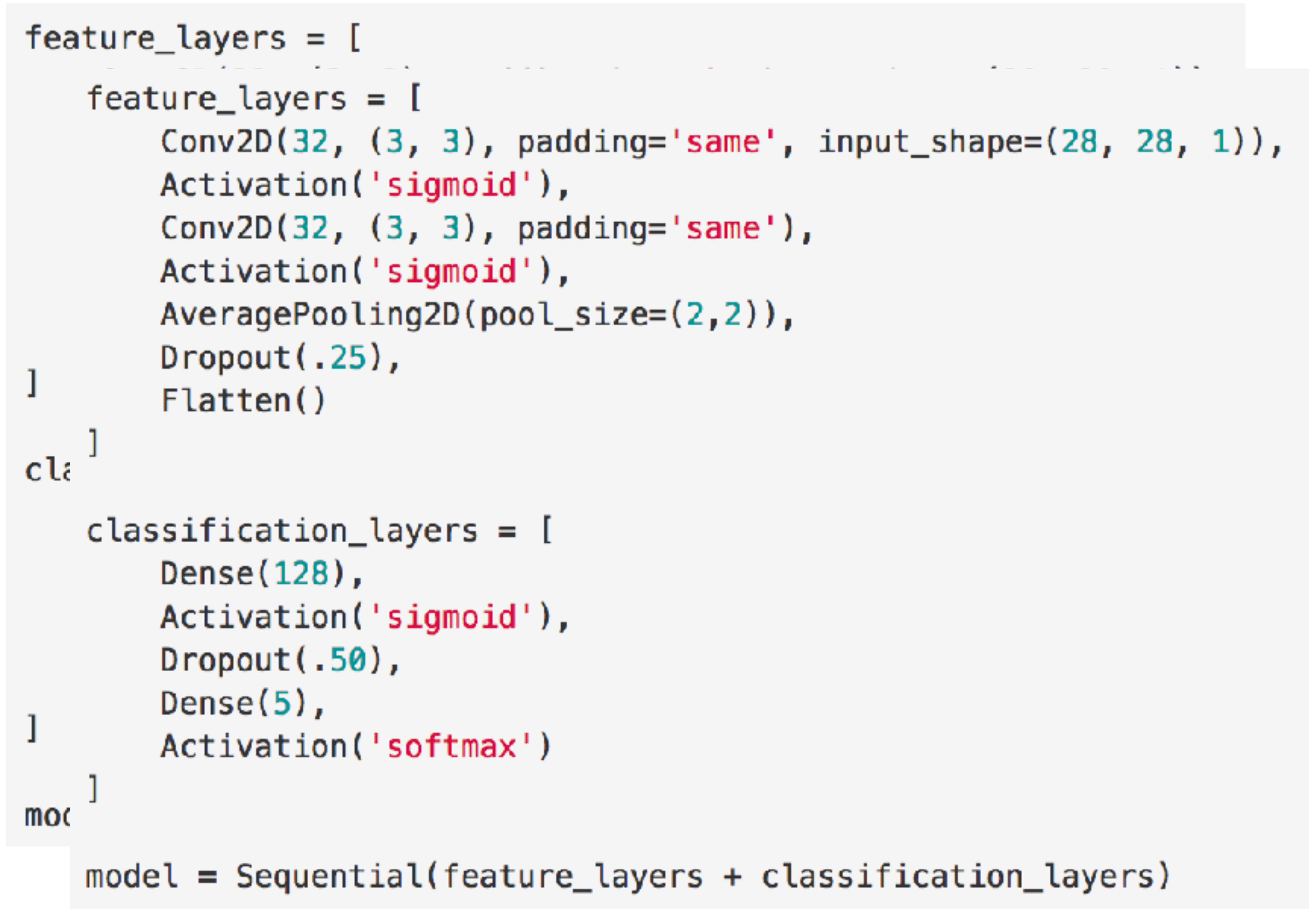
Encrypted Prediction



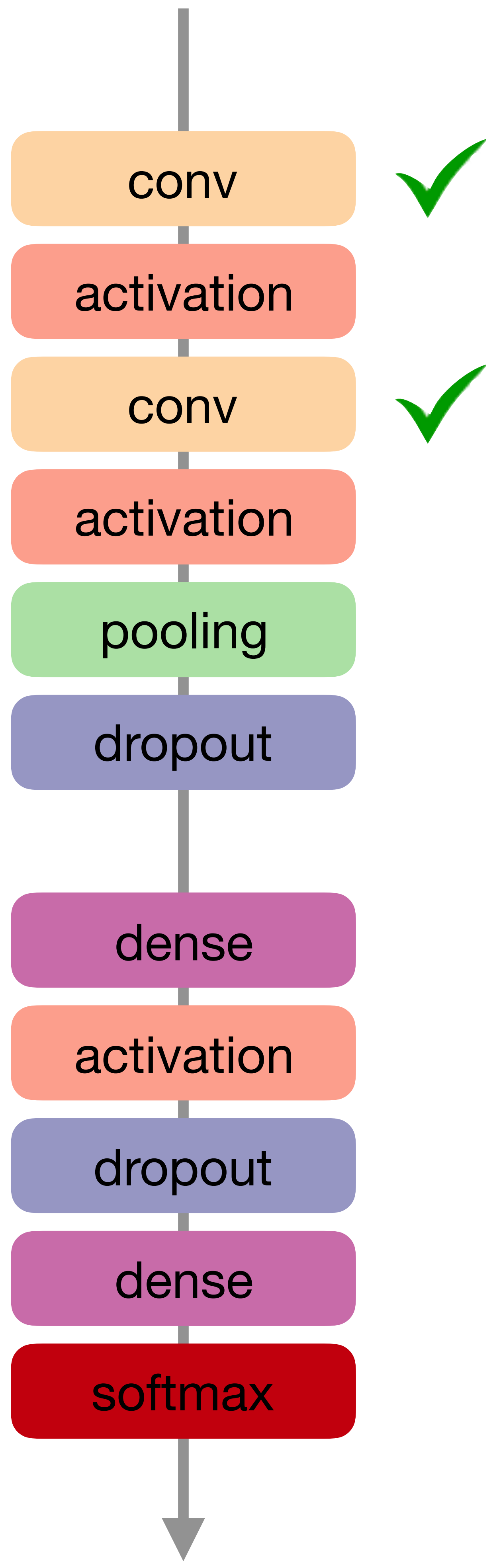
```
feature_layers = [
    Conv2D(32, (3, 3), padding='same', input_shape=(28, 28, 1)),
    Activation('sigmoid'),
    Conv2D(32, (3, 3), padding='same'),
    Activation('sigmoid'),
    AveragePooling2D(pool_size=(2,2)),
    Dropout(.25),
    Flatten()
]

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

model = Sequential(feature_layers + classification_layers)
```



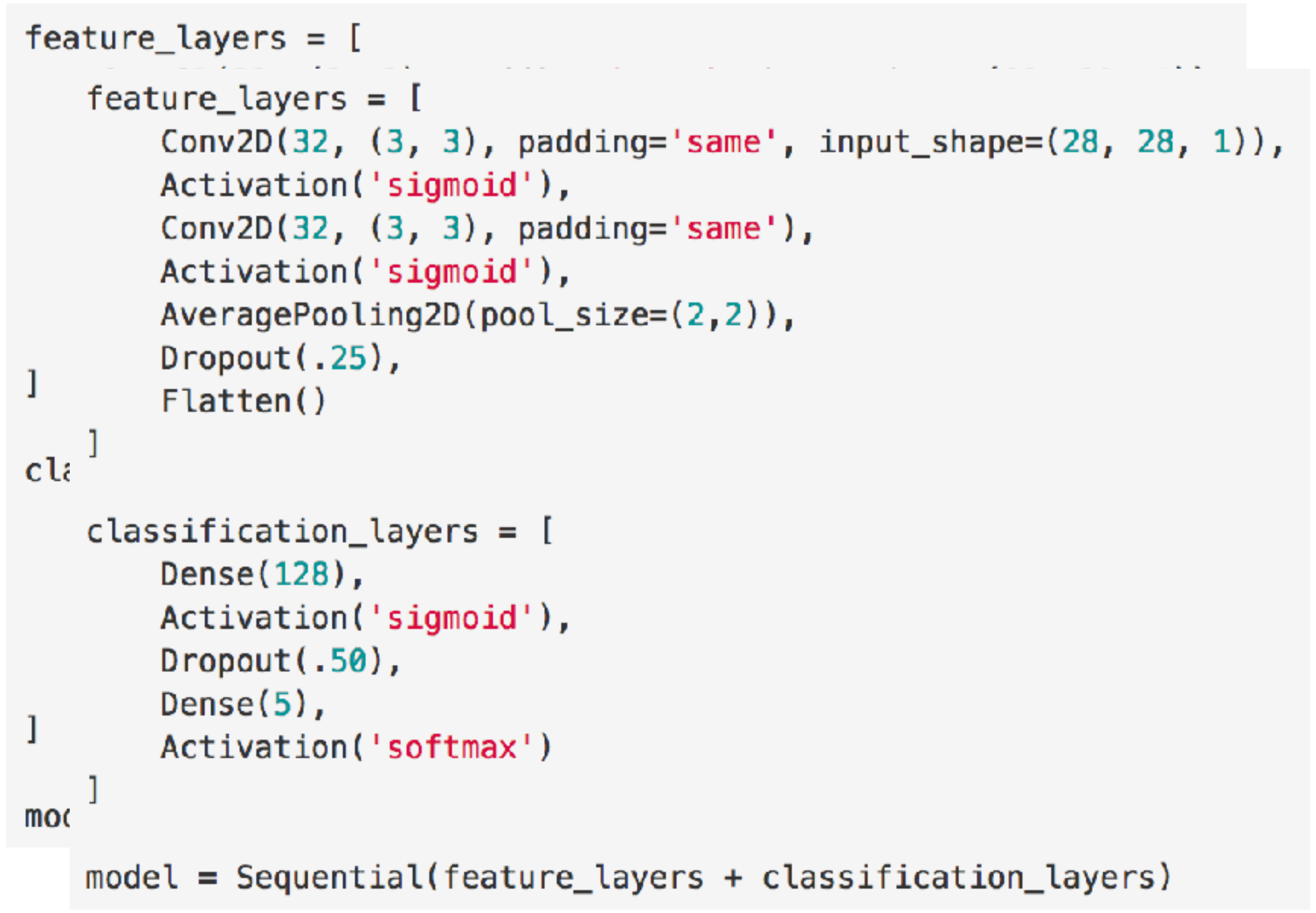
Encrypted Prediction



```
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)
```

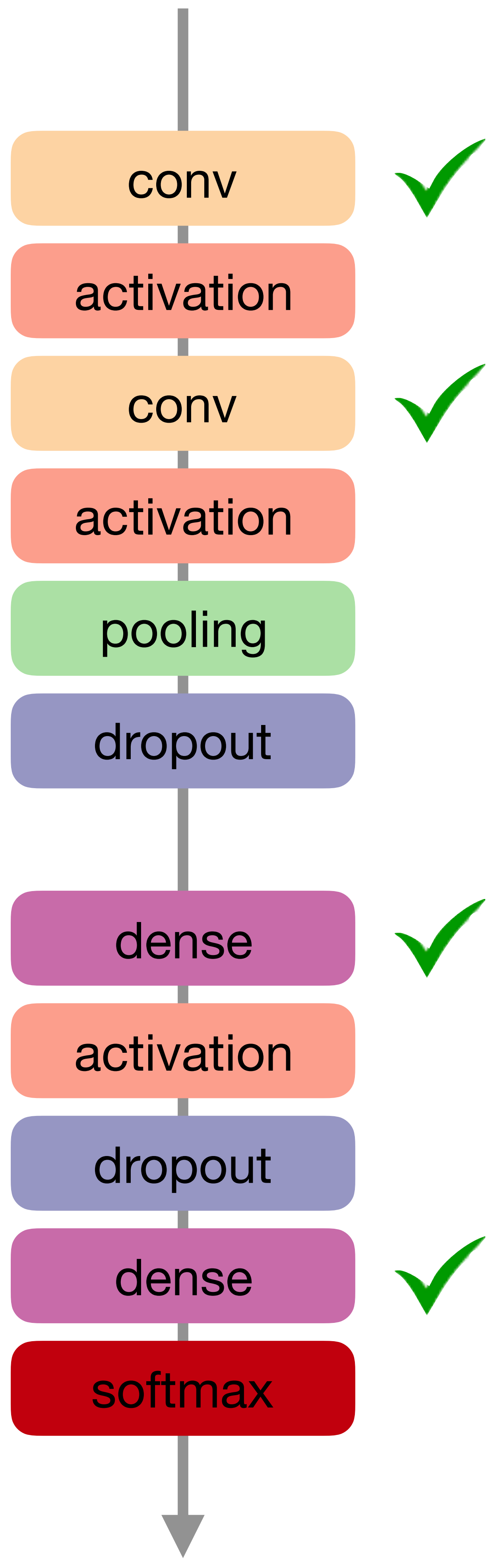


Encrypted Prediction

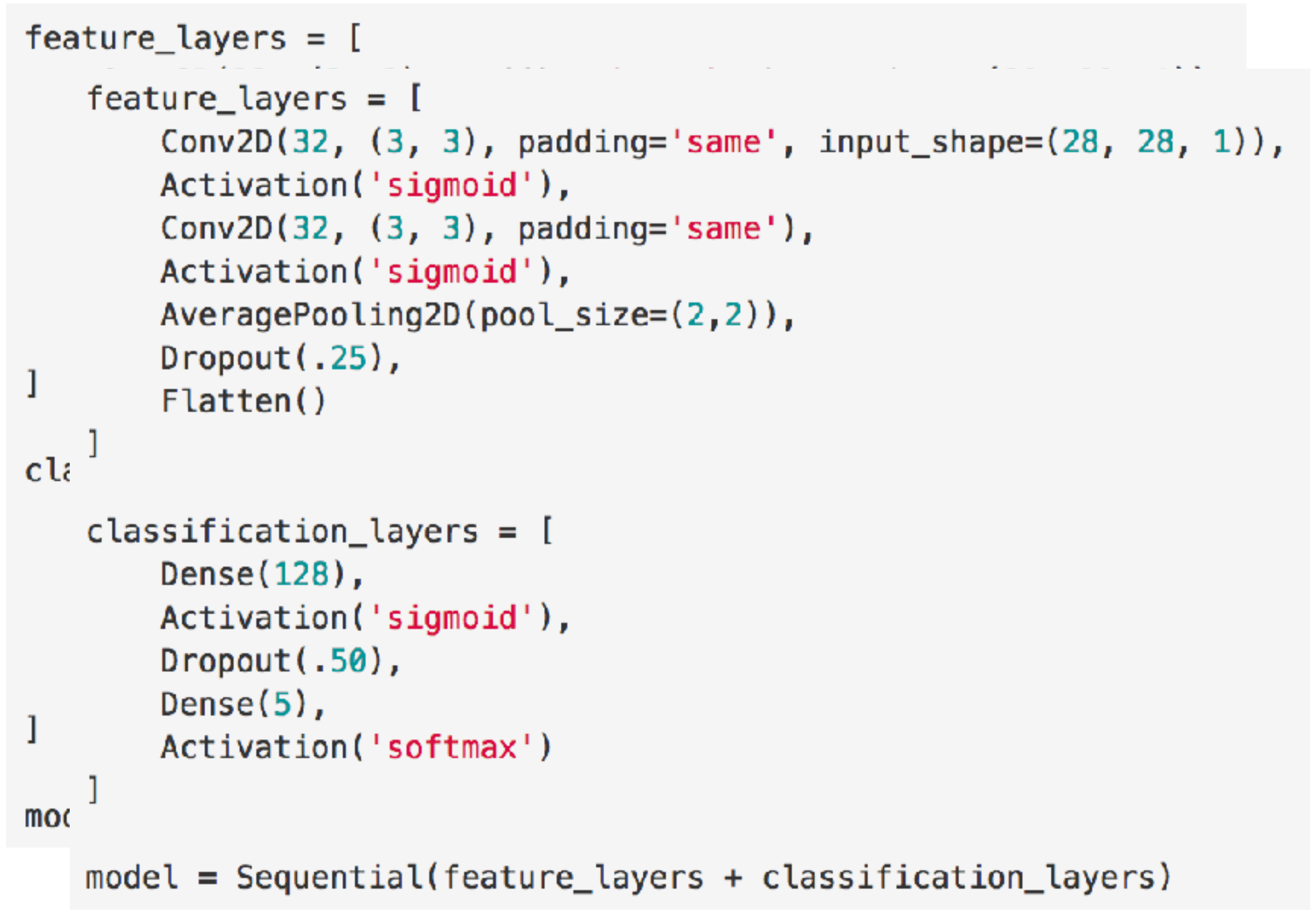


```
feature_layers = [
]
cl
]
mo
```

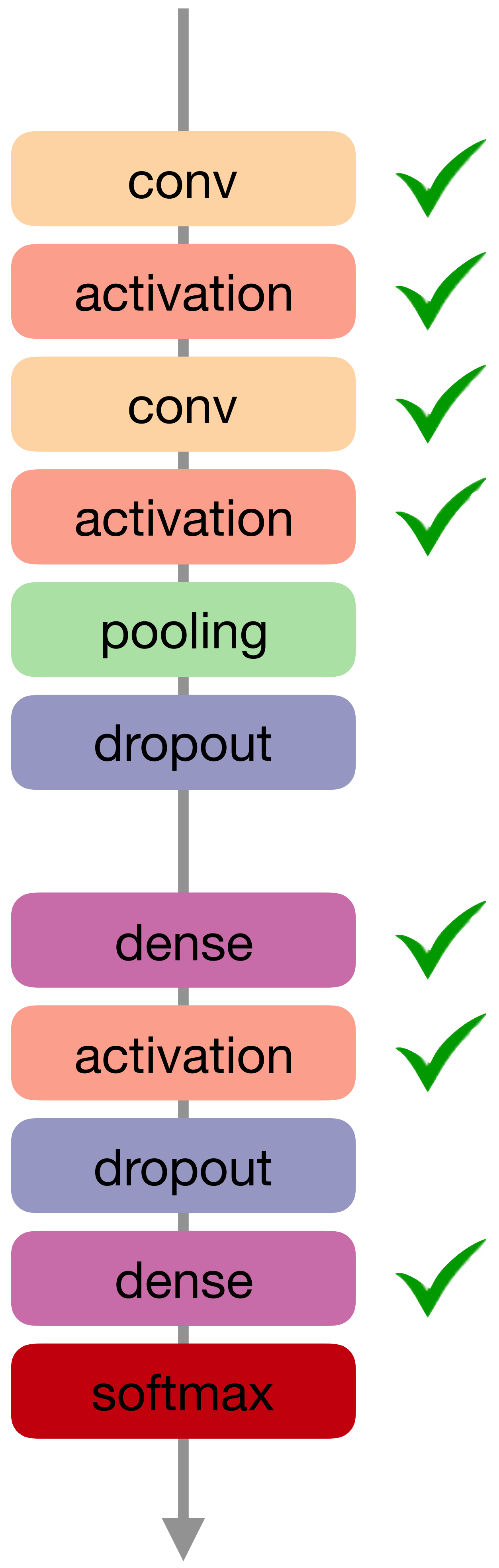
```
feature_layers = [
    Conv2D(32, (3, 3), padding='same', input_shape=(28, 28, 1)),
    Activation('sigmoid'),
    Conv2D(32, (3, 3), padding='same'),
    Activation('sigmoid'),
    AveragePooling2D(pool_size=(2,2)),
    Dropout(.25),
    Flatten()
]

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

model = Sequential(feature_layers + classification_layers)
```



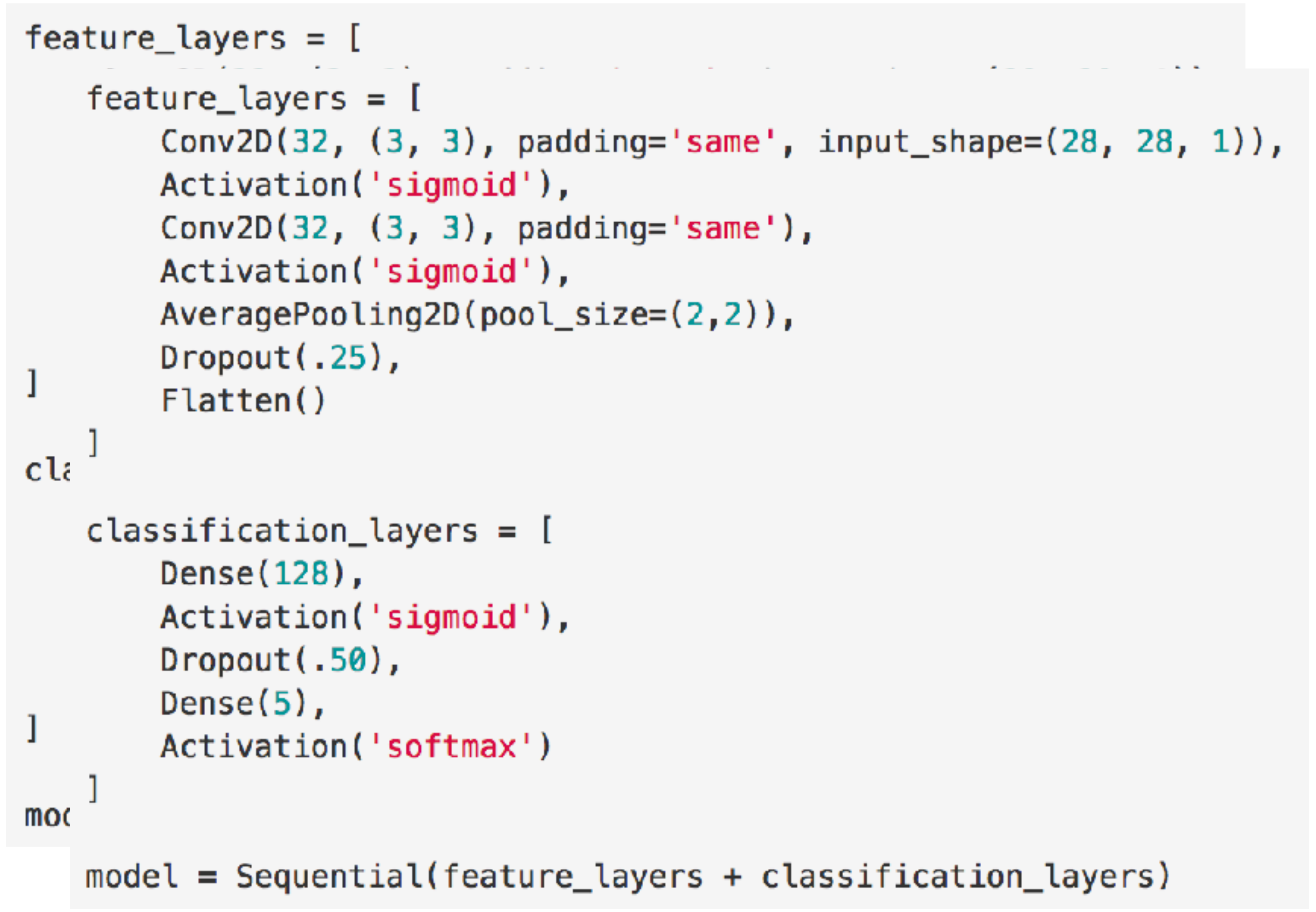
Encrypted Prediction



```
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)
```

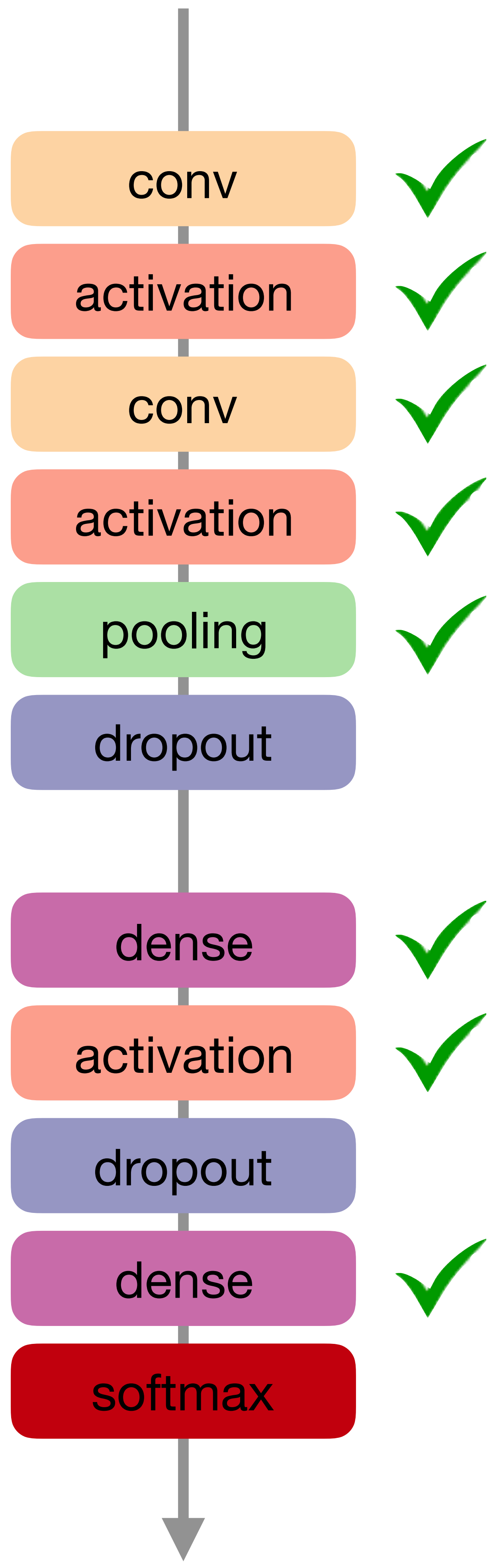


Encrypted Prediction

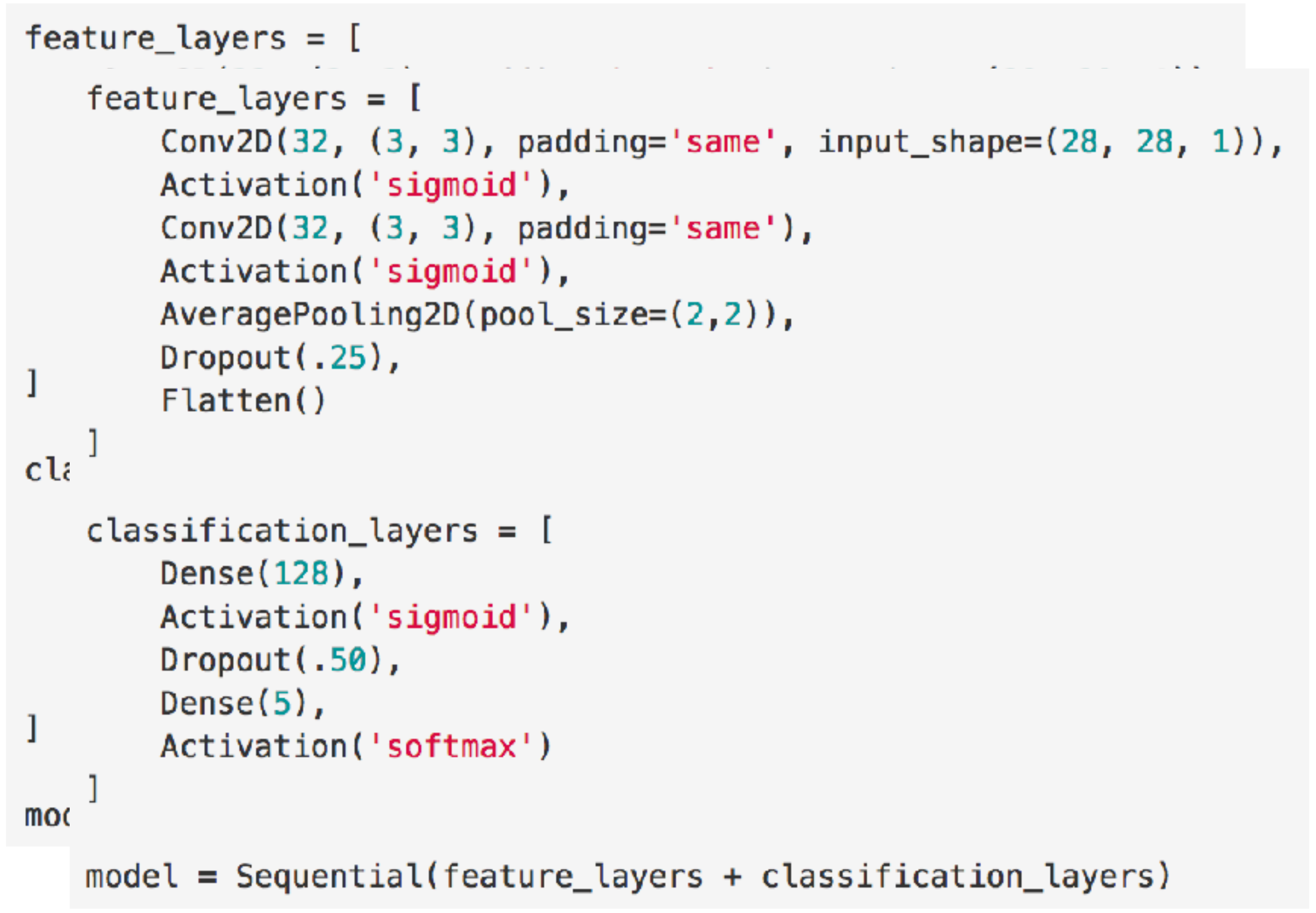


```
feature_layers = [
]
cla
]
moc
```

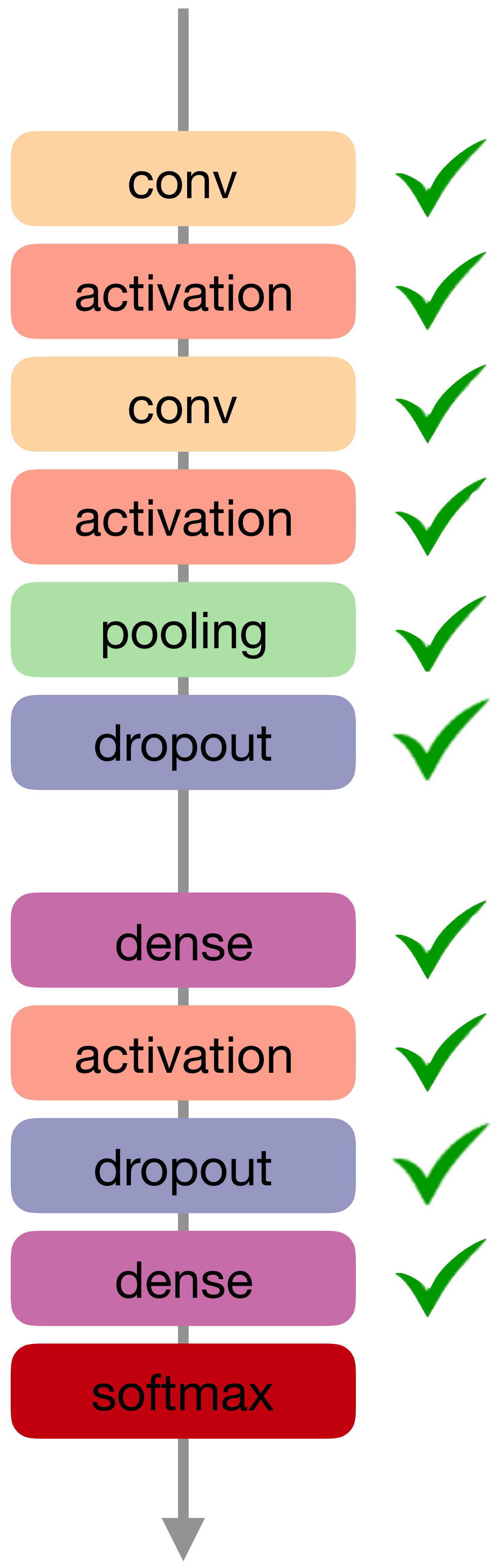
```
feature_layers = [
    Conv2D(32, (3, 3), padding='same', input_shape=(28, 28, 1)),
    Activation('sigmoid'),
    Conv2D(32, (3, 3), padding='same'),
    Activation('sigmoid'),
    AveragePooling2D(pool_size=(2,2)),
    Dropout(.25),
    Flatten()
]

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

model = Sequential(feature_layers + classification_layers)
```



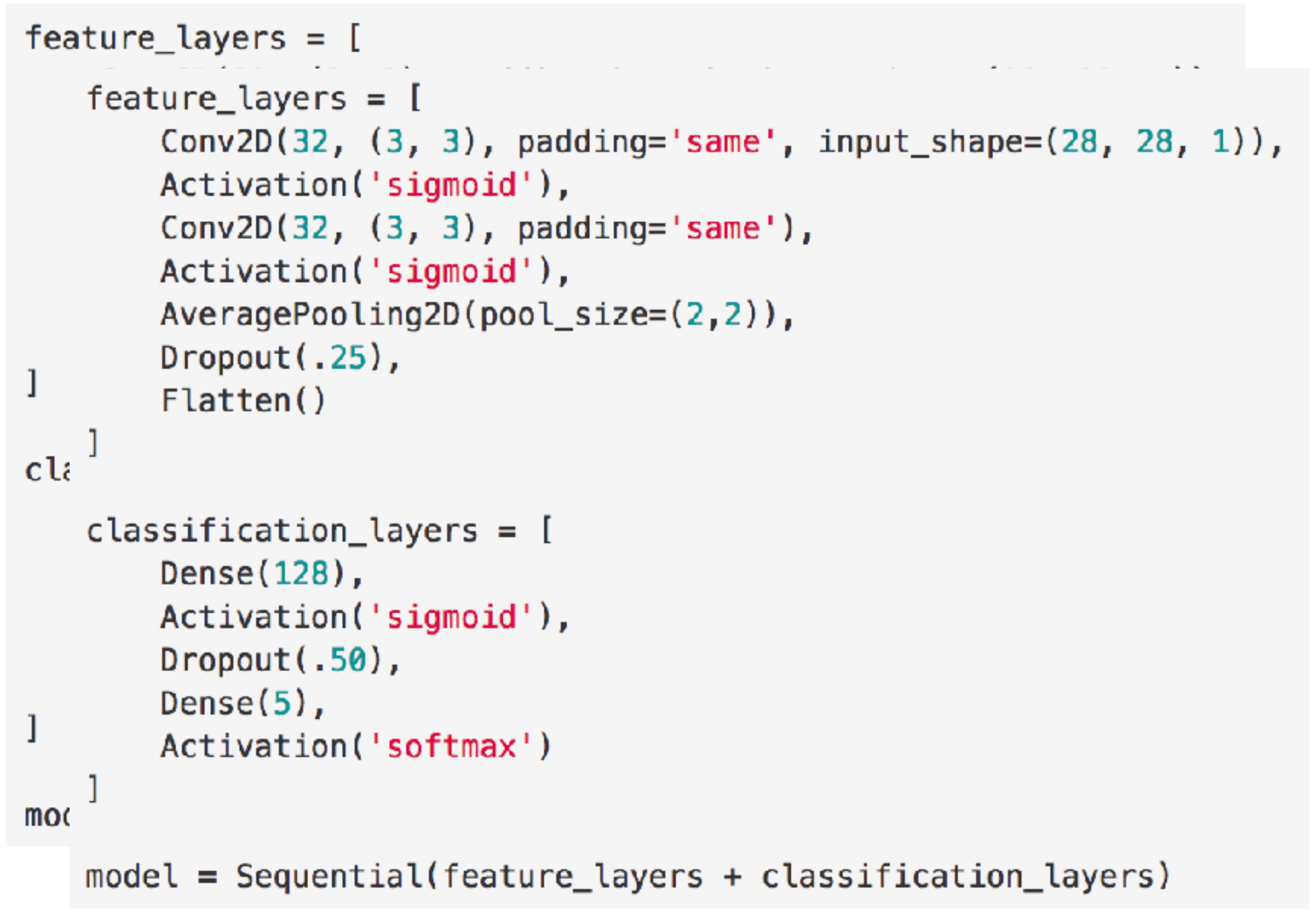
Encrypted Prediction



```
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)
```

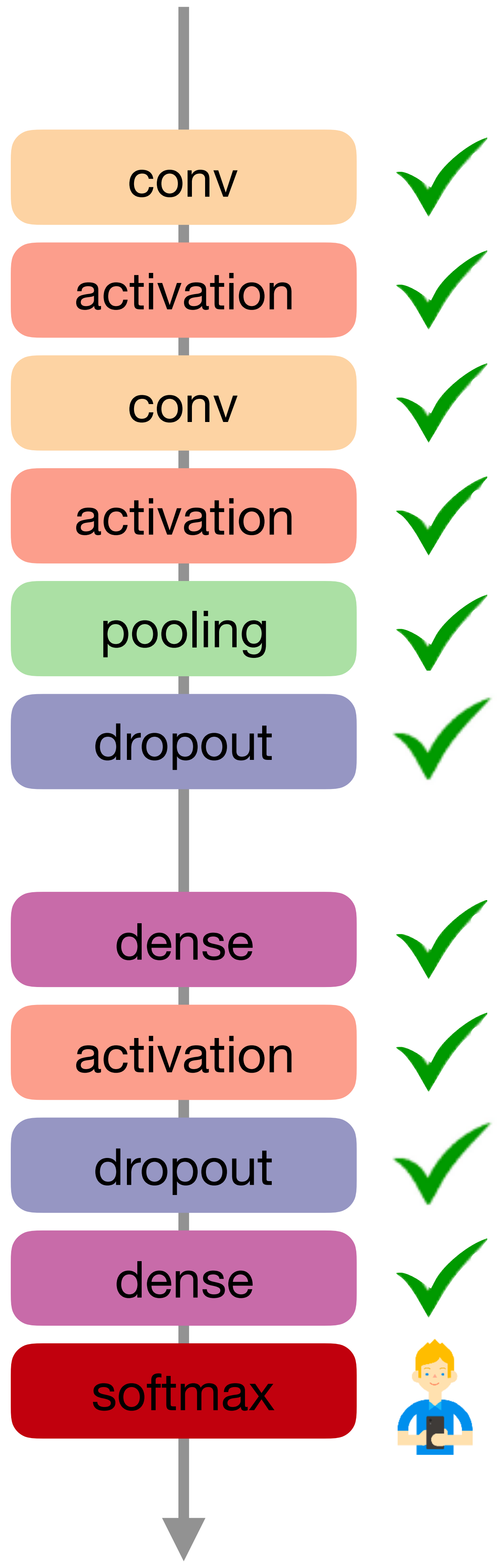


Encrypted Prediction

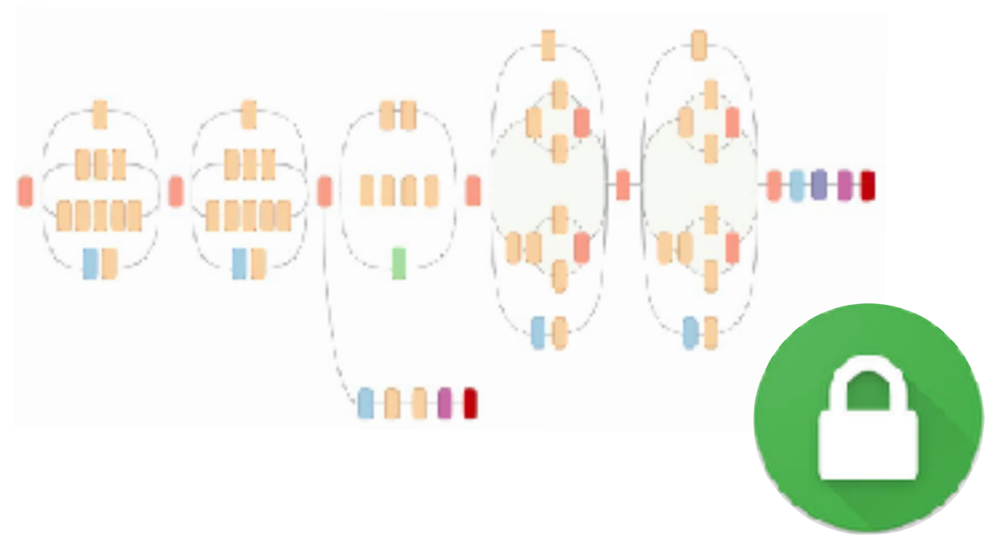


feature_layers = [

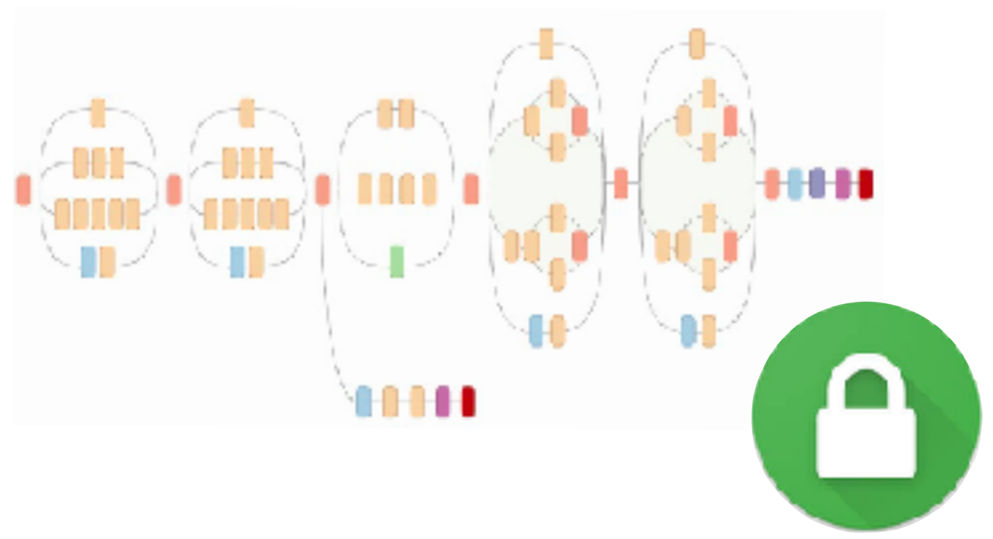
```
feature_layers = [
    Conv2D(32, (3, 3), padding='same', input_shape=(28, 28, 1)),
    Activation('sigmoid'),
    Conv2D(32, (3, 3), padding='same'),
    Activation('sigmoid'),
    AveragePooling2D(pool_size=(2,2)),
    Dropout(.25),
    Flatten()
]

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

model = Sequential(feature_layers + classification_layers)
```



Private Machine Learning

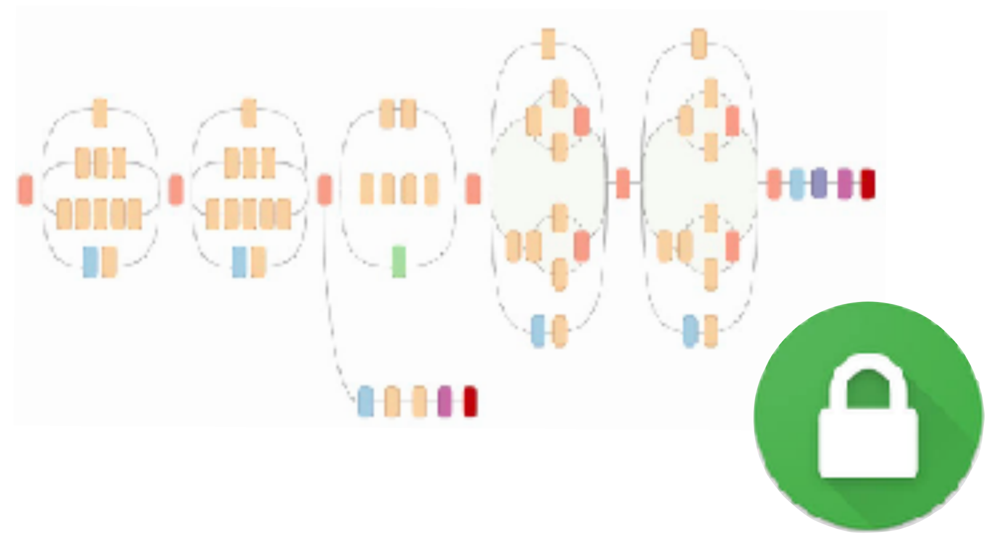


Private Machine Learning



access to data

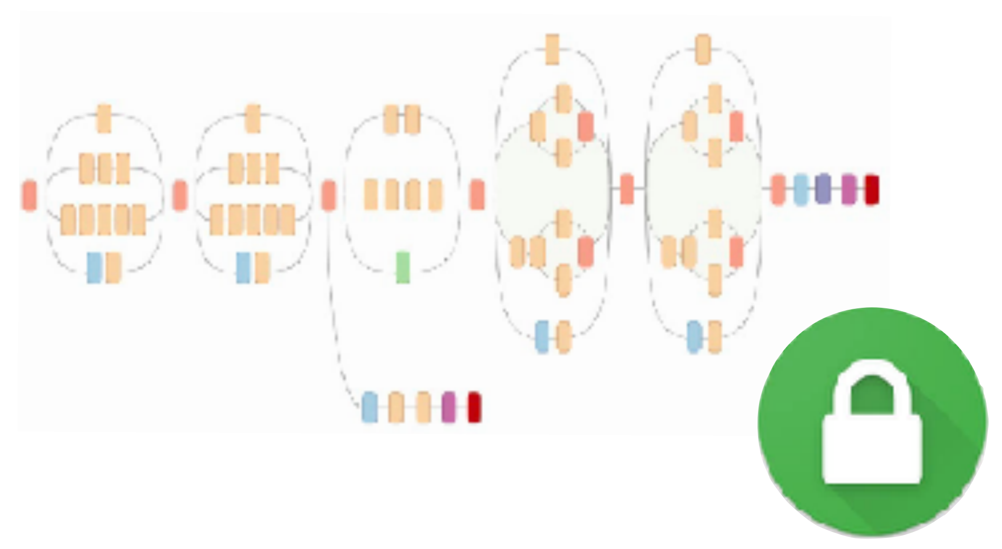
Private Machine Learning



access to data

monetise on data

Private Machine Learning

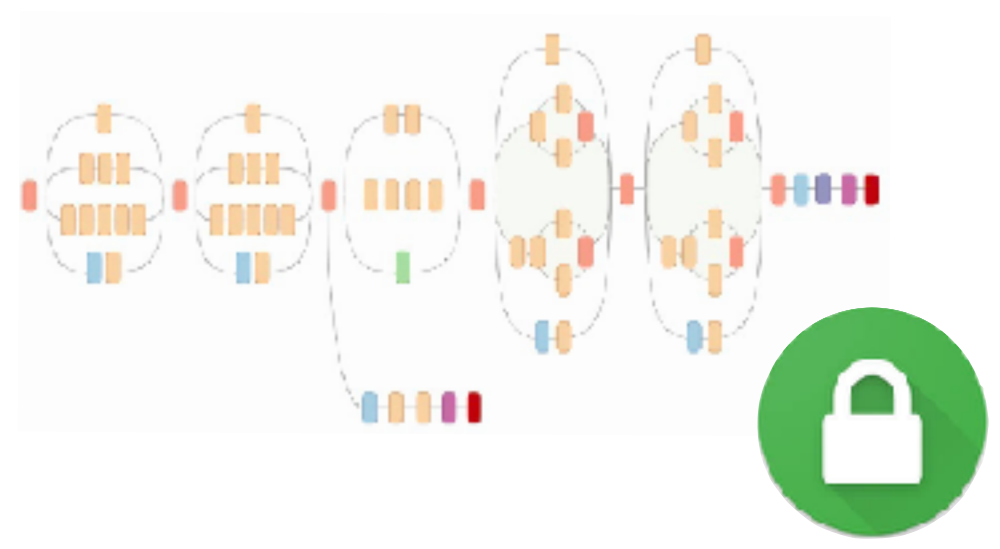


access to data

reduce risk

monetise on data

Private Machine Learning



access to data

reduce risk

monetise on data

incentivise use

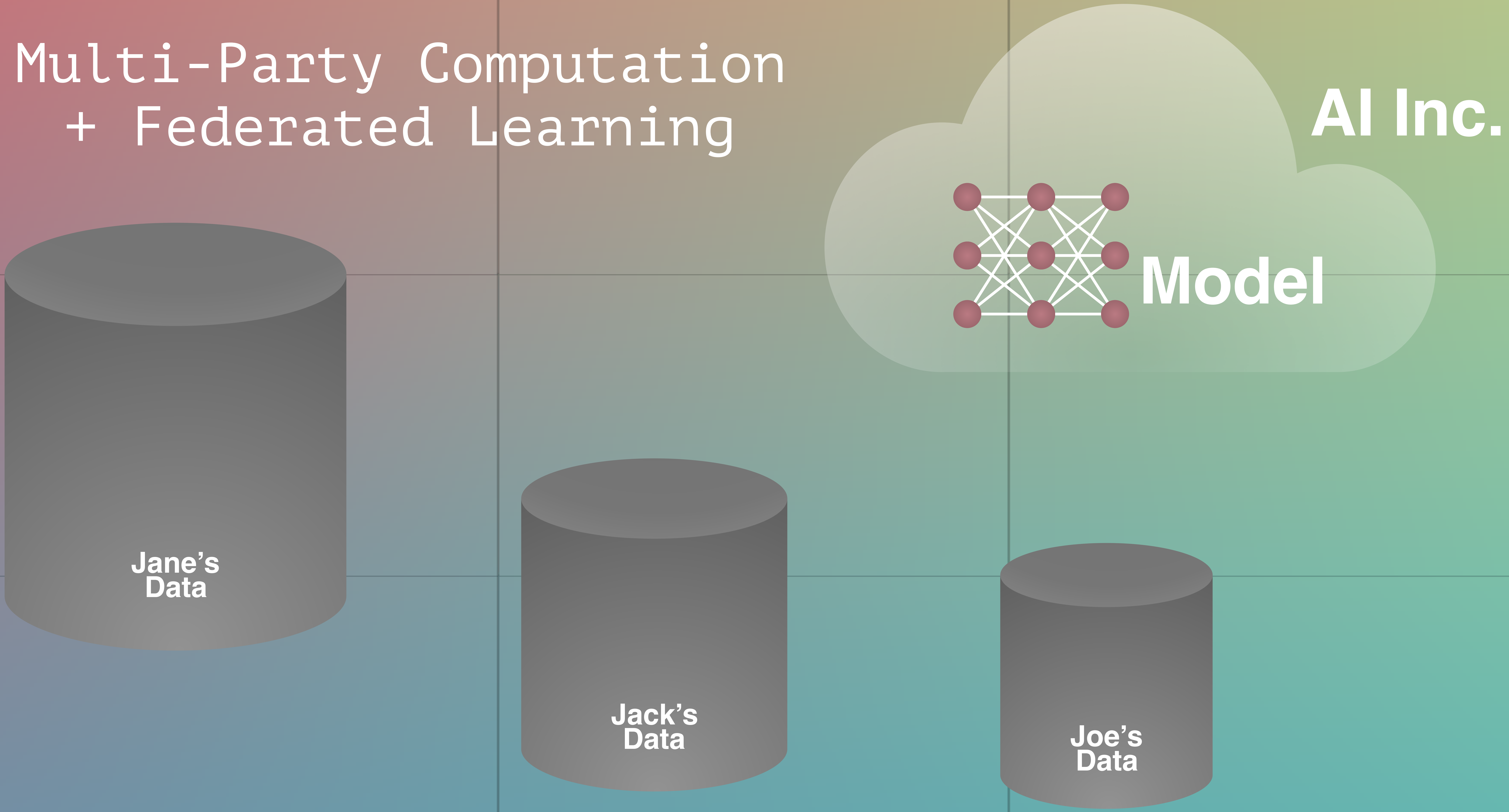
Tools for Safe AI

- Federated Learning
- Homomorphic Encryption
- Multi-Party Computation
- Gradient Validation Markets

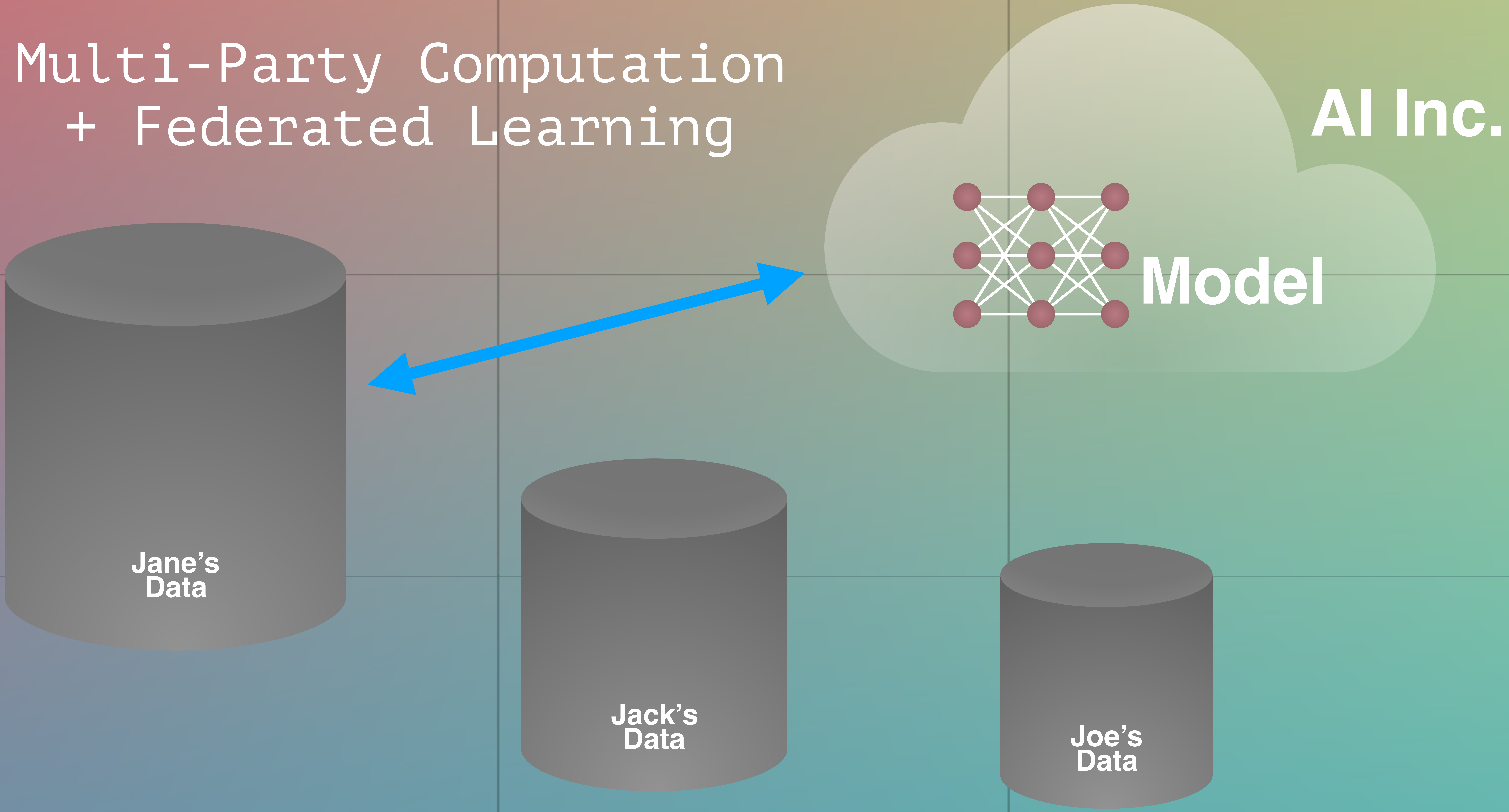
OpenMined

Slides originally by Andrew Trask

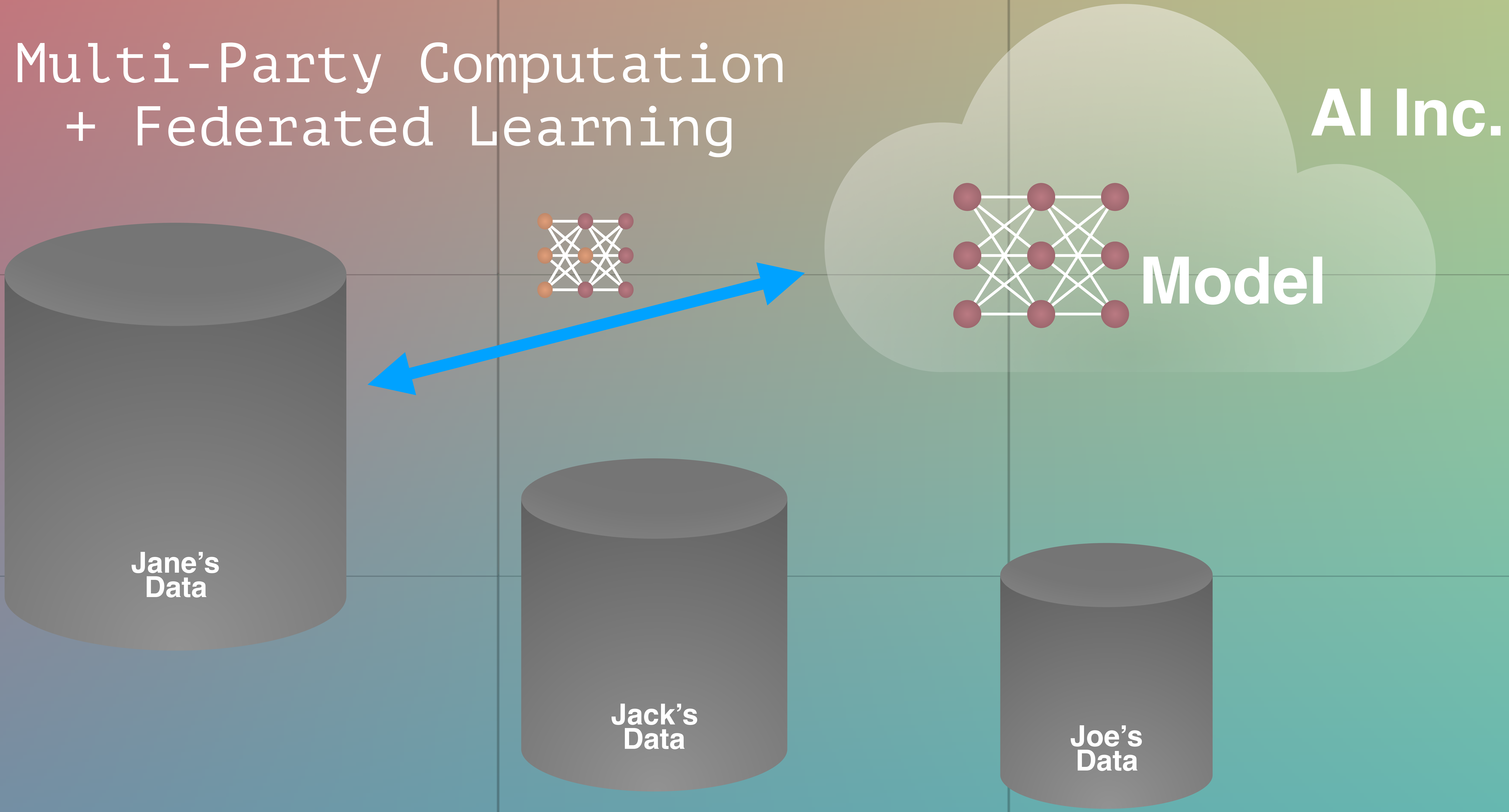
Multi-Party Computation + Federated Learning



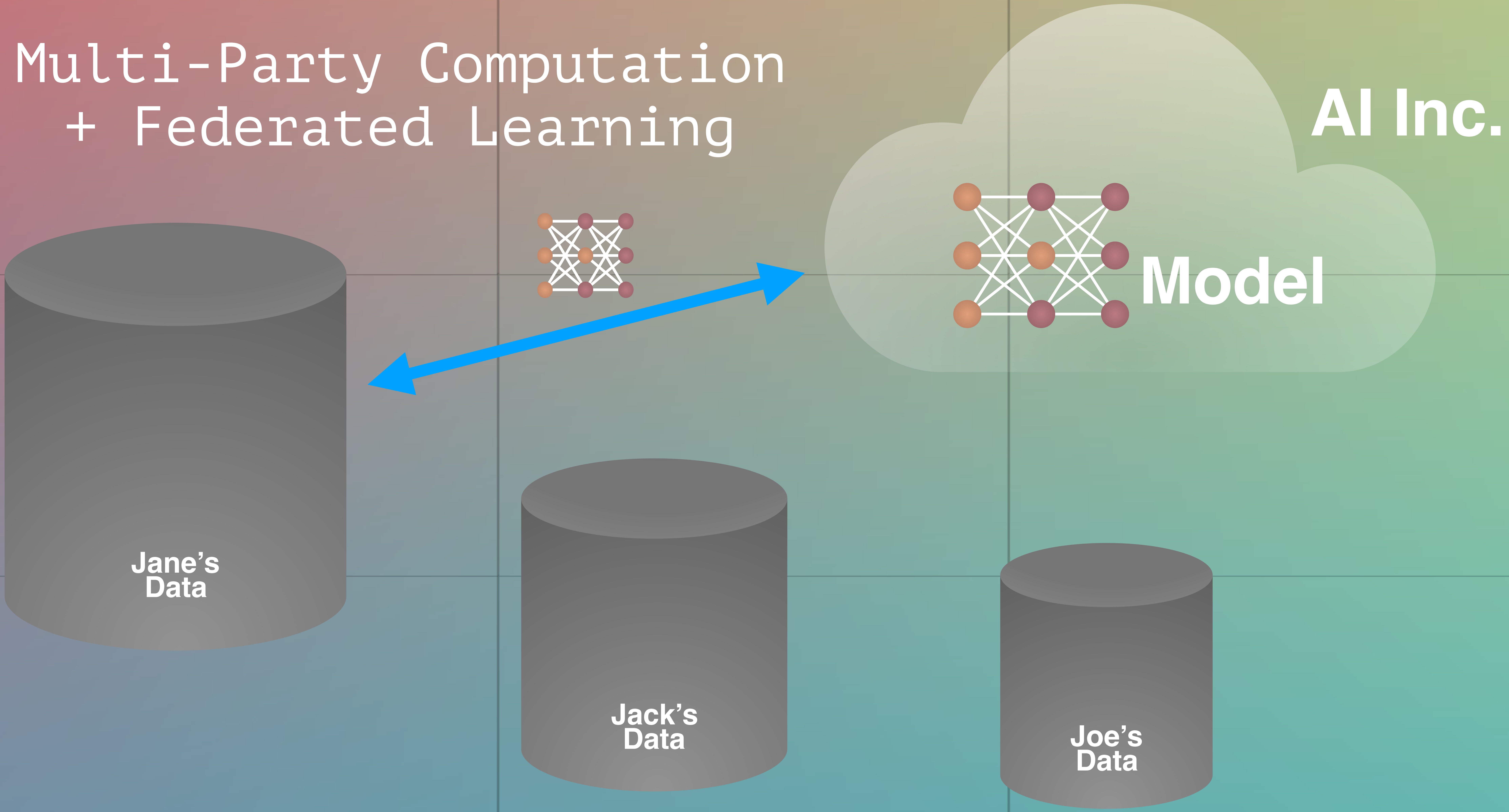
Multi-Party Computation + Federated Learning



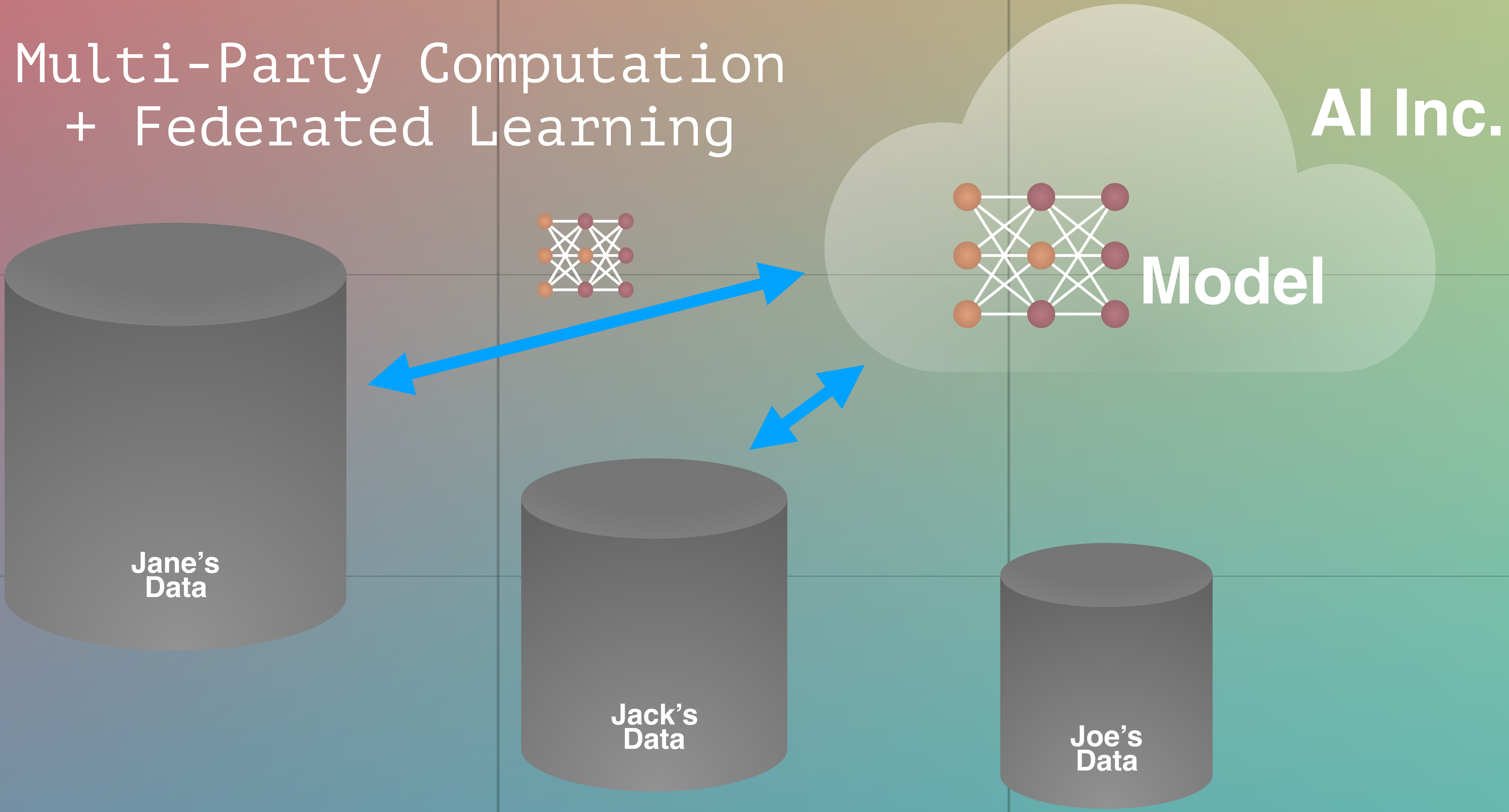
Multi-Party Computation + Federated Learning



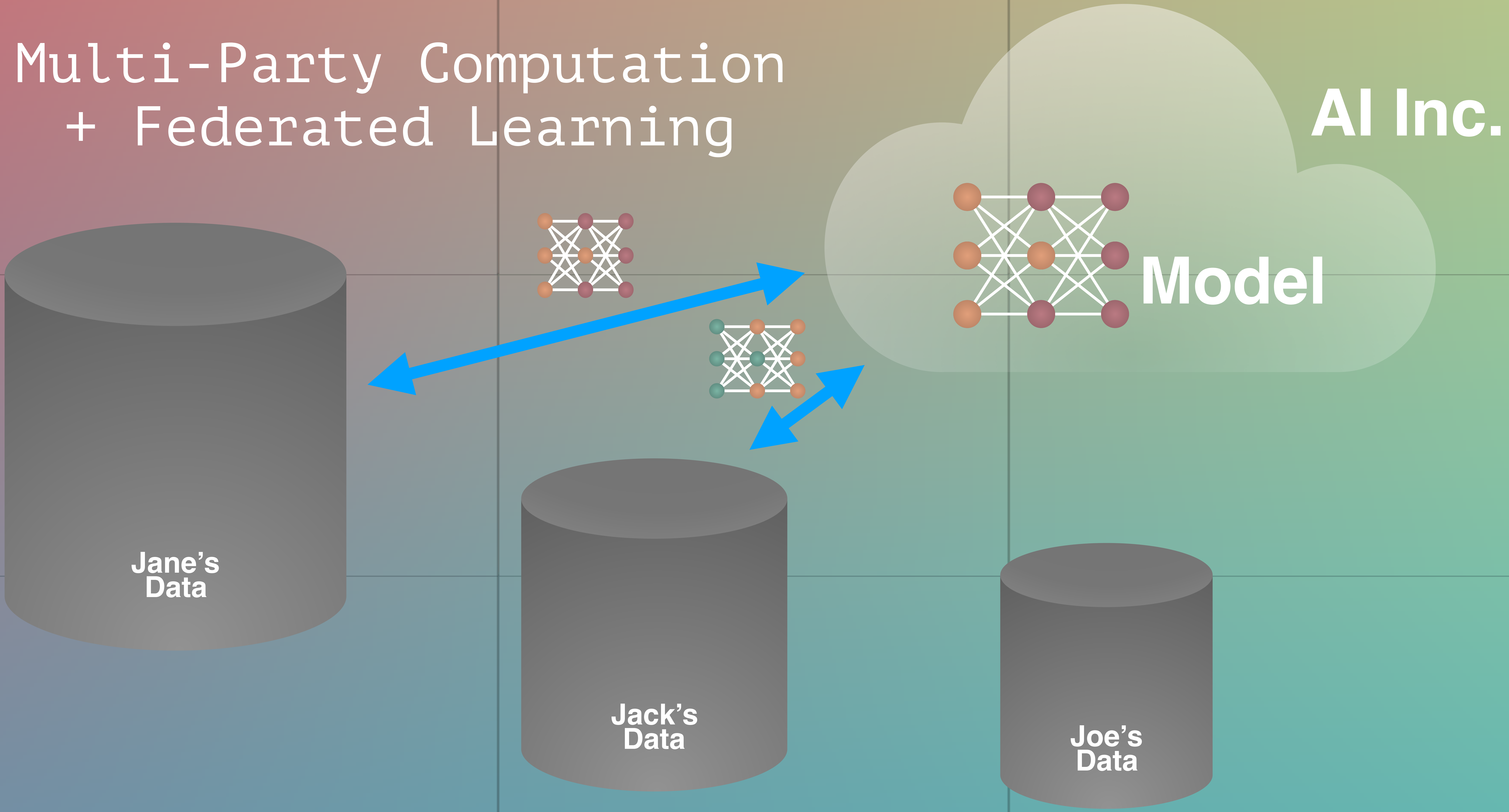
Multi-Party Computation + Federated Learning



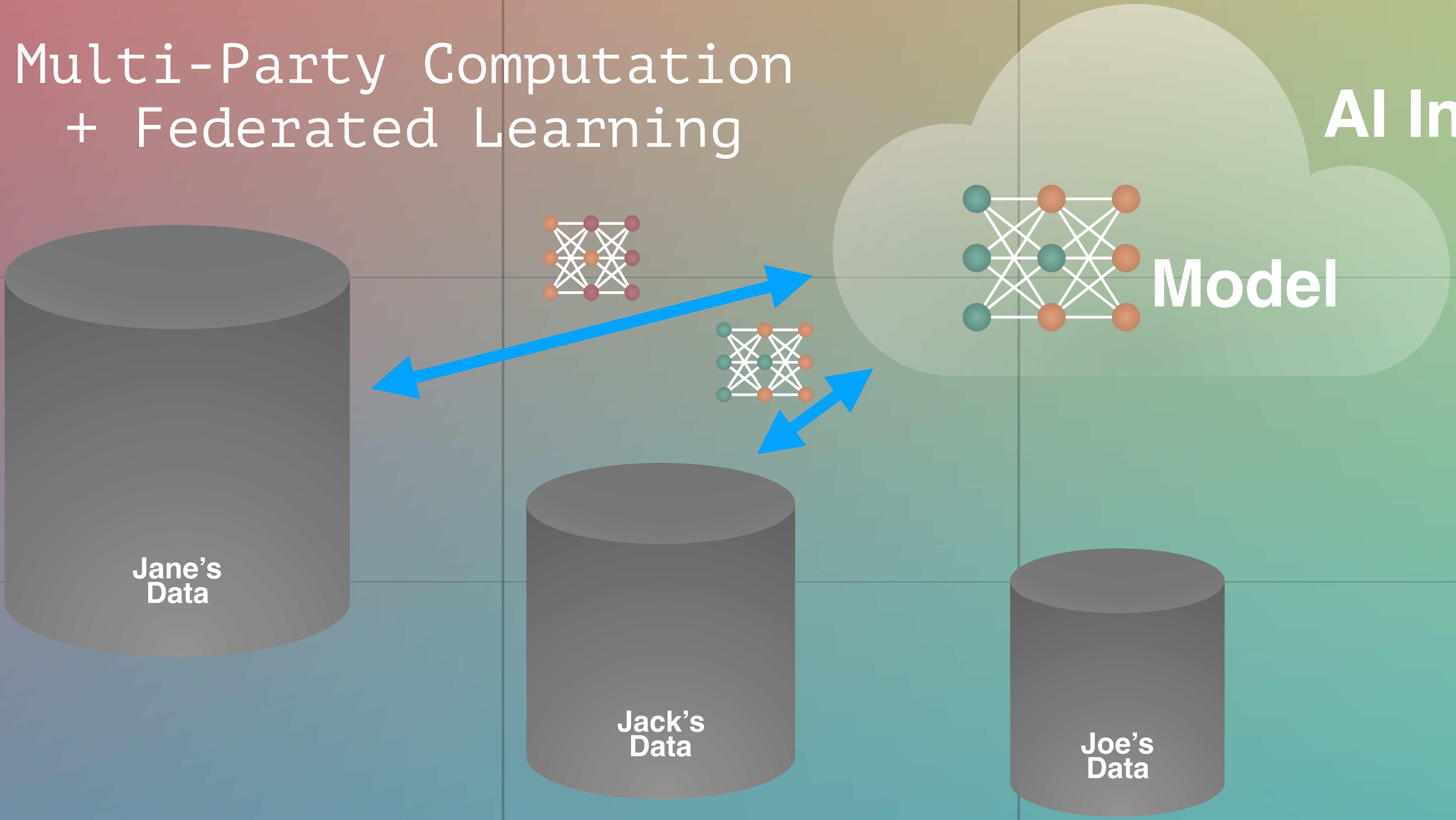
Multi-Party Computation + Federated Learning



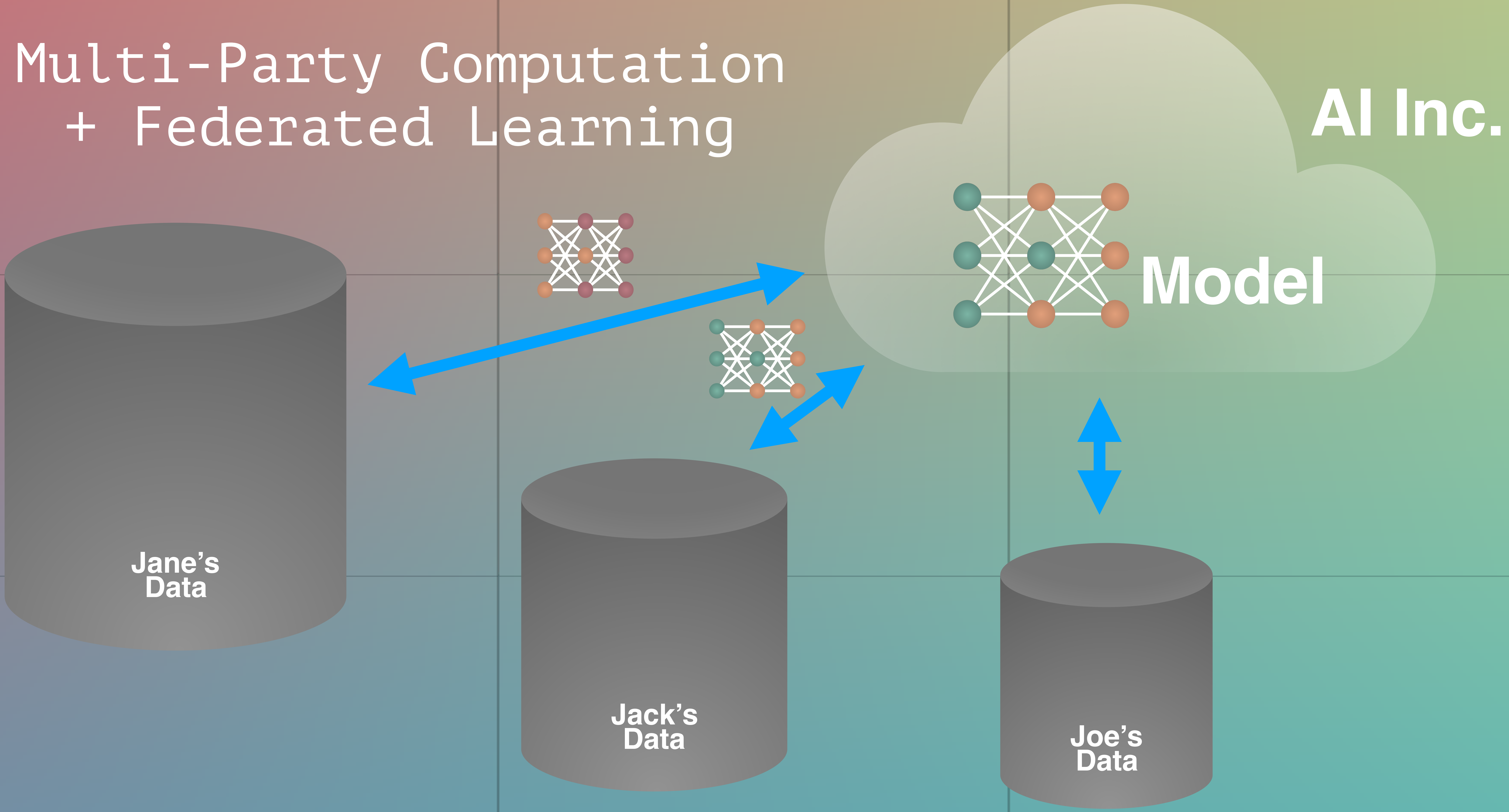
Multi-Party Computation + Federated Learning



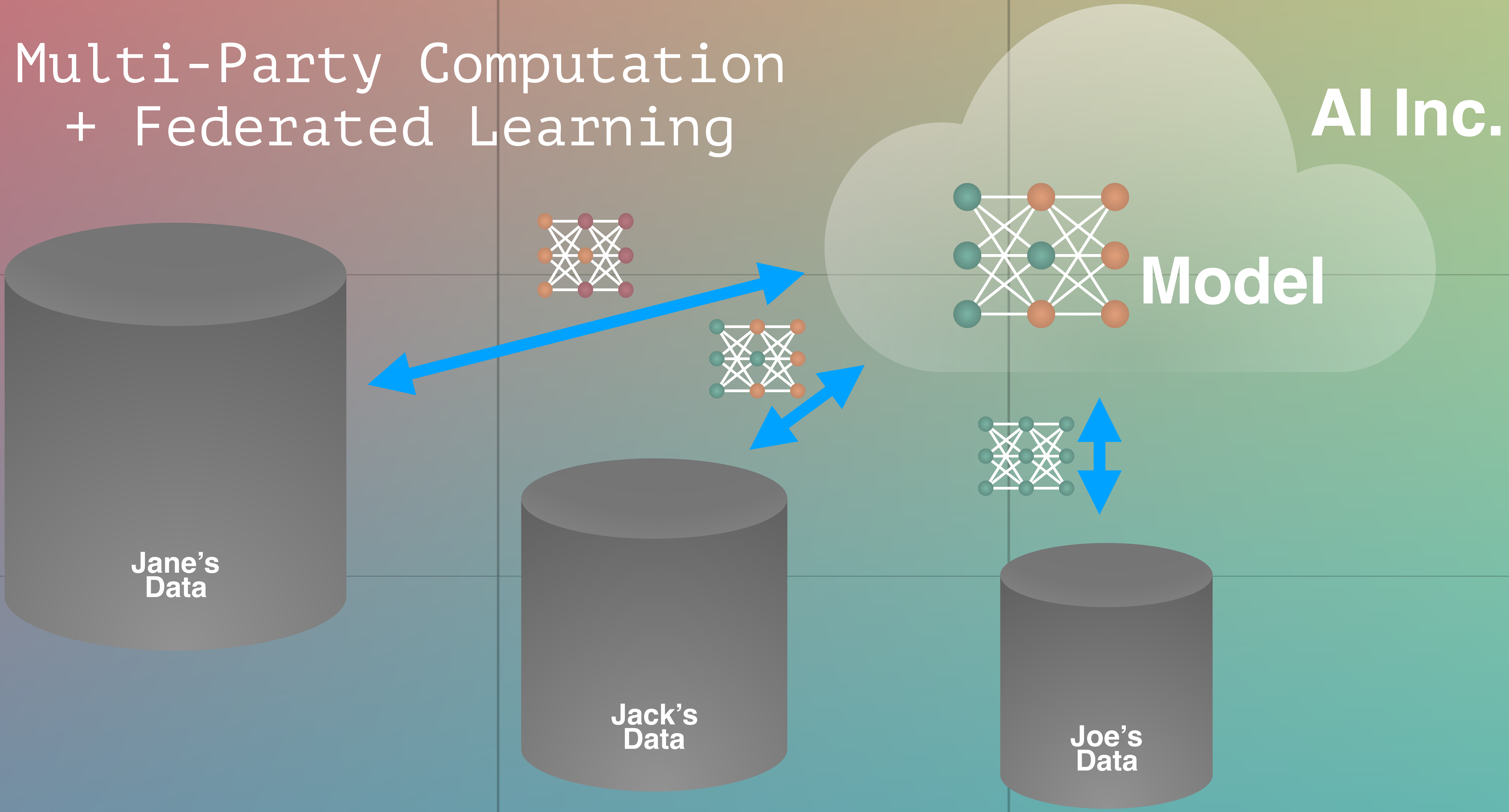
Multi-Party Computation + Federated Learning



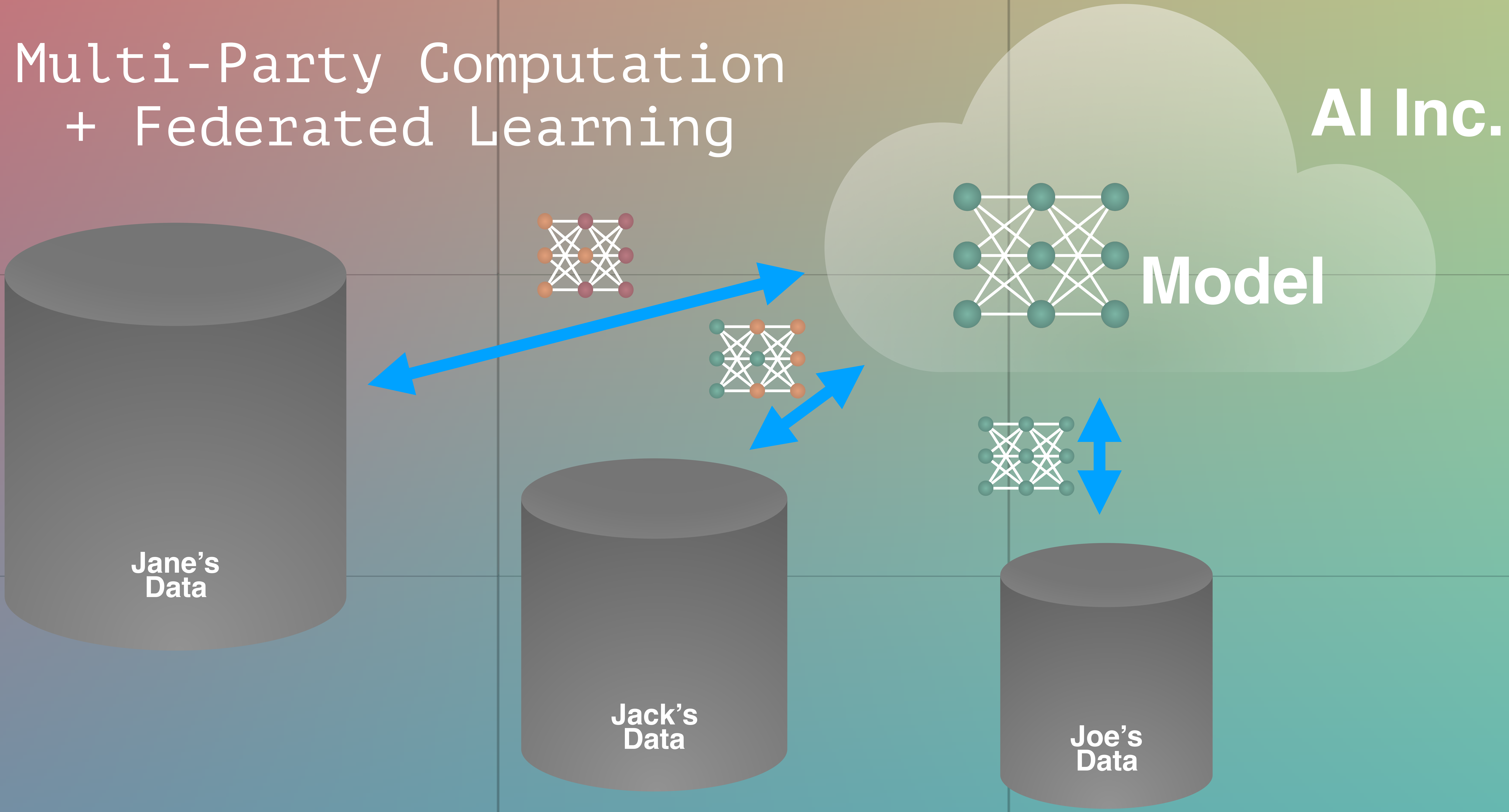
Multi-Party Computation + Federated Learning



Multi-Party Computation + Federated Learning



Multi-Party Computation + Federated Learning



Thank you

Private Image Analysis with MPC

Training CNNs on Sensitive Data using SPDZ

by Morten Dahl on September 19, 2017

This post is still a work in progress; in particular, its concrete practically is yet to be seen from implementation experiments.

TL;DR: *In this blog post we take a typical CNN deep learning model and go through a series of steps that enable both training and prediction to instead be done on encrypted data.*

Using deep learning to analysis images through convolutional neural networks (CNNs) has gained enormous popularity over the last few years due to their

@mortendahlcs

mortendahl.github.io

OpenMined

@openminedorg

openmined.org