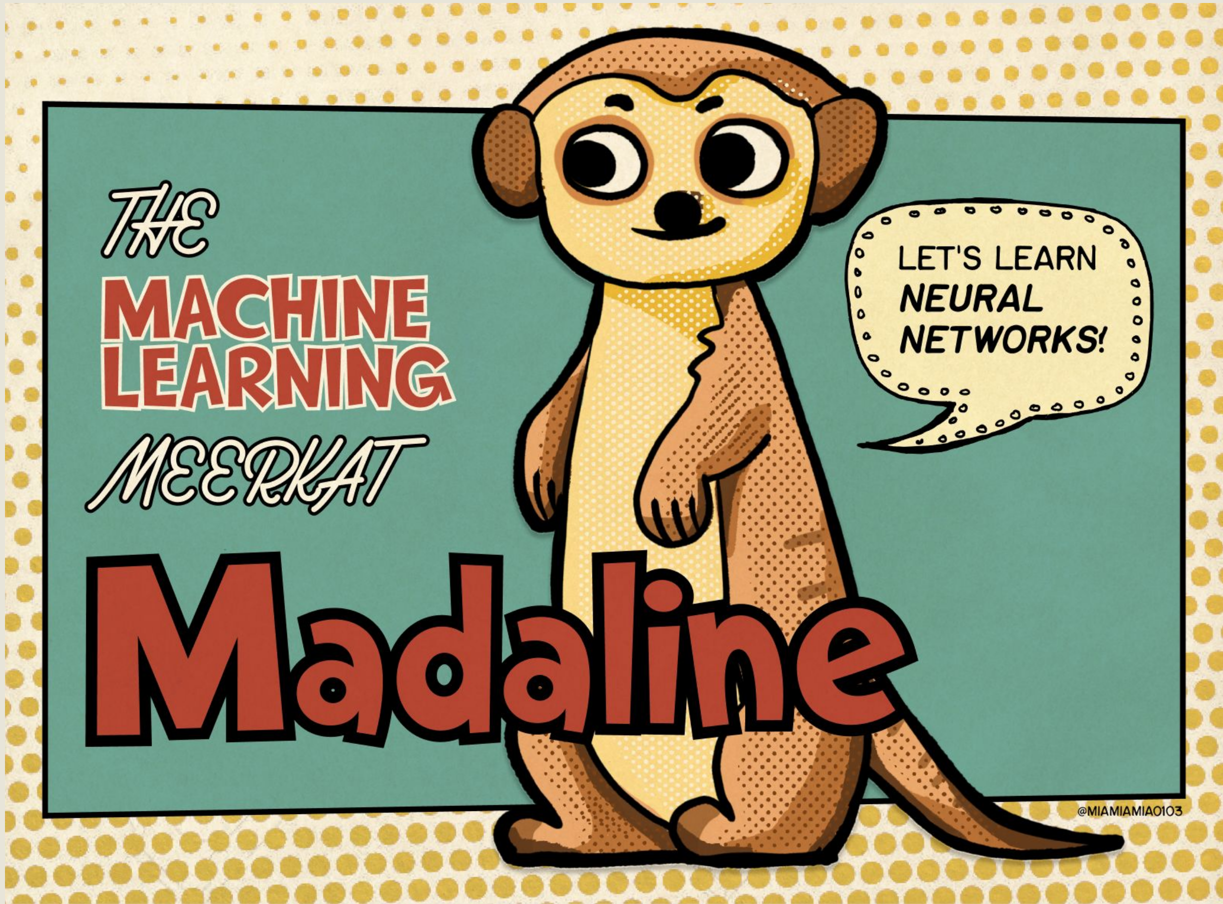
Machine Learning & Neural Networks

1

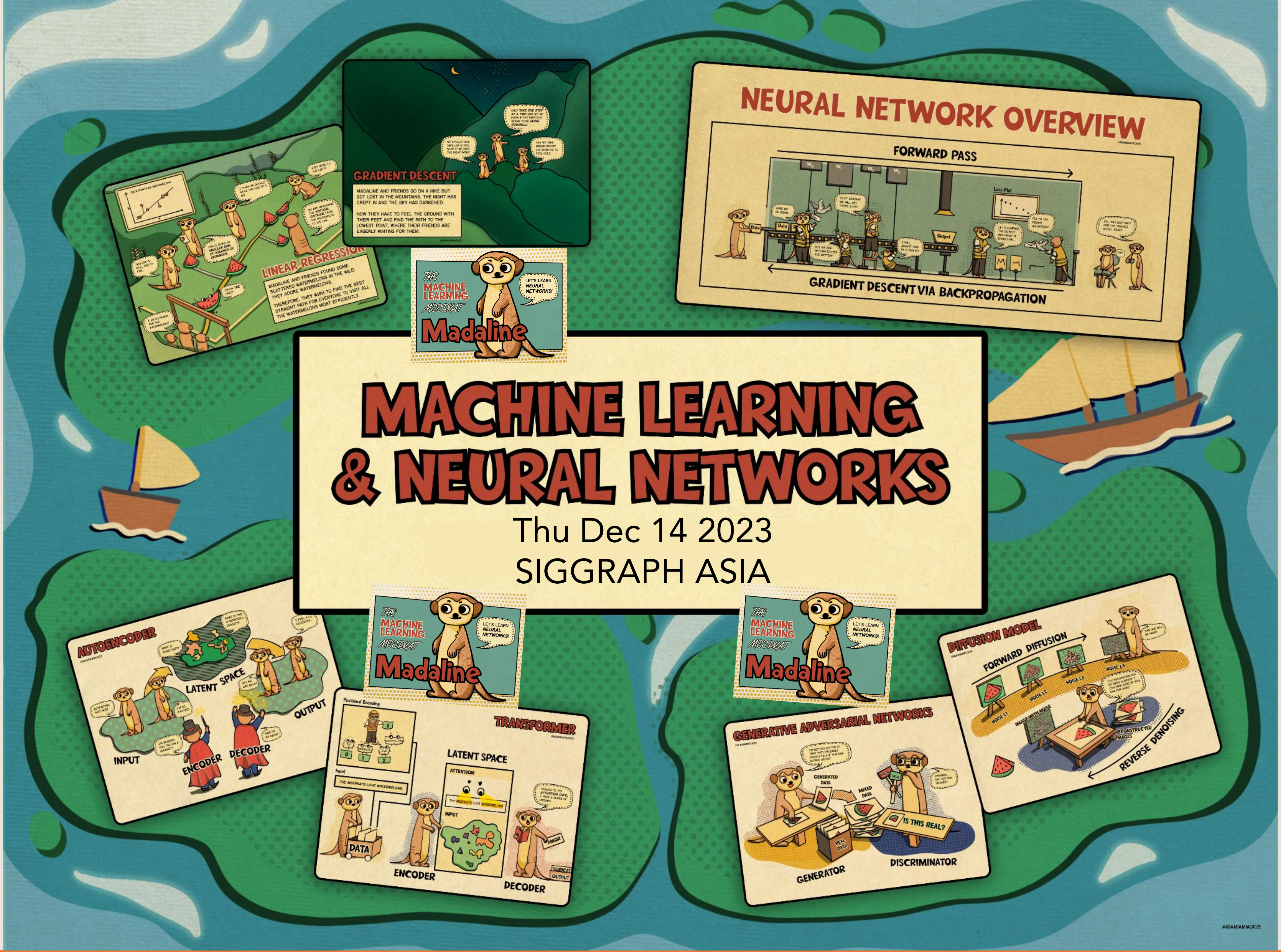
Rajesh Sharma @xarmalarma
Mia Tang @miamiamia0103



MACHINE LEARNING & NEURAL NETWORKS

Thu Dec 14 2023

SIGGRAPH ASIA



Course Materials - Slides and Notebooks

http://bit.ly/3qm76Fg

Course Details

Hour 1 - BASICS, NEURAL NETWORKS, MATH

Learning Outcome:

After this hour you should be able to:

- Define machine learning
- Do preliminary data analysis
- Understand the general framework for ML development
- Understand Regression and Classification
- Describe a Neural Network and associated terms
- Know how to minimize loss via Gradient Descent
- Gain a probabilistic intuition for Log-likelihood, MSE

-- HOUR 1 TOPICS --

- • Introduction and Course Overview
- • Software setup for Hands-on programming
- • What is Machine Learning, What are Neural Networks?
- • Framework for Learning: Theory, Intuition, Practice
- • Machine Learning Model vs Theoretical Model Example
- • Data Analysis: Example Housing Prices
- • General Framework for ML development, Training & Inference
- • Example: Regression with Neural Networks
- • Math: Loss Minimization, Gradient Descent, MLE, Log-Likelihood
- • Classification: Example: Flower Type Identification

Hour 1.5 - NEURAL NETWORKS

Learning Outcome:

After this hour you should be able to:

- Understand different kinds of Neural Networks
- Explain supervised learning for images

-- HOUR 1.5 TOPICS --

- Types of Neural Networks
- Example: AutoEncoder, Application to Denoising
- Example: Convolutional Neural Network
- Example: Variational Autoencoder
- Latent Space Examination
- Example: Generative Adversarial Network

Hour 2 GENERATIVE MODELS - TRANSFORMERS

Learning Outcome:

After this hour you should be able to:

- Understand 'attention'
- Learn what is Word2Vec and what are Word-Embeddings
- Explain the basic Transformer architecture
- Describe how Chat-GPT like models work

-- HOUR 2 TOPICS --

- Attention mechanism
- Transformer Architecture
- Transformer → Chat
- Fine Tuning
- Reinforcement Learning from Human Feedback (RLHF)

Hour 3 OTHER GENERATIVE MODELS

Learning Outcome:

After this hour you should be able to:

- Understand NeRFs and Gaussian splatting for novel view synthesis
- Explain the diffusion model and how these are used for image generation
- Realize the ethical and legal issues in AI and ML
- Separate hype from reality
- Seek additional resources for further learning

-- HOUR 4 TOPICS --

- Gaussian Splatting
- NeRFs
- Diffusion
- Ethical Issues in Machine Learning & AI
- Summary and Next Steps for Additional Learning

ML/AI is already everywhere

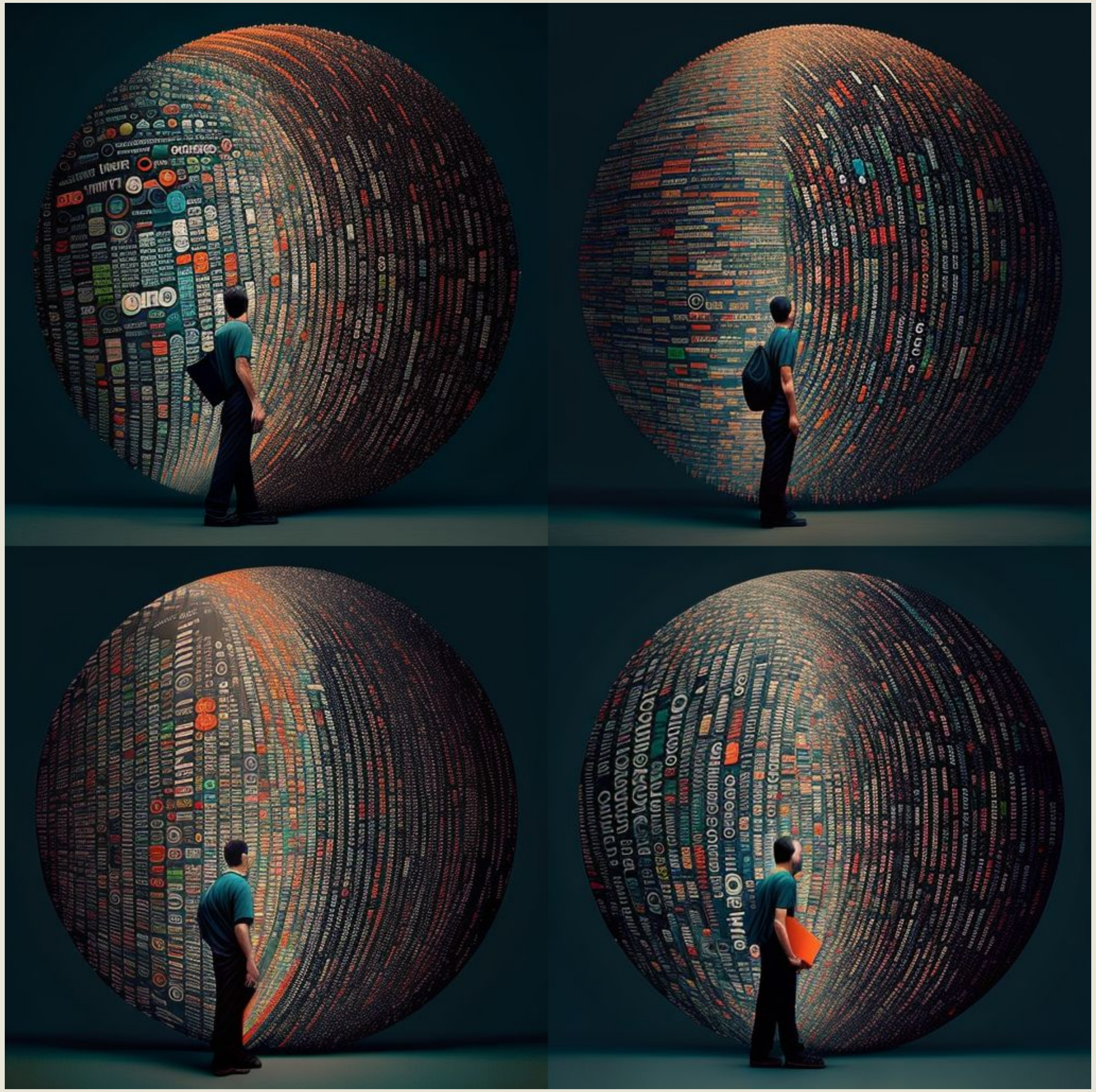
Smartphones: voice assistant, identity, battery life
Email/Text: spam filters, autocomplete, grammar
Driving: vision, lidar, self-driving
Photography: enhancement, geometry capture
Navigation: live view, directions, mapping
Banking: identity, fraud detection, credit scoring
Agriculture: watering, fertilizer, weed control
e-Commerce: recommendations, adtech, translation
Entertainment: recommendations, CGI, SORA, Diffusion
Coding/Chat: ChatGPT, Bard, Gemini, Claude, LLama

Computer Graphics Applications

- ★ *Scheduling Optimization*
- ★ *Character AI*
- ★ *Style Transfer*
- ★ *Slow Motion*
- ★ *Up-Res*
- ★ *Photogrammetry*
- ★ *Rendering*
- ★ *Denoising*
- ★ *Story Sentiment*
- ★ *Rough to Fine*
- ★ *Body Tracking*
- ★ *Image Generation*
- ★ *Simulation*
- ★ *Materials/PBR*

Why now?

Enormous data



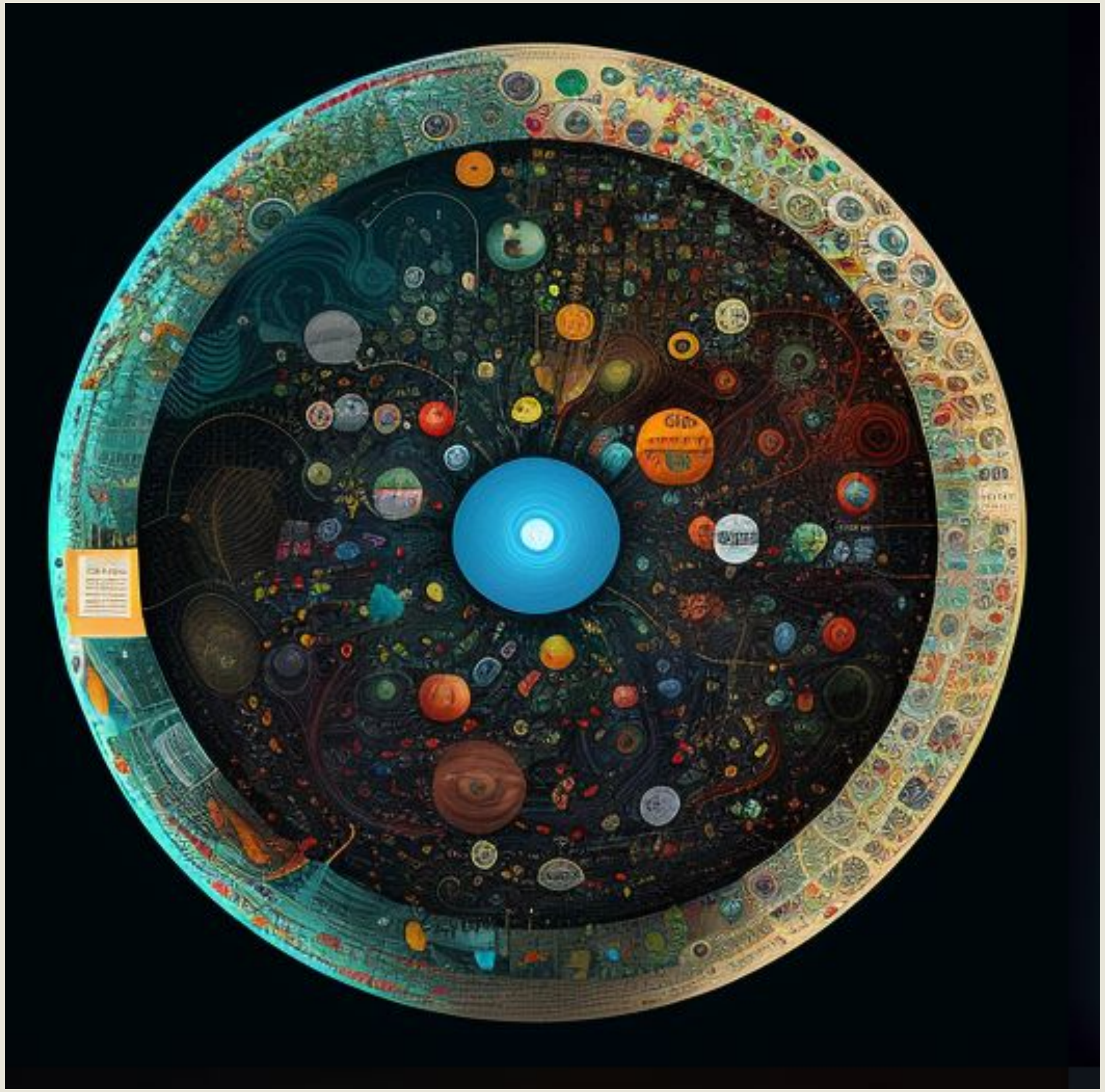
Why now?

Faster Compute (GPU)

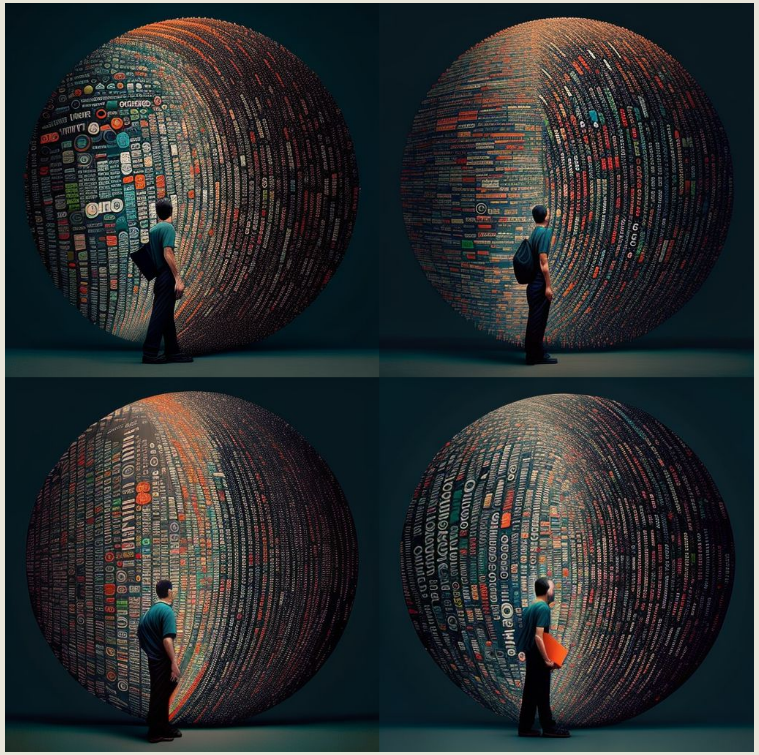
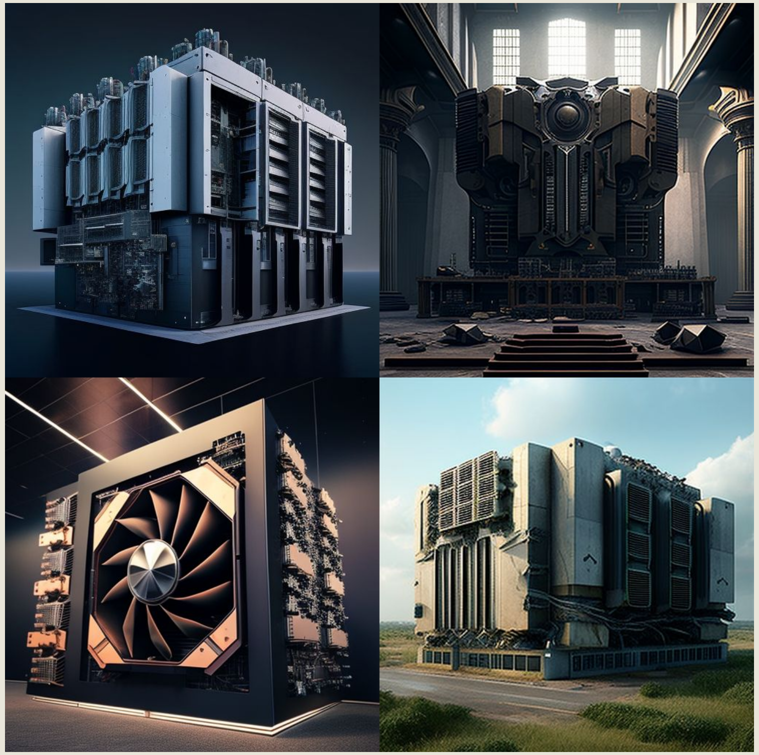
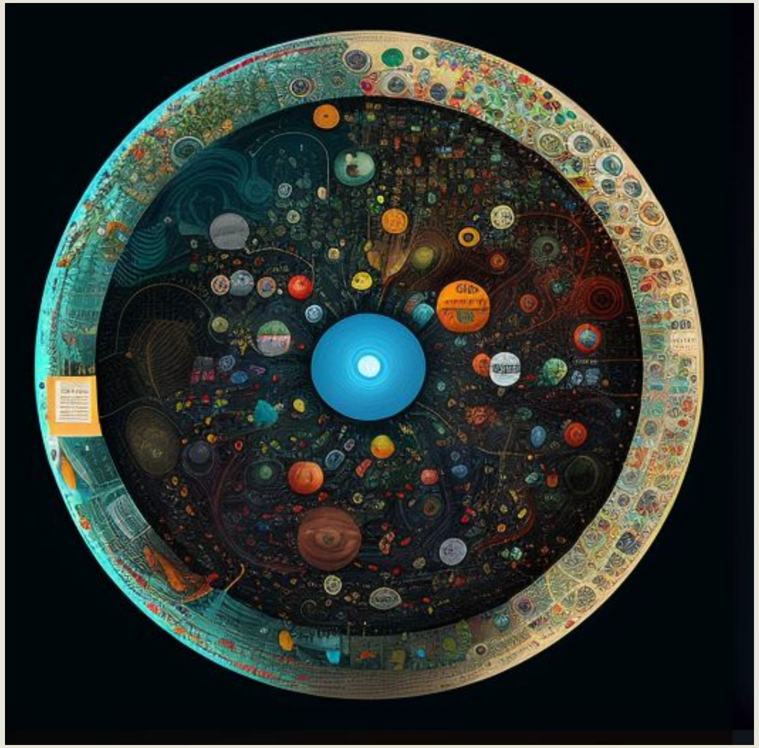
Enormous data

Why now?

Internet & Cloud
Faster Compute (GPU)
Enormous data



Why now?

Advanced Research
New Algorithms (Models)
Novel Applications

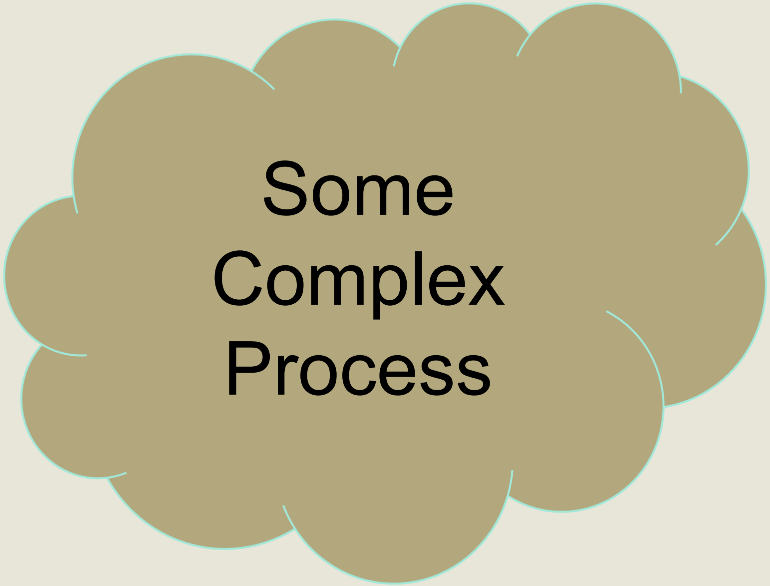
What is it?

Use and development of computer systems that are able to learn and adapt without following explicit instructions by using algorithms and statistical models to analyze and draw inferences from patterns in data.

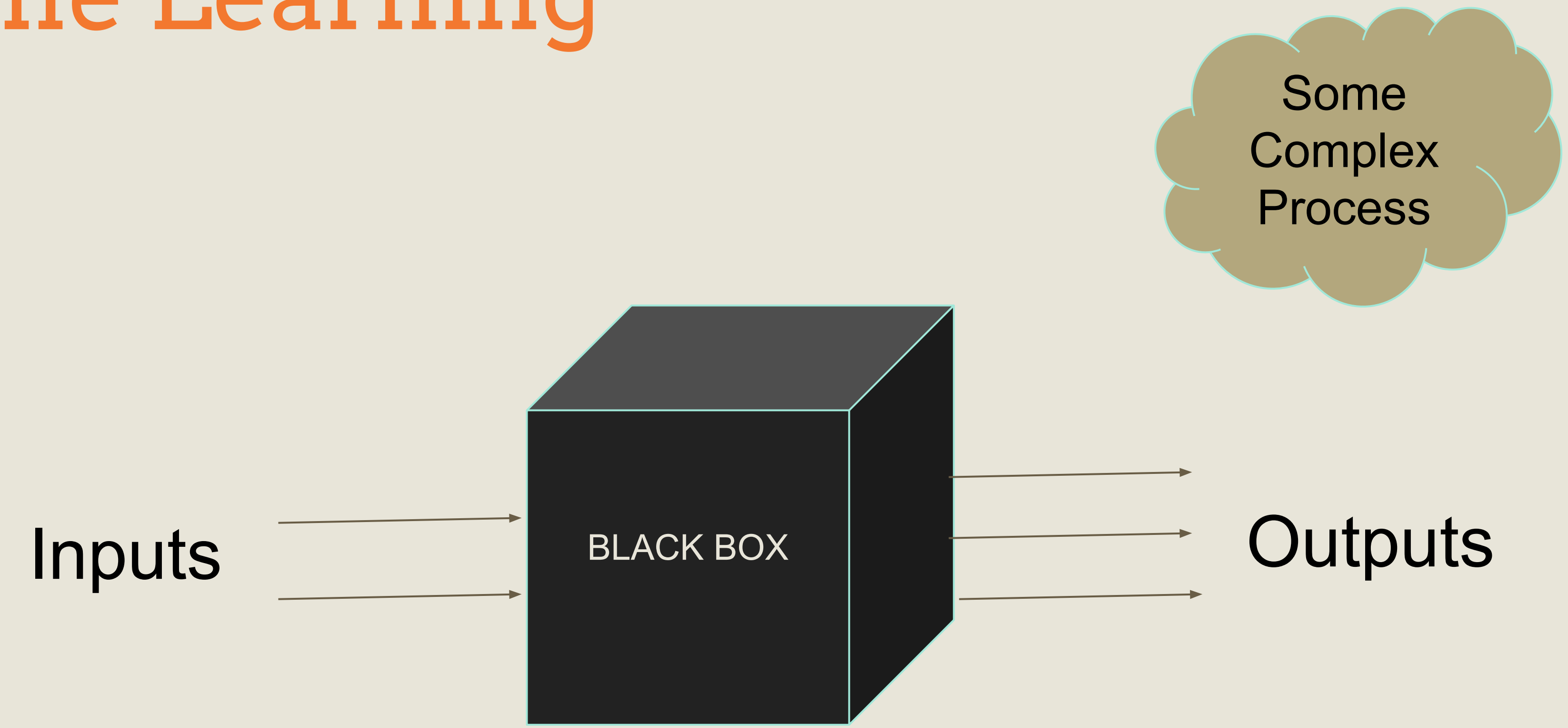
Machine Learning

Use and development of
-- computer systems that are able to learn and adapt
-- without following explicit instructions
-- by using algorithms and statistical models
-- to analyze and draw inferences
-- from patterns in data

Machine Learning

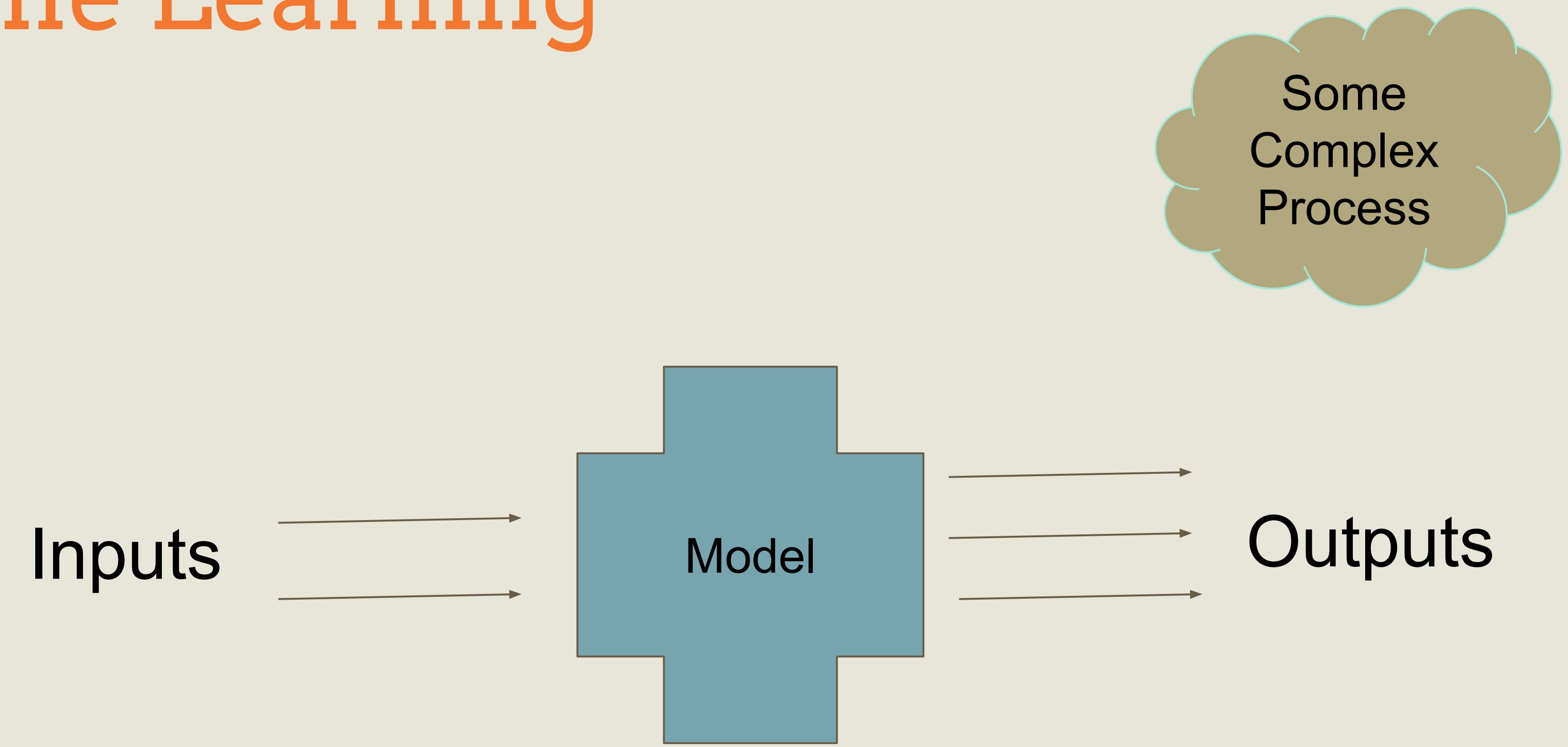


Machine Learning

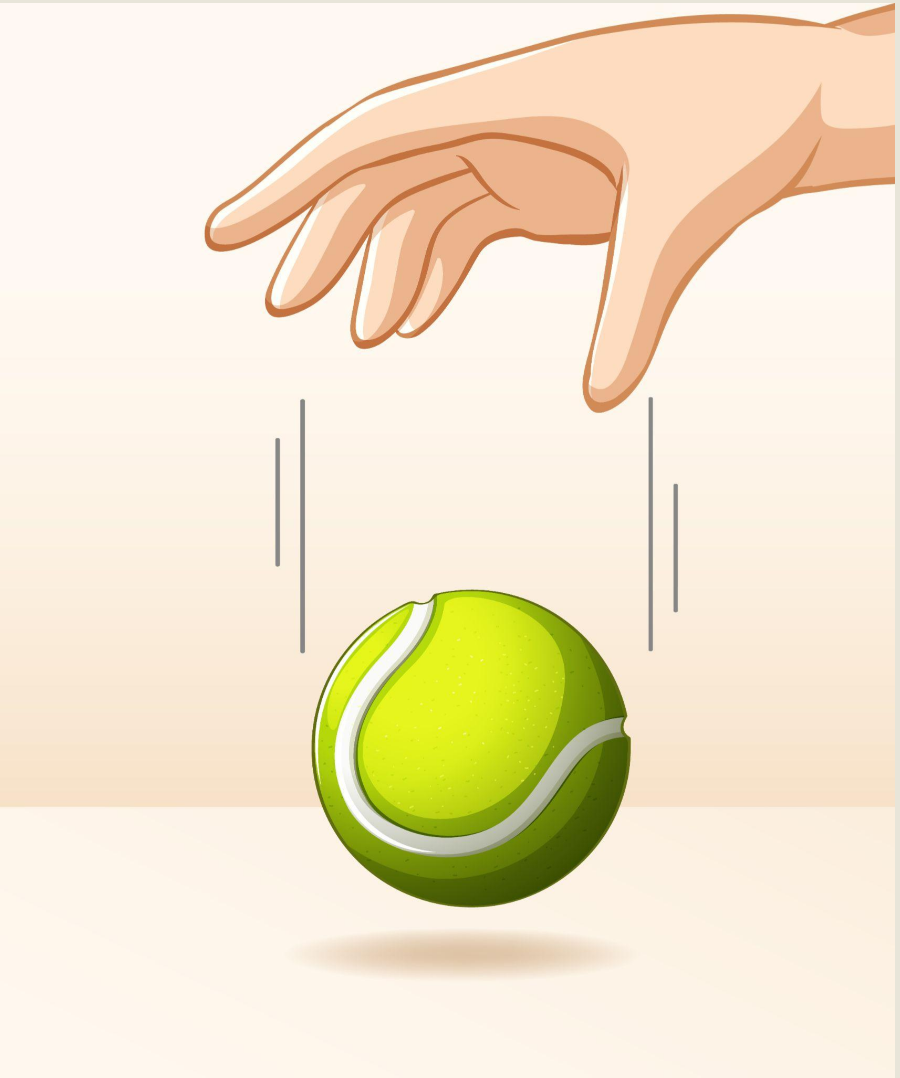


Machine Learning

Machine Learning



Example: Measuring Gravity

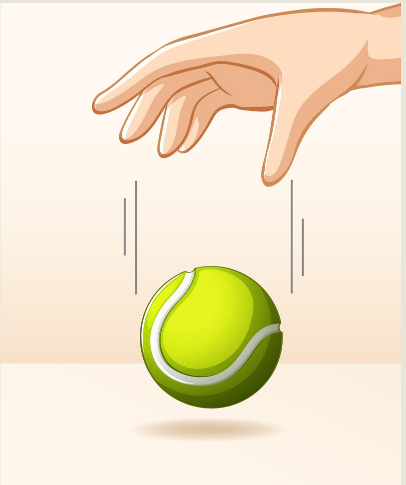


Measuring Gravity

Experimentation & Data Collection

- Drop ball from different heights
- Measure time for drop
- Repeat experiment multiple times

Example: Measuring Gravity



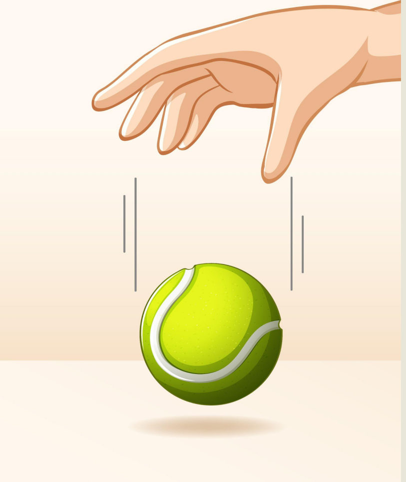
Measuring Gravity

Experiment and Data Collection

- Drop ball from different heights
- Measure time for drop
- Repeat experiment multiple times

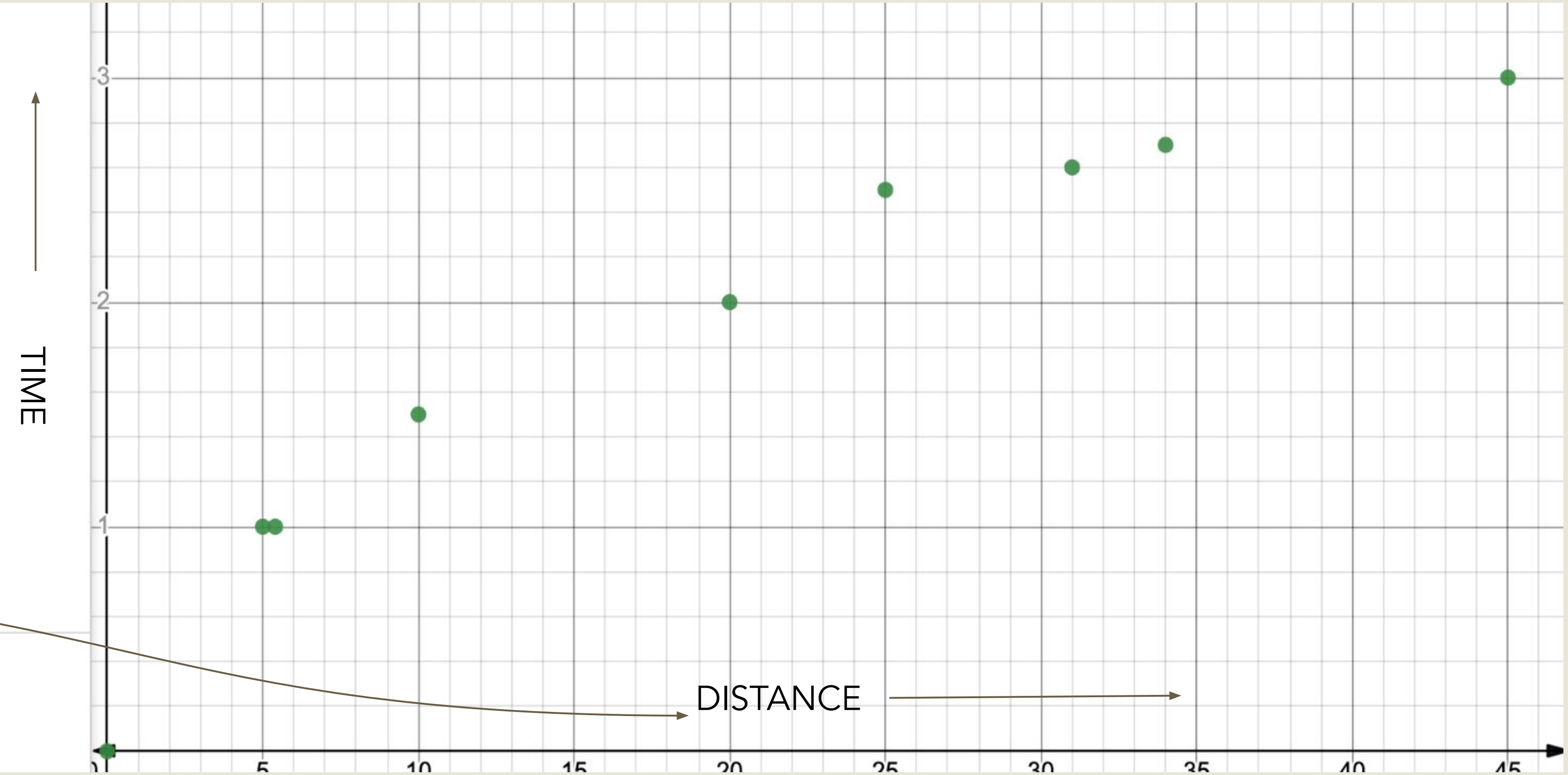
x_1	y_1
0	0
5	1
5.4	1
20	2
25	2.5
31	2.6
34	2.7
45	3
10	1.5

Example: Measuring Gravity

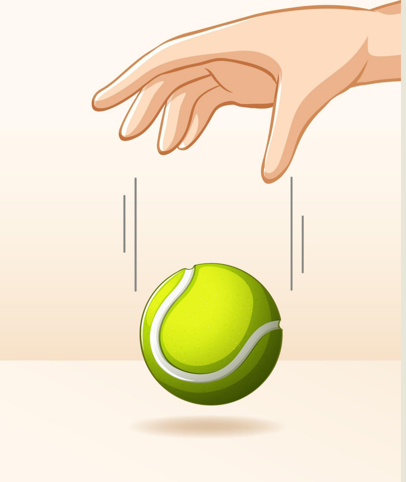


Measuring Gravity

0	0
5	1
5.4	1
20	2
25	2.5
31	2.6
34	2.7
45	3
10	1.5



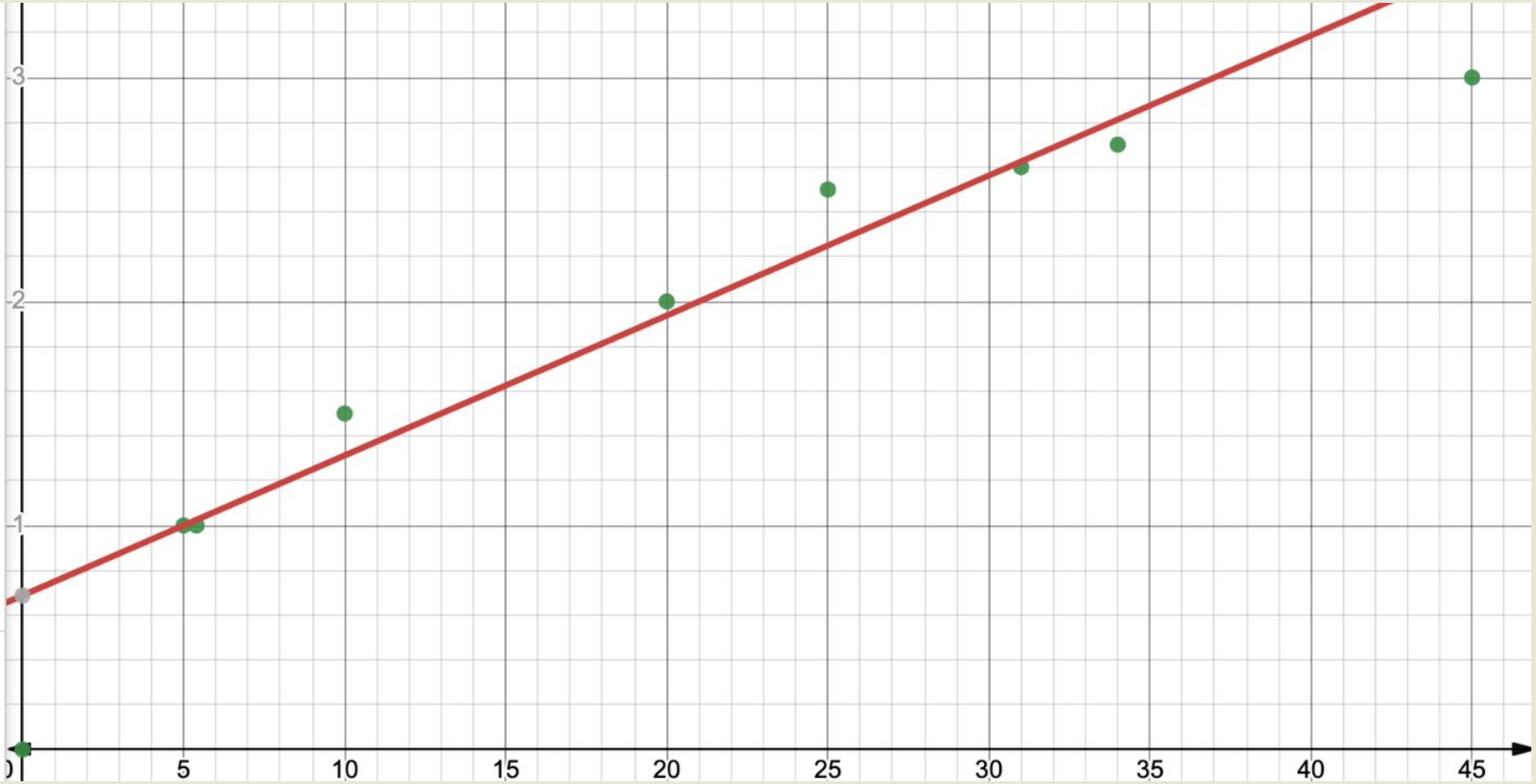
Example: Measuring Gravity



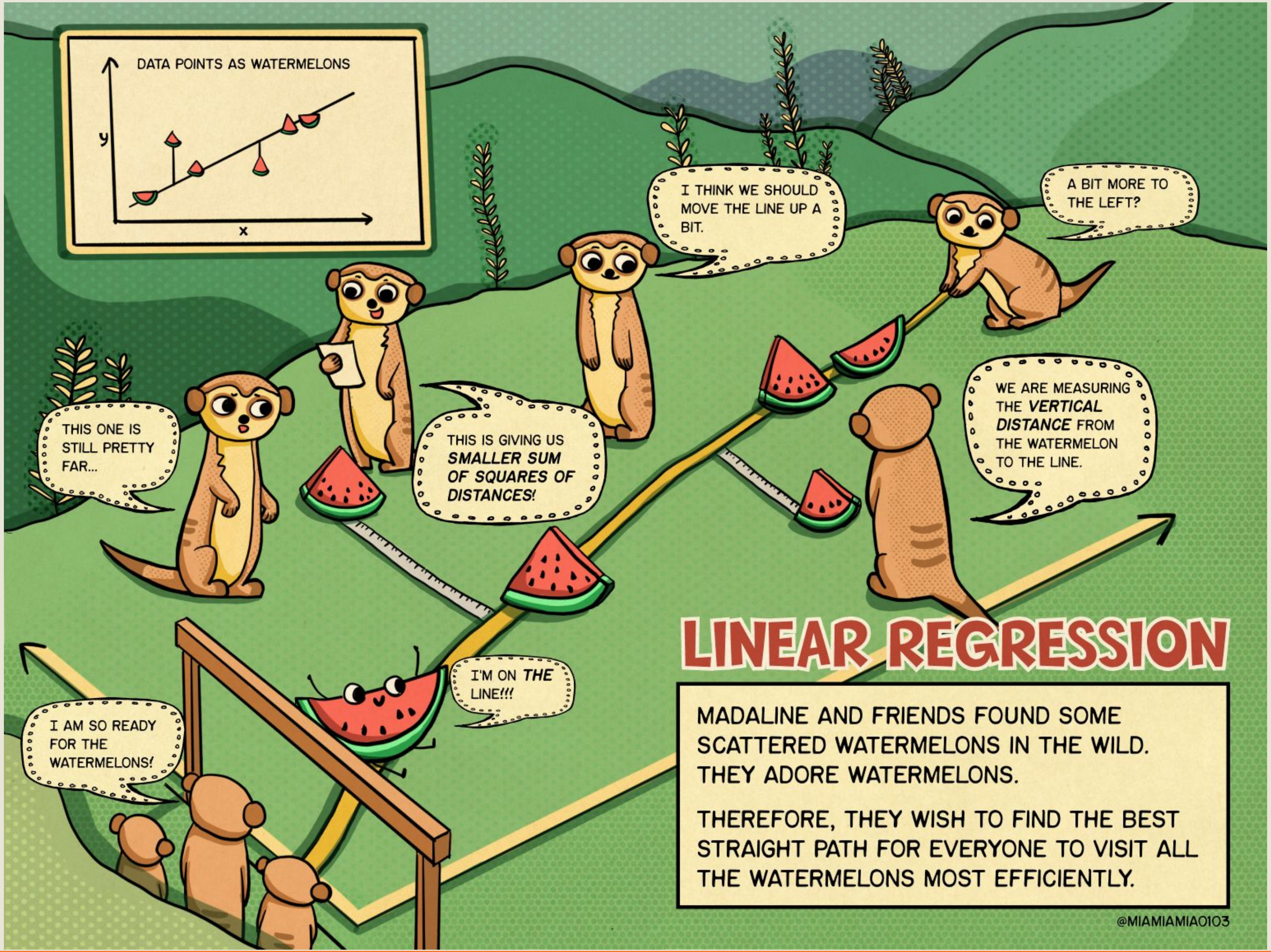
Measuring Gravity

0	0
5	1
5.4	1
20	2
25	2.5
31	2.6
34	2.7
45	3
10	1.5

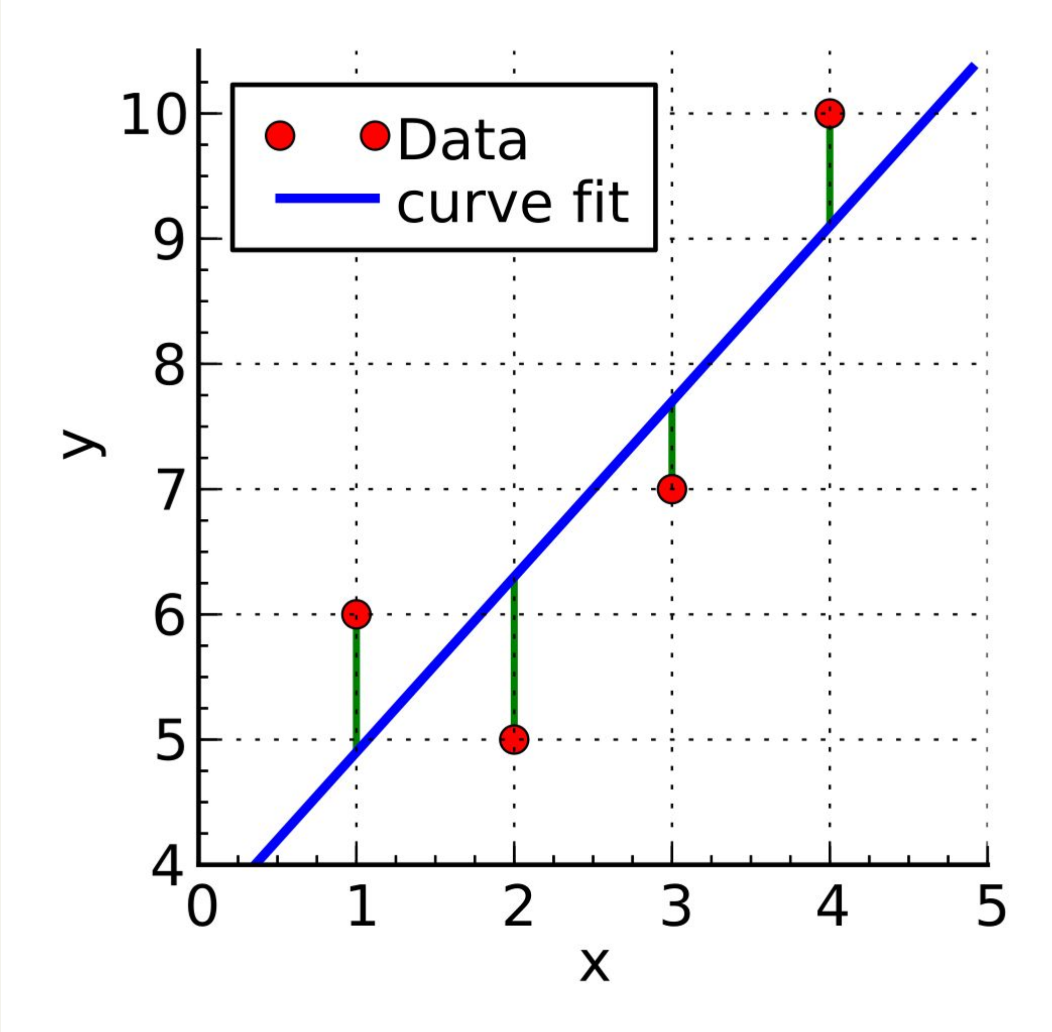
dist = 15*t - 11



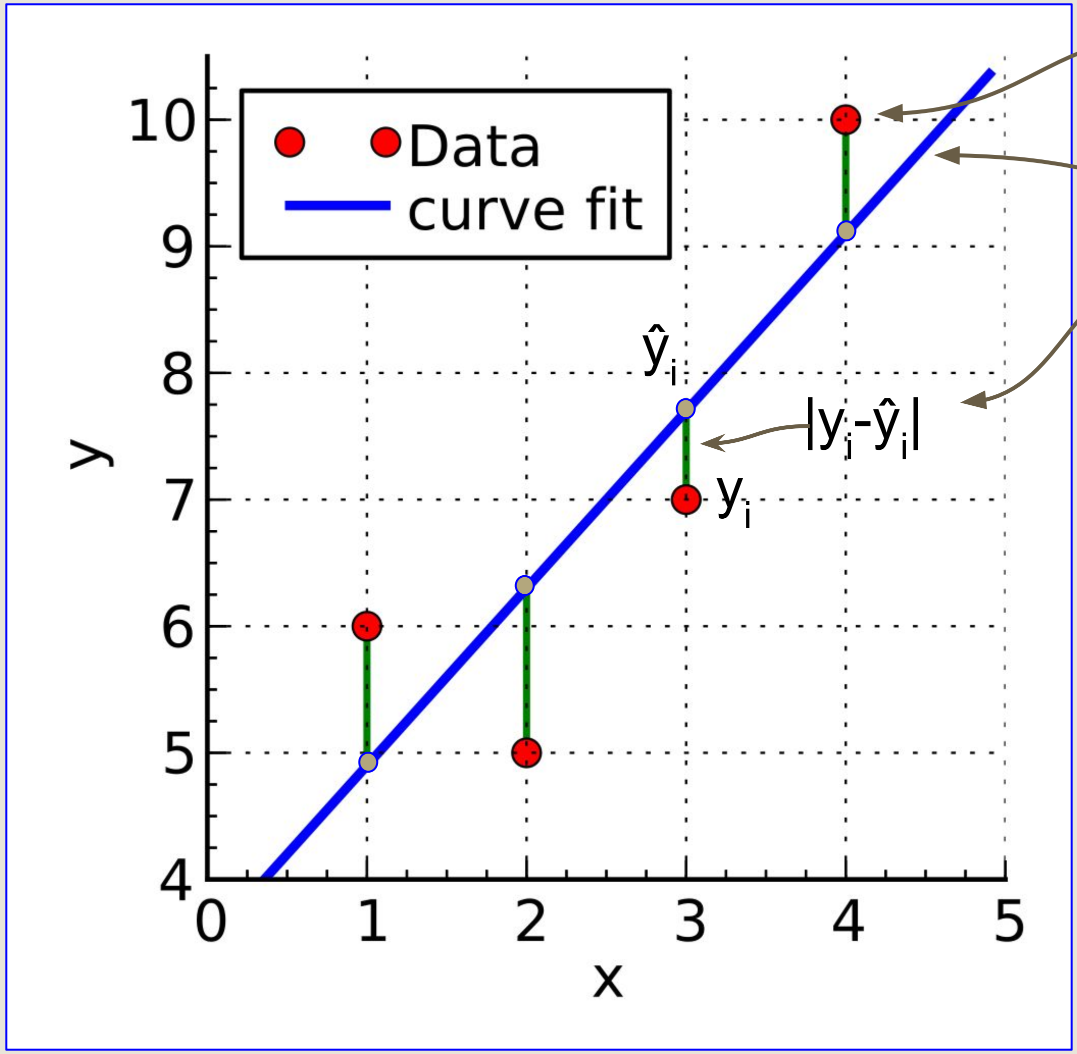
Example (Linear Regression)



Example (Regression)



Example (Regression) - Sum of least squares



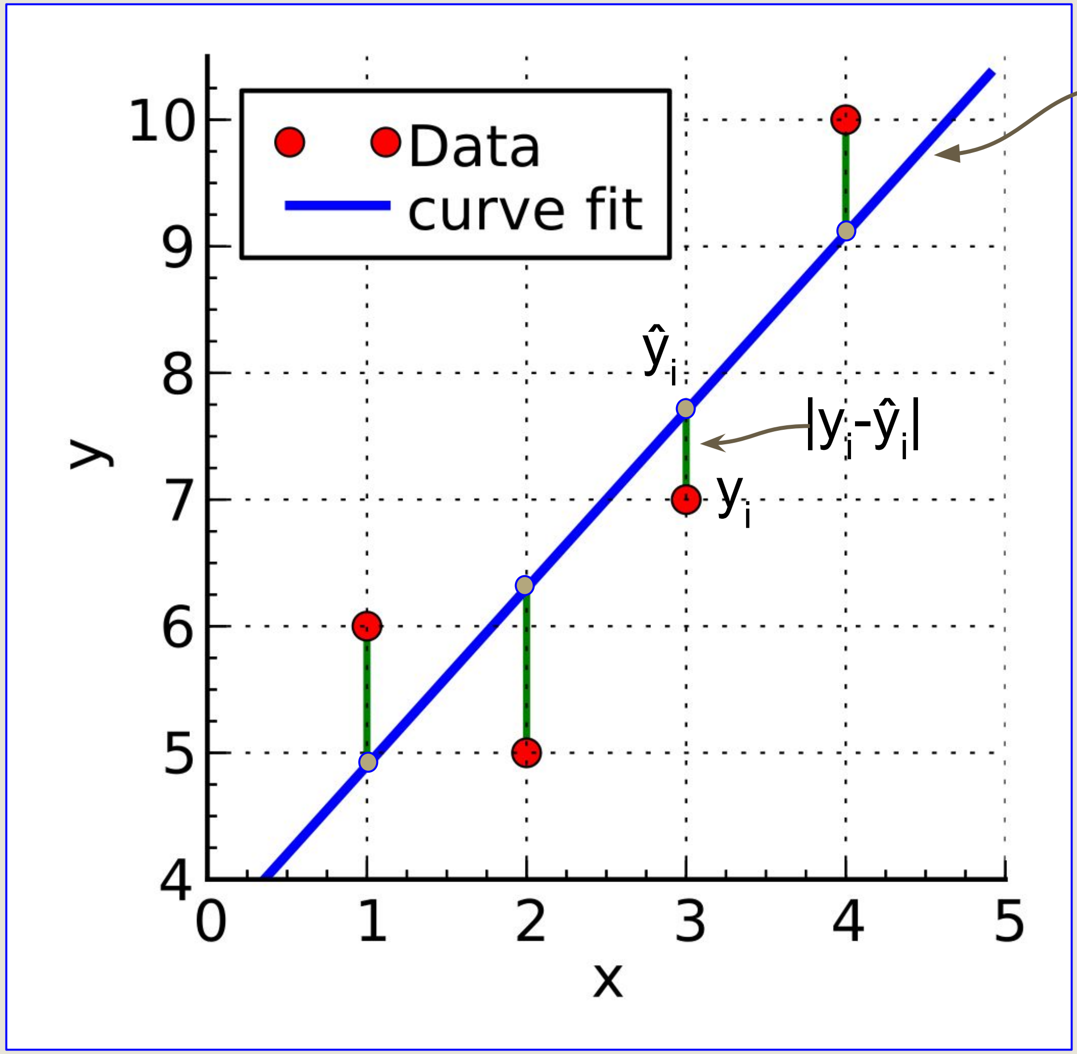
Actual: y_i

Prediction:

$\hat{y} = ax + b$

Error: $|y_i-\hat{y}_i|$

Example (Regression) - Sum of least squares



Prediction:

$\hat{y} = ax + b$

Actual: y_i

Error: $|y_i - \hat{y}_i|$

Total Squared Error:

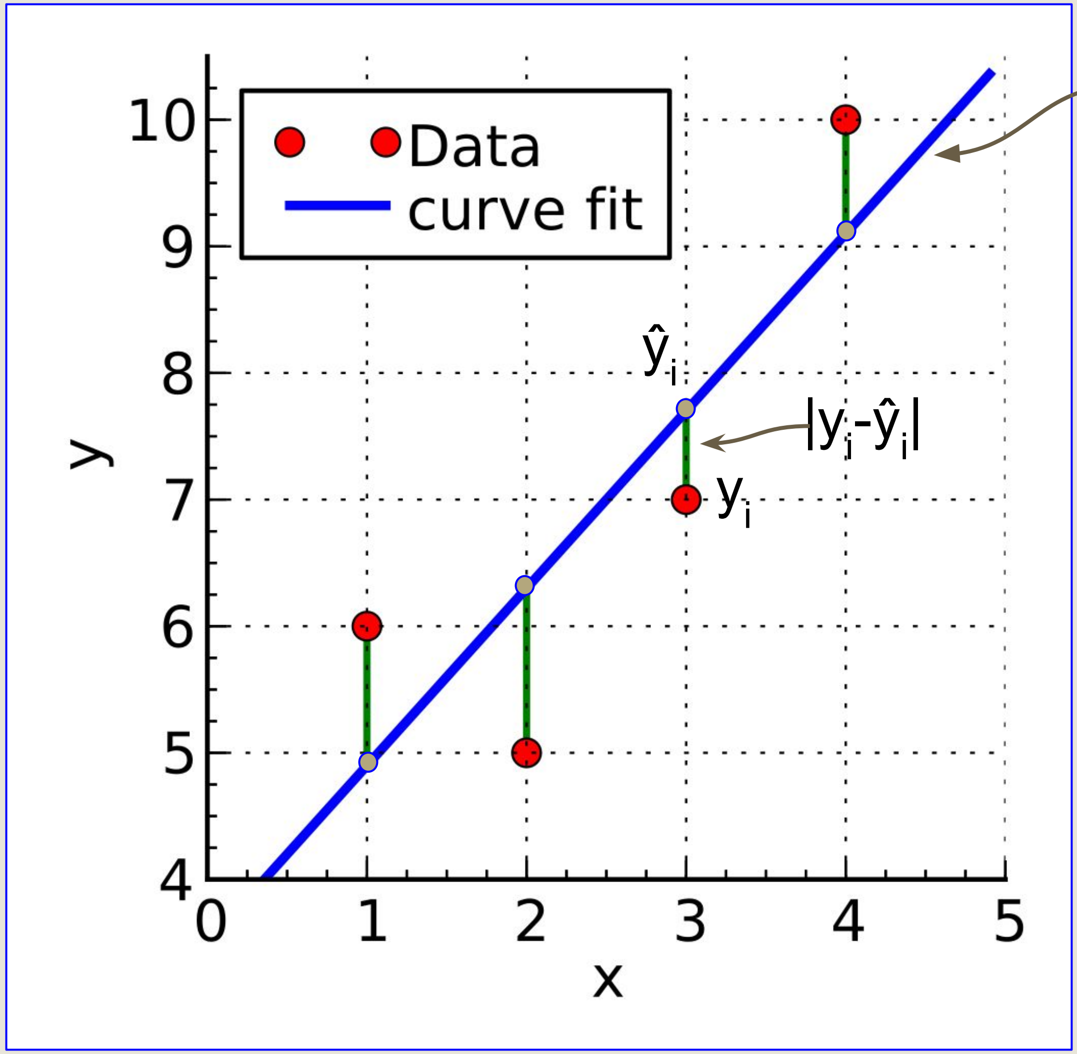
$\sum (y_i - \hat{y}_i)^2$, for i=(1, n)

Minimize Total Squared Error:

$Err(a,b) = \sum(y_i - ax_i - b)^2$

(a,b) are the parameters (weights)

Example (Regression) - Sum of least squares



Prediction:

$\hat{y} = ax + b$

Actual: y_i

Error: $|y_i - \hat{y}_i|$

Total Squared Error:

$\sum (y_i - \hat{y}_i)^2$, for i=(1, n)

Minimize Total Squared Error:

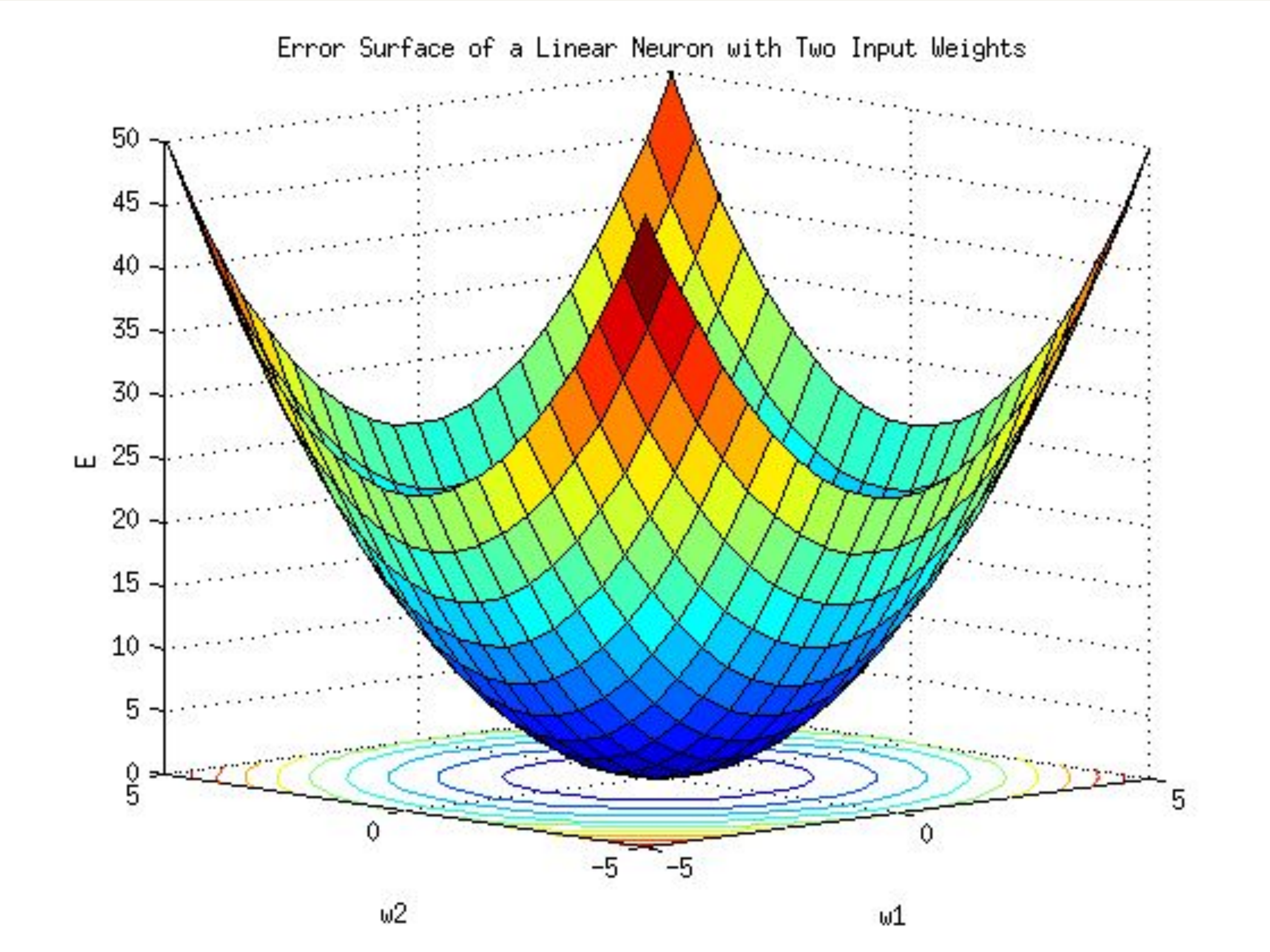
$Err(a,b) = \sum(y_i - ax_i - b)^2$

(a,b) are the parameters (weights)

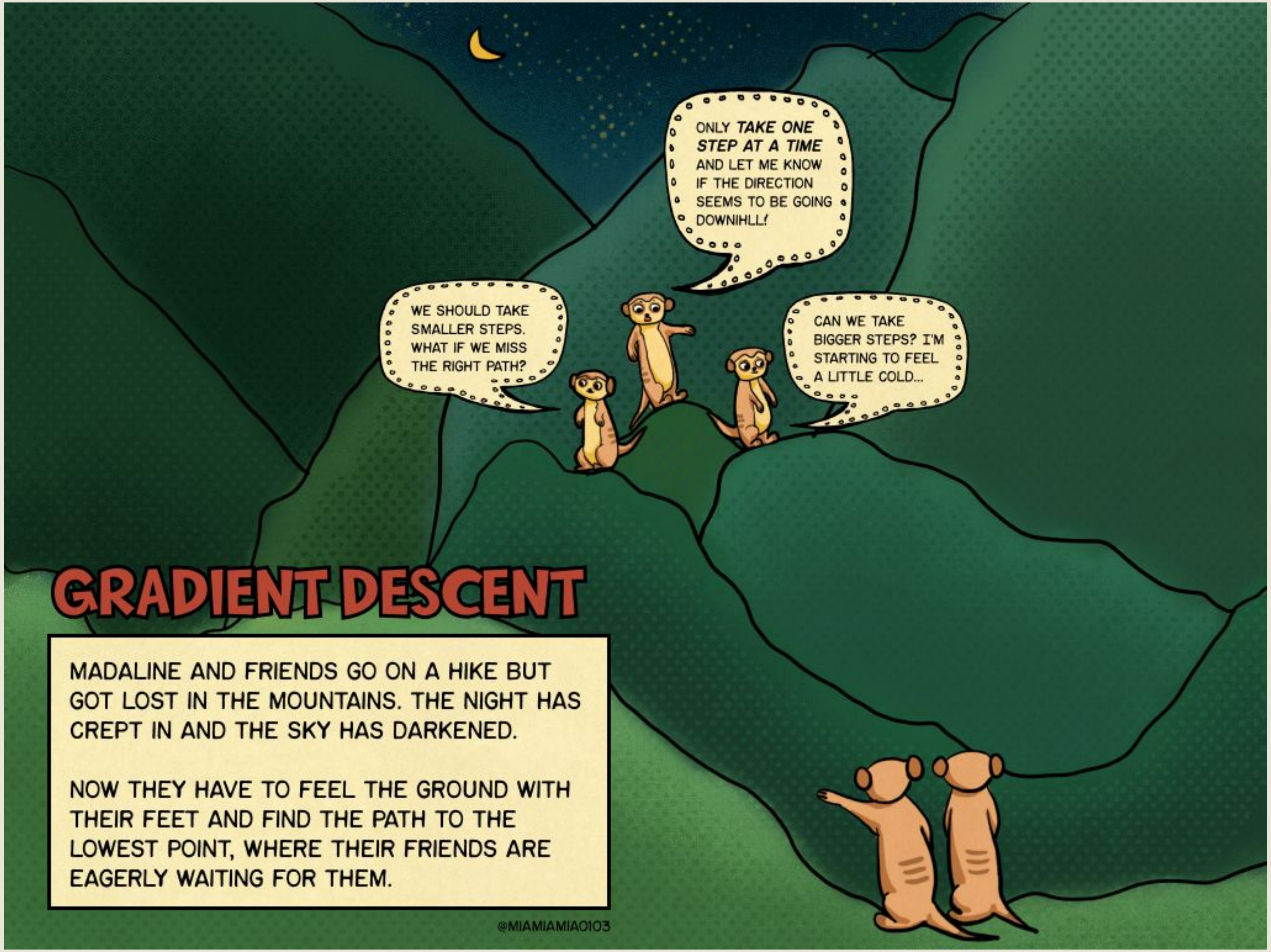
Regression - Minimize Error (Cost) via Gradient Descent

Minimize Error:

$$\sum(y_i - ax_i - b)^2$$



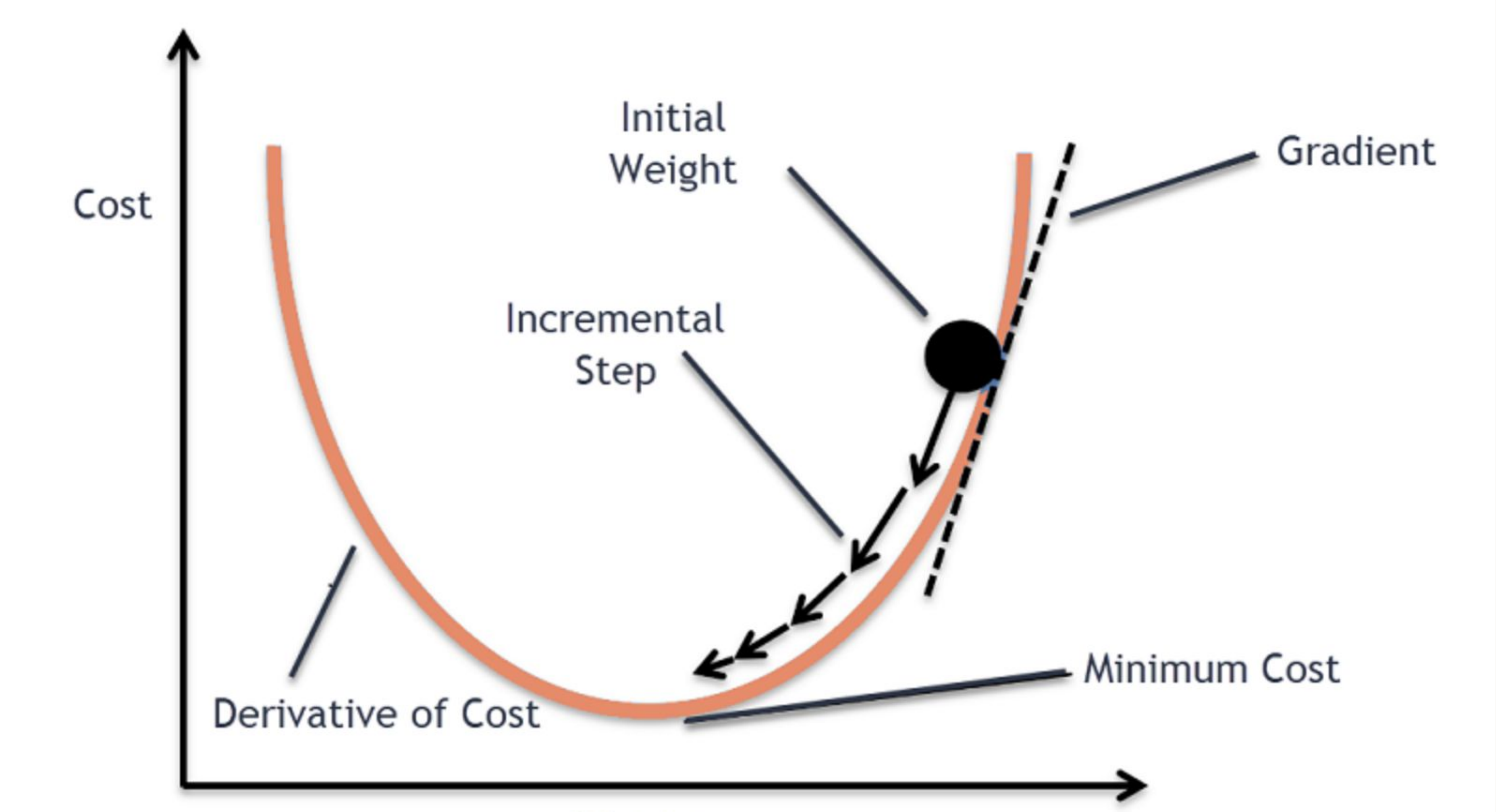
Regression - Minimize Error (Cost) via Gradient Descent



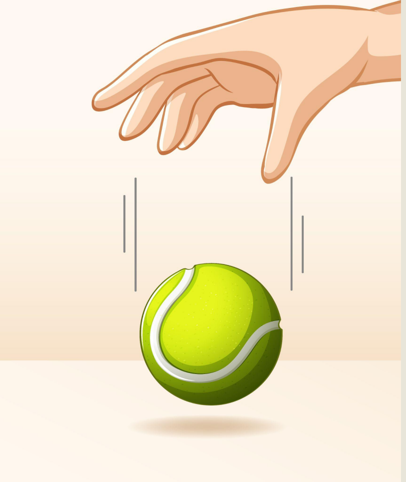
Regression - Minimize Error (Cost) via Gradient Descent

Minimize Cost:

$$\sum(y_i - ax_i - b)^2$$



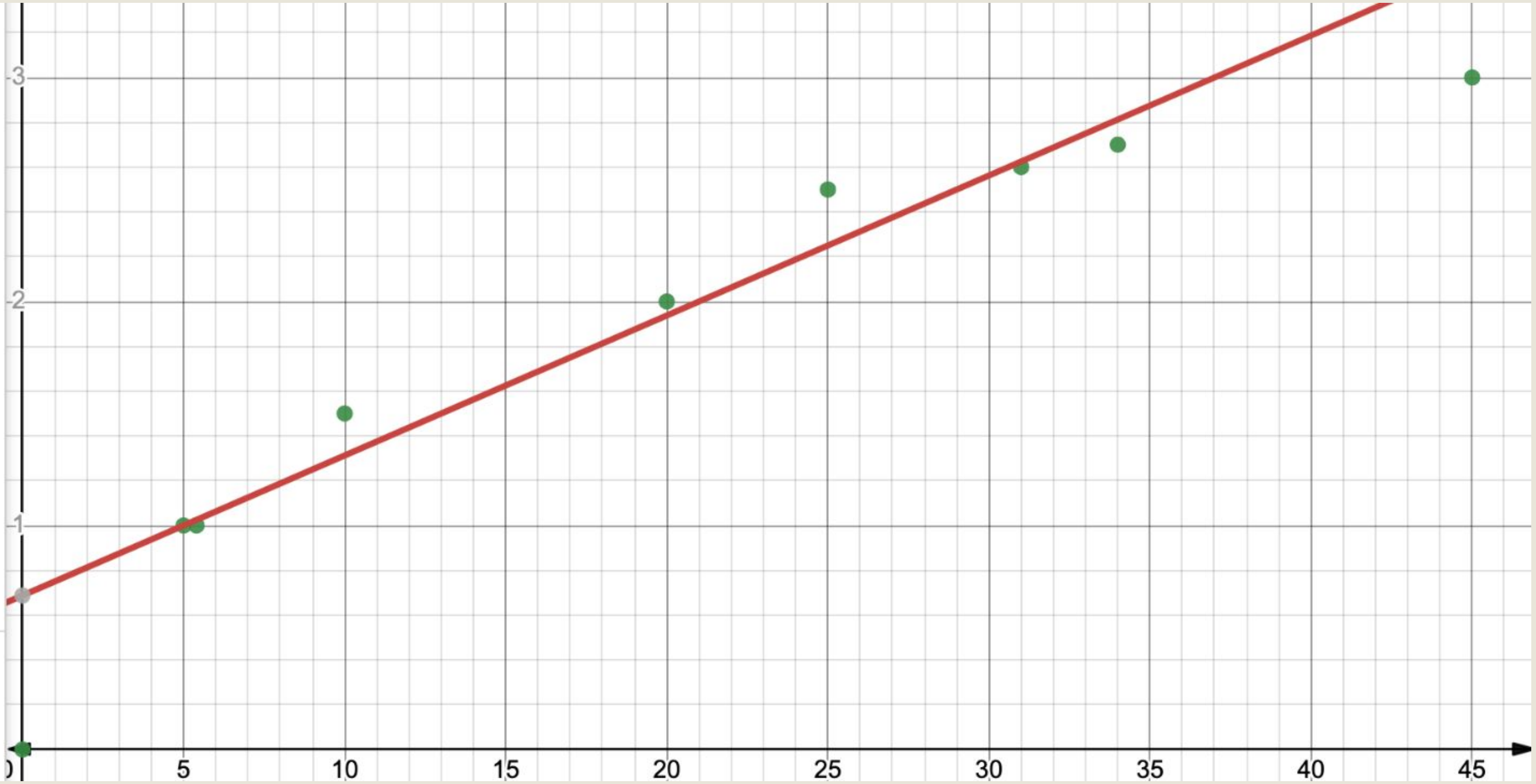
Example: Measuring Gravity



Measuring Gravity

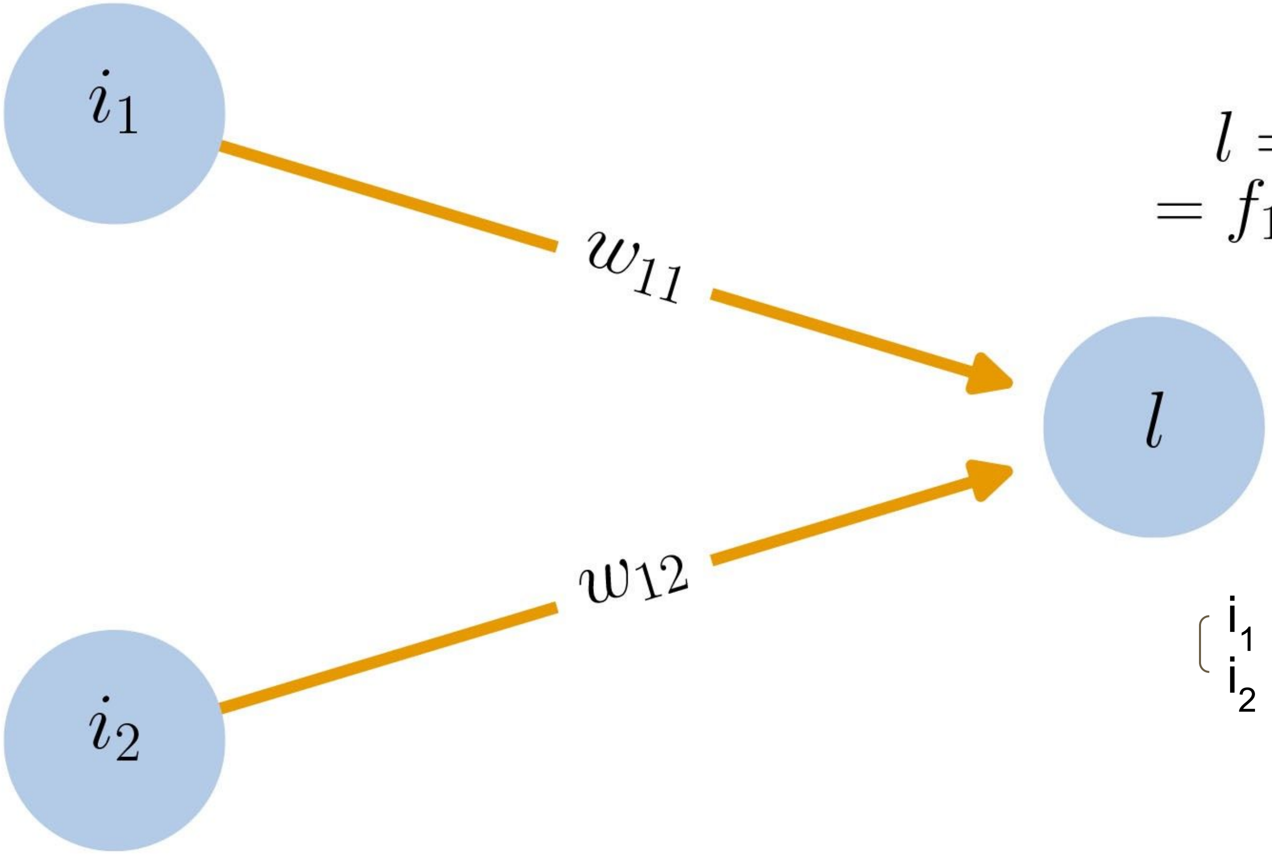
0	0
5	1
5.4	1
20	2
25	2.5
31	2.6
34	2.7
45	3
10	1.5

t = 0.06*d - 0.7



Linear Regression - Is this Machine Learning?

Linear function as a Network & a Matrix op



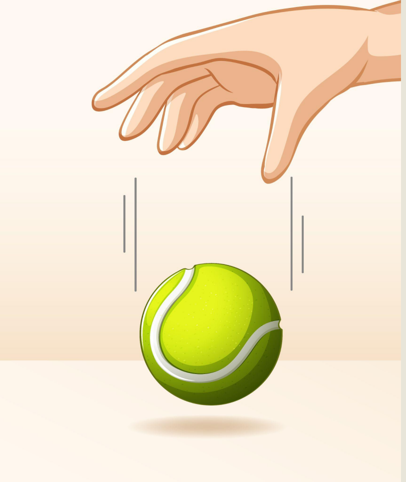
$$l = w_{11} \times i_1 + w_{12} \times i_2$$
$$= f_1(i_1, i_2)$$

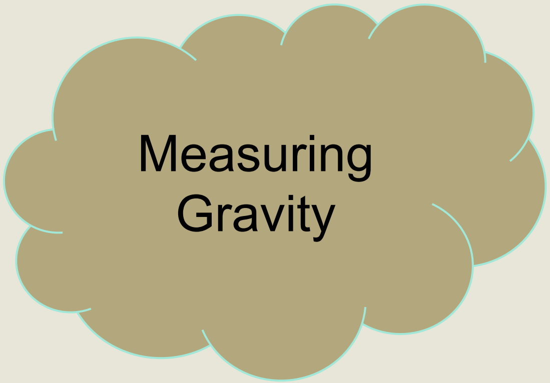
Line: $y = a + b*x$

$i_1=x$, $i_2=1$, $w_{11}=b$, $w_{12}=a$

$$\begin{bmatrix} i_1 \\ i_2 \end{bmatrix} \cdot \begin{bmatrix} w_{11} \\ w_{12} \end{bmatrix}$$

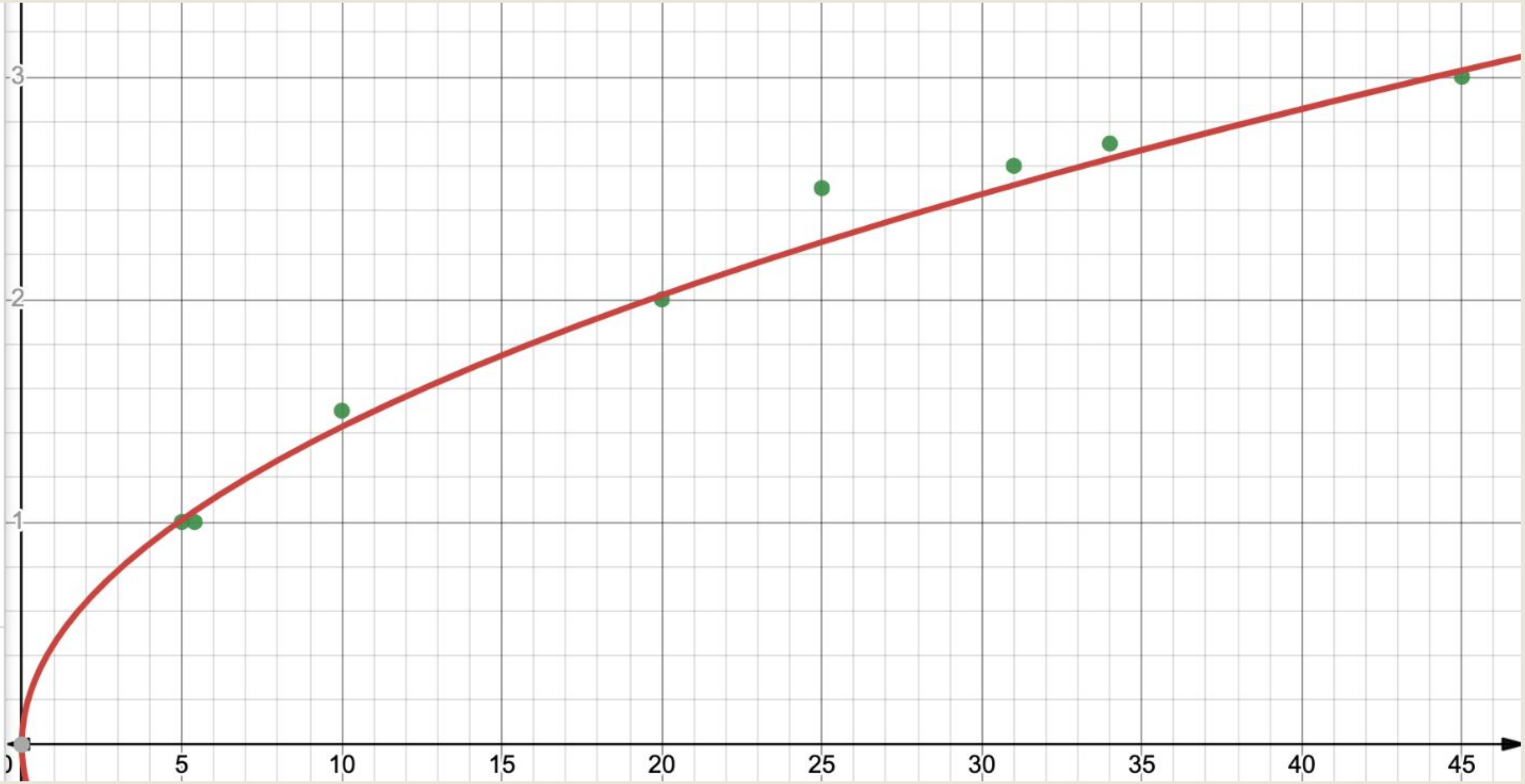
Example: Measuring Gravity Non-Linear Relationship!



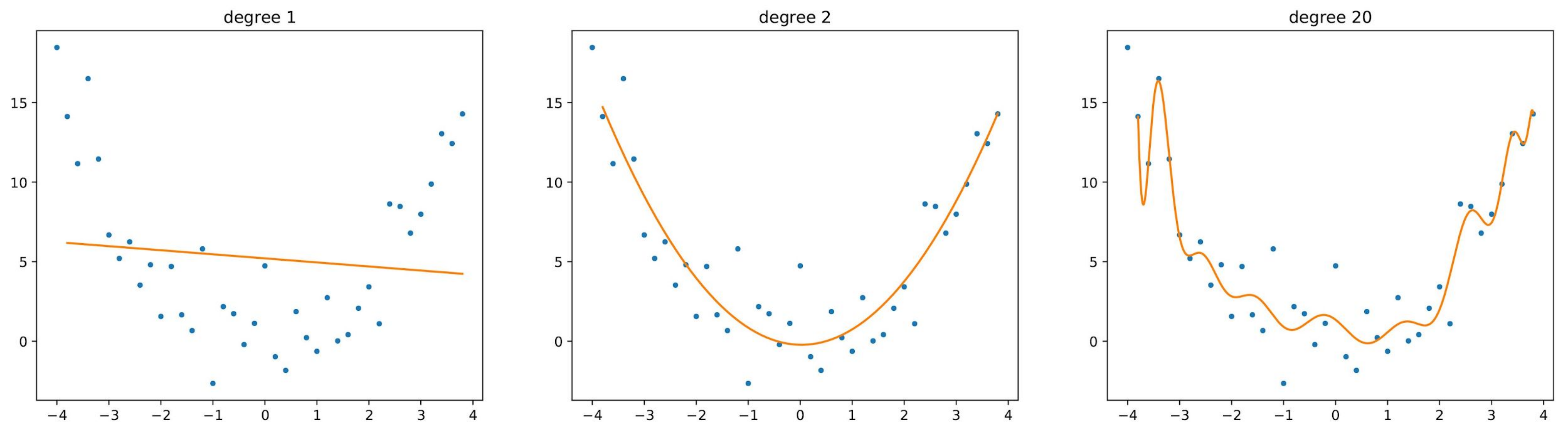


0	0
5	1
5.4	1
20	2
25	2.5
31	2.6
34	2.7
45	3
10	1.5

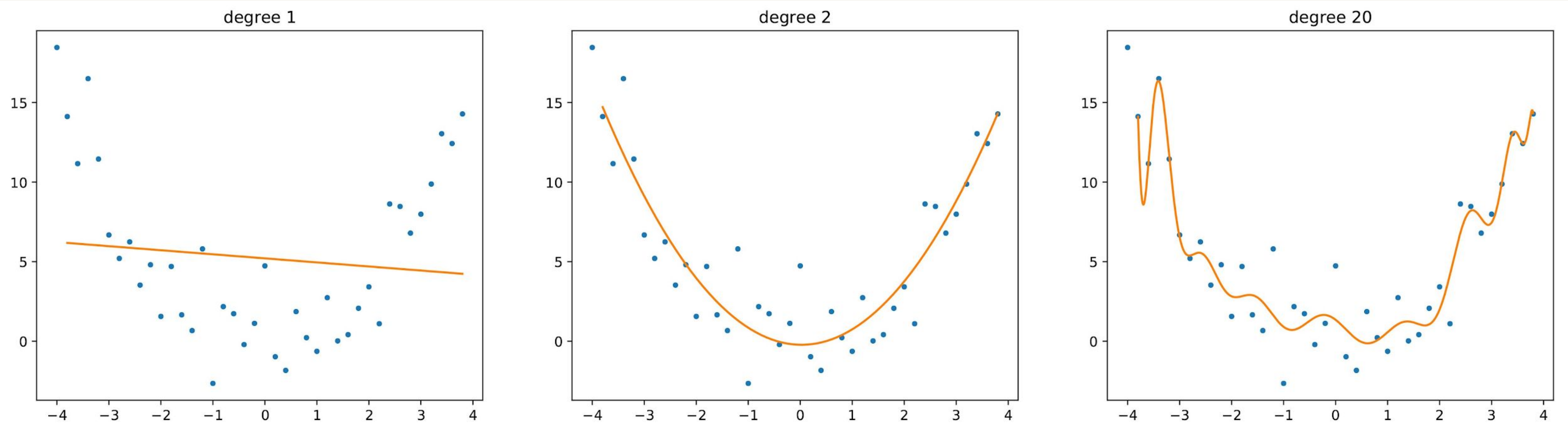
dist = 0.5*9.81*t*t



Example



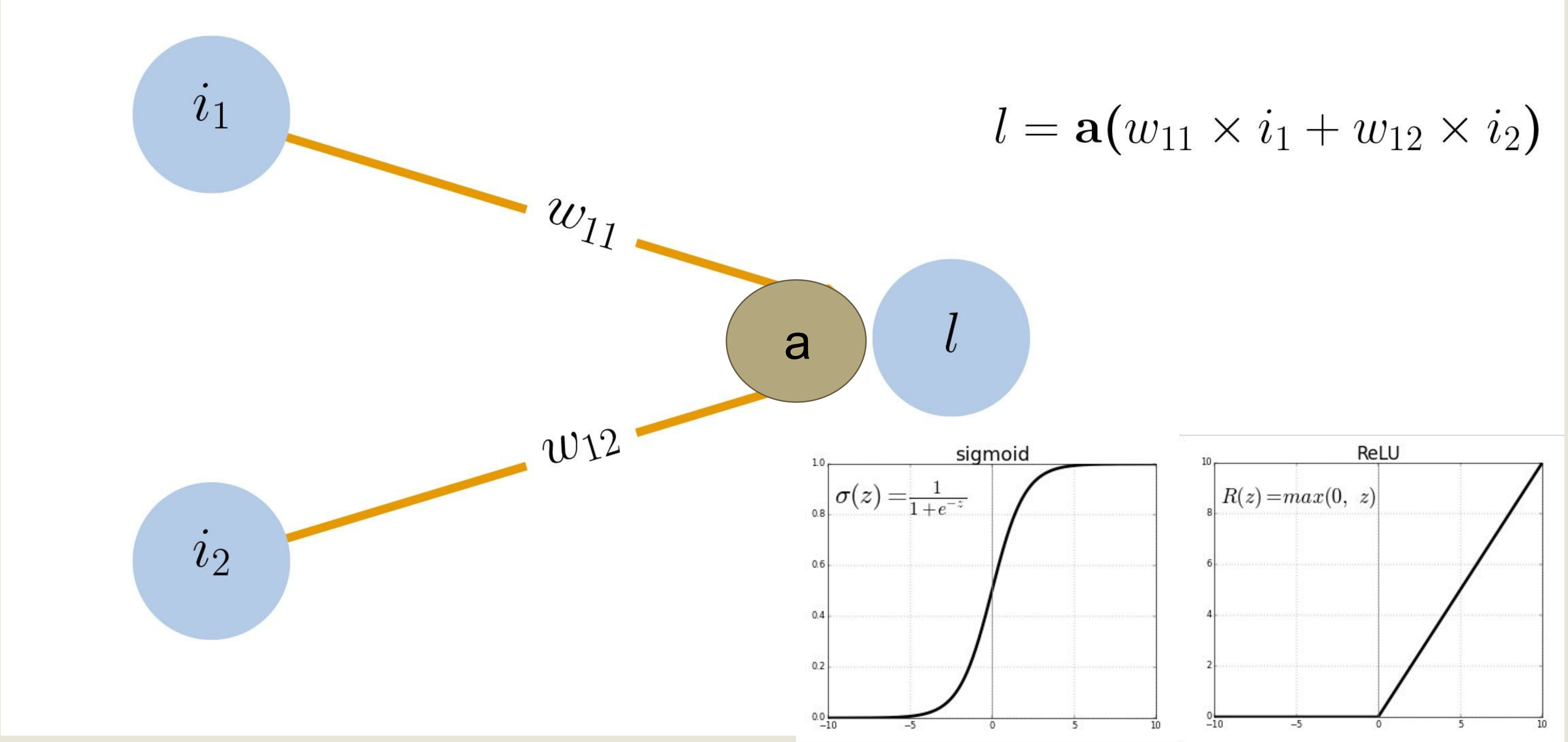
Example



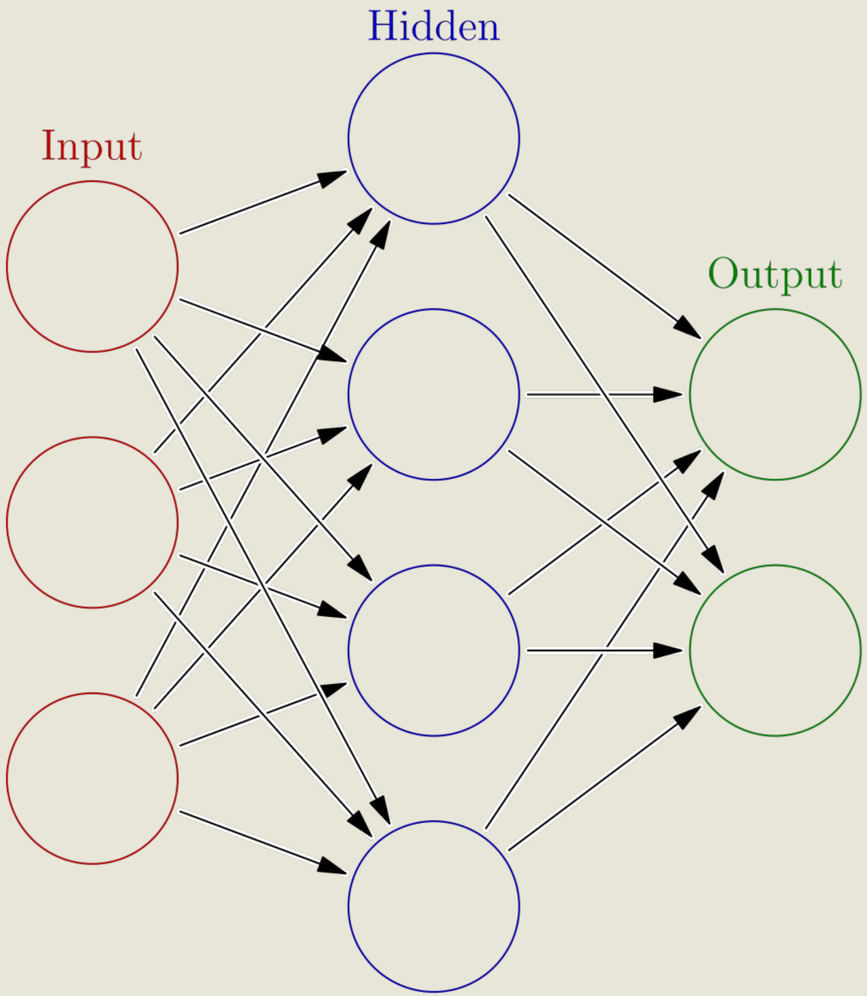
BIAS
(UNDERFIT)

VARIANCE
(OVERFIT)

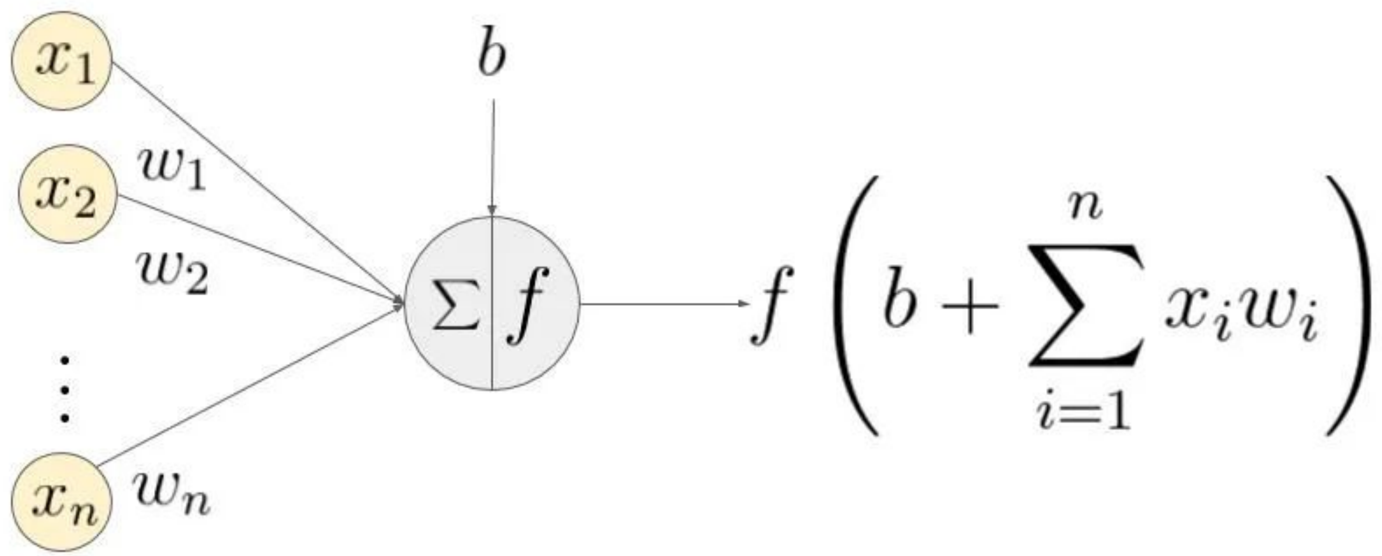
Adding non-linearity via an activation function



Adding complexity via a layer:



Forward Pass: Generalizing for 1-Node



An example of a neuron showing the input (x_1 - x_n), their corresponding weights (w_1 - w_n), a bias (b) and the activation function f applied to the weighted sum of the inputs.

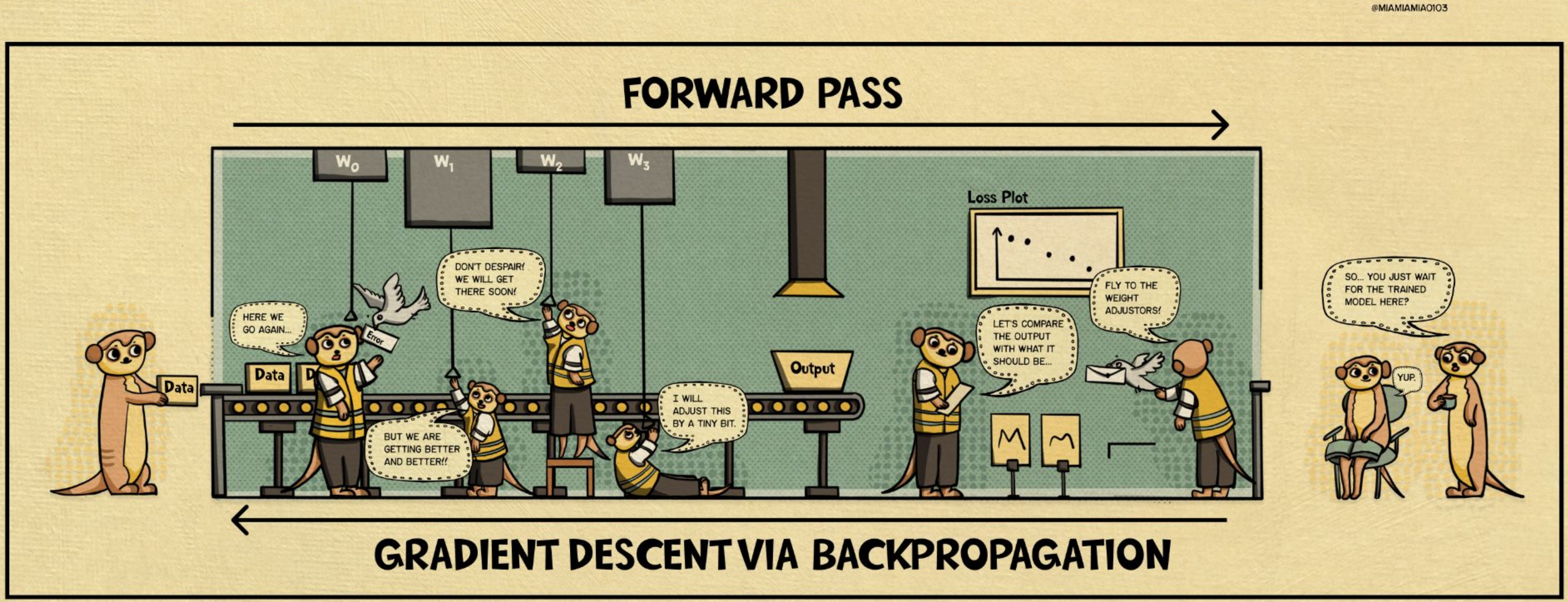
Universal Approximation Theorem:

In the mathematical theory of artificial neural networks, the **universal approximation theorem** states[1] that a feed-forward network with a single hidden layer containing a finite number of neurons can approximate arbitrary well real-valued continuous functions on compact subsets of $\mathbf{R}^n$.

But, No Free Lunch Theorem:

For optimization problems... if an algorithm performs well on a certain class of problems then it necessarily pays for that with degraded performance on the set of all remaining problems.

NEURAL NETWORK OVERVIEW



Solving the network

- *Set the initial weights of the network randomly*
- *Make a forward pass through the network and compute output*
- *Compare the output with expected result and compute loss*
- *Change the weights by a small amount (Gradient descent via back prop)*
- *Repeat* until desired minimization of error (cost) is achieved

Try it yourself:

- *Colab: Jupyter derived python IDE in the cloud*
- Software and tools:
  - Python 3.x - programming
  - Tensorflow 2.1.0 - machine learning
  - Numpy - numerical mathematics, linear algebra
  - Pandas - data analysis
  - Matplotlib - plotting
  - Seaborn - advanced plotting

Files:

- Housing.ipynb,
- HousingRegression.ipynb,
- FlowerClassification.ipynb

ML Problem Solving Process:

- FRAMING: What is observed & what answer you want to predict
- DATA COLLECTION: Collect, clean, and prepare data
- DATA ANALYSIS: Visualize & analyze the data
- FEATURE PROCESSING: Transform raw data for better predictive input
- MODEL BUILDING: Design and build the learning algorithm
- TRAINING: Feed data to the model and **evaluate** the quality of the models
- PREDICTION: Use model to generate **predictions** for new data instances

ML project - process - apply to housing problem 1/7

- *FRAMING*: what is observed & what answer you want to predict

Observed: parameters related to homes for a census block

Predict: Median home-price for the block

Housing project steps - 2/7

- *DATA COLLECTION*: Collect, clean, and prepare data

Already given in a csv file: `housing.csv`

`isna(), np.where(), dropna()`

Housing project steps - 3/7

- *DATA ANALYSIS*: Visualize & analyze the data
    - `sns.pairplot(...)`
    - `data.describe()`

Housing project steps - 4/7

- *FEATURE PROCESSING*: Transform raw data for better predictive input

-- Normalize `(x-x.min())/(x.max()-x.min())` : brings data between 0 and 1

-- Standardize `(x-x.mean())/x.std()`: remaps to mean of 0, and std_dev of 1

-- Keep 20% for testing:

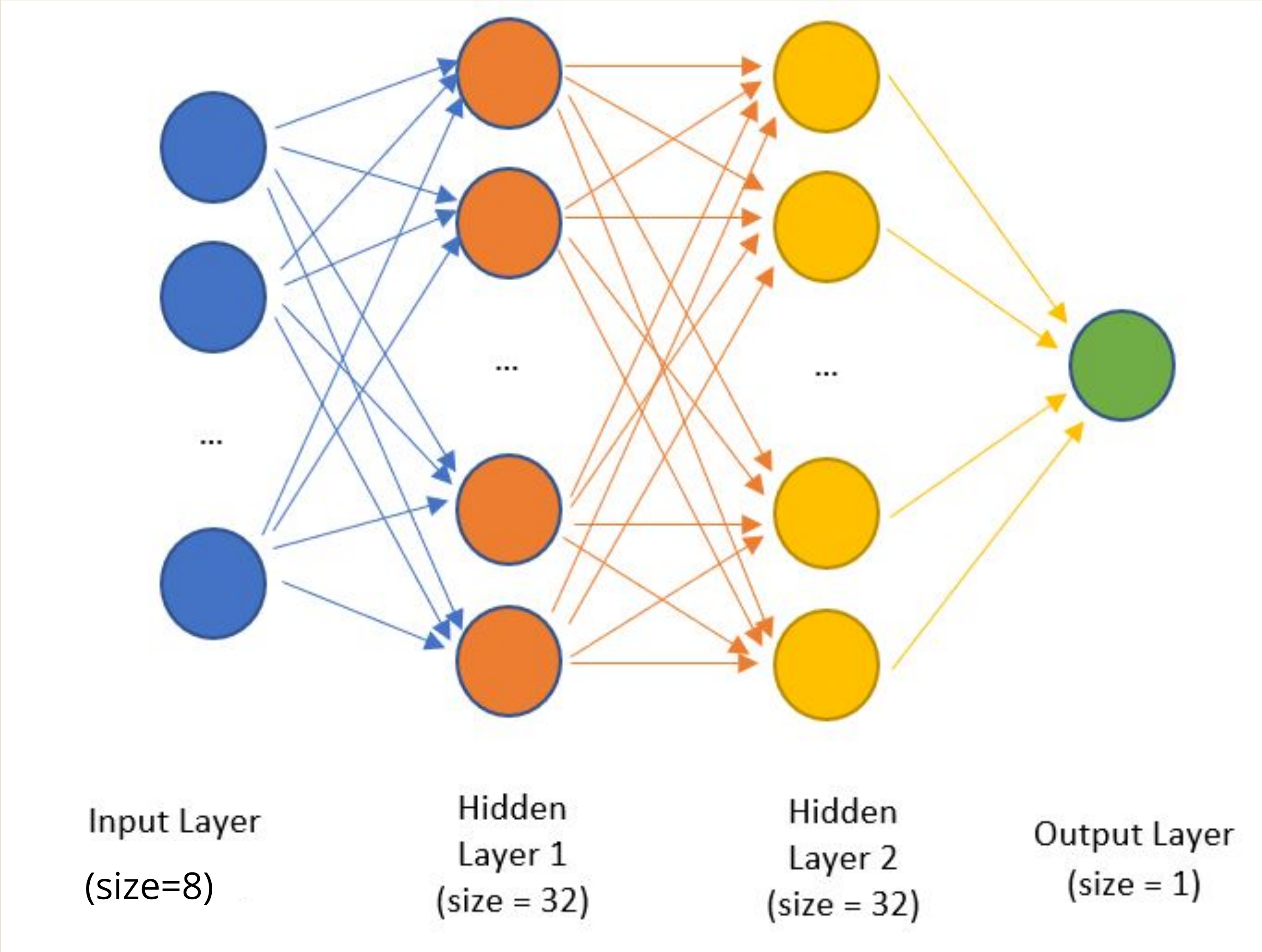
```
train=data.sample(frac=0.8)

test=data.drop(train.index)
```

-- Separate features (x) from labels (y):

```
X_train = train.drop('median_house_value', axis=1)

Y_train = train['median_house_value']
```

Housing project steps - 5/7

- *MODEL BUILDING*: Feed features to learning algorithm to build models



Housing project steps - 5/7

- *MODEL BUILDING*: Feed features to learning algorithm to build models

```
import tensorflow as tf

INPUT_SHAPE=[9]

model = tf.keras.Sequential([
    tf.keras.layers.InputLayer(INPUT_SHAPE, name="Input_Layer"),
    tf.keras.layers.Dense(32, activation='relu', name="dense_01"),
    tf.keras.layers.Dense(32, activation='relu', name="dense_02"),
    tf.keras.layers.Dense(1, name="Output_Layer")
  ])

model.compile(loss='mse',
              optimizer=tf.keras.optimizers.RMSprop(0.001),
              metrics=['mae', 'mse'])

print(model.summary())
```

Housing project steps - 6/7

- *TRAINING*: compute weights and Evaluate the quality of the models

```
example_batch = x_train[:10]
example_result = model.predict(example_batch)
print(example_result)

history = model.fit(x_train, y_train,
                    batch_size=32,
                    epochs=10,
                    validation_split=0.2,
                    verbose=1)

# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validate'], loc='upper left')
plt.show()

loss, mae, mse = model.evaluate(x_test, y_test, verbose=2)
print("Loss:", loss, " mae:", mae, " mse:", mse)
```

Summarizing

- *Given: Features (X, Attributes), Output (Y, Labels, Ground Truth): Y=f(X)*
- *Network (Model)*
- *Loss Function (Metric, Cost)*
- *Activation Function (adds non-linearity, Ex: sigmoid, ReLU)*
- *Training (fit, Optimization to minimize Loss Function)*
- *Evaluate (performance, correctness)*
- *Predict (Inference, on new data)*

Housing project steps - 7/7

- *PREDICTION*: Use model to generate predictions for new data instances

```
p_test = model.predict(x_test)
print(p_test, y_test)

a = plt.axes(aspect='equal')
plt.scatter(y_test, p_test)
plt.xlabel('True Values')
plt.ylabel('Predictions')
lims = [0, 1]
plt.xlim(lims)
plt.ylim(lims)
plt.plot(lims, lims)
plt.show()

error = p_test.flatten() - y_test
print(error)
plt.hist(error, bins = 25)
plt.xlabel("Prediction Error")
plt.ylabel("Count")
plt.show()
```

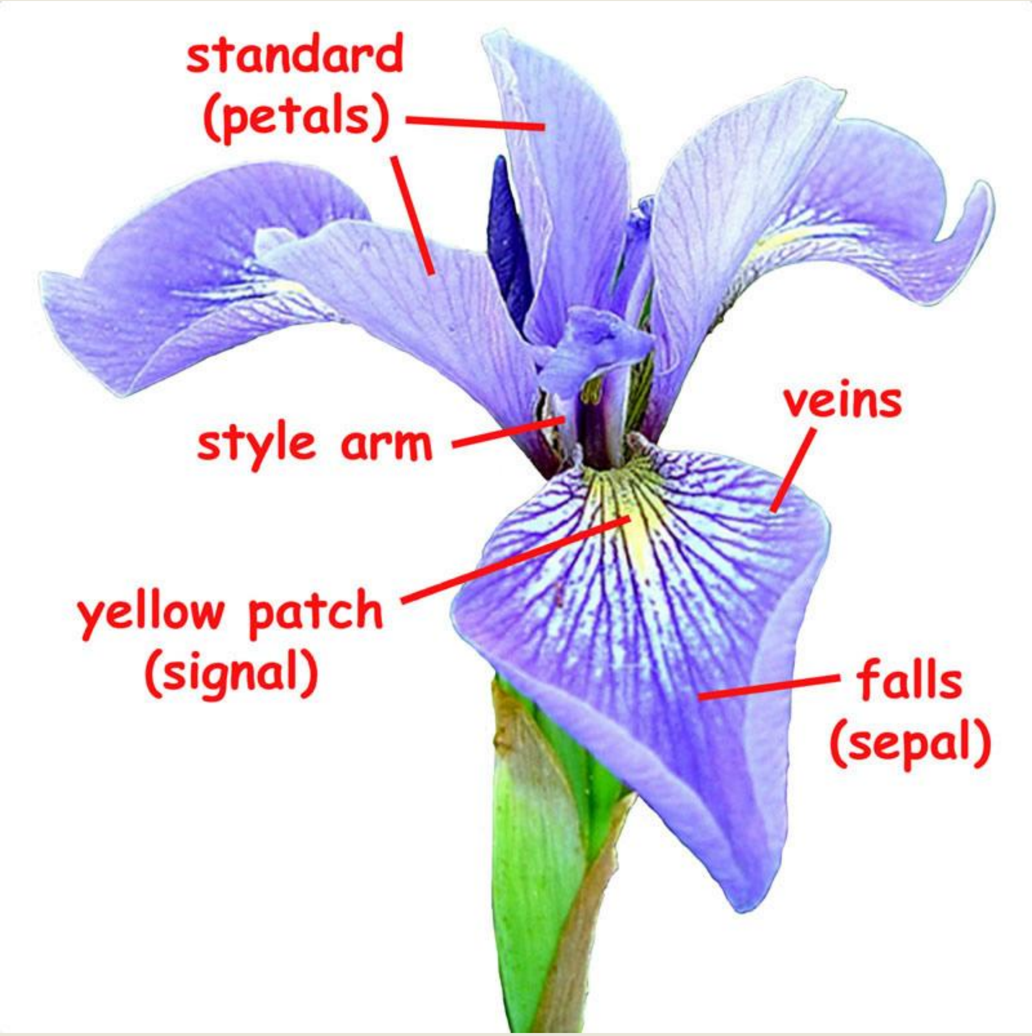
Housing project steps - Improving Results

- *Tune* hyperparameters
    - Increase/Decrease # of epochs
    - Try different batch sizes: 16
    - Try different learning rates (0.01, 0.000001)
    - Try different optimizers (‘adam’)
    - Try different loss functions (‘mse’, ‘mae’)
    - Make the network deeper (layers) or denser (nodes per layer)

Flower Project - Classification

- *FRAMING*: what is observed & what answer you want to predict
  - Given data: about 3 species of iris flowers
- *DATA COLLECTION*: Collect, clean, and prepare data
- *DATA ANALYSIS*: Visualize & analyze the data
- *FEATURE PROCESSING*: Transform raw data for better predictive input:
- *MODEL BUILDING*: Feed features to learning algorithm to build models
- *TRAINING*: Evaluate the quality of the models
- *PREDICTION*: Use model to generate predictions for new data instances

Flower Project - Classification



Flower Classification: get data ready

120	4	setosa	versicolor	virginica
0	6.4	2.8	5.6	2.2	2
1	5.0	2.3	3.3	1.0	1
2	4.9	2.5	4.5	1.7	2
3	4.9	3.1	1.5	0.1	0
4	5.7	3.8	1.7	0.3	0

Given: Features about Iris: sepal length, sepal width, petal length, petal width
Task: Classify into kind of Iris: setosa (0), versicolor (1), or virginica (2)

- Plot data
- Split data for training and testing
- Normalize training data

Flowers - Classification - get data ready

```
#----------DATA READING
filename = 'https://storage.googleapis.com/download.tensorflow.org/data/iris_training.csv'
# read file
csv_data = pd.read_csv(filename, sep= ',')
print(csv_data.head())

column_names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'species']
class_names = ['Iris setosa', 'Iris versicolor', 'Iris virginica']

#----------DATA CLEANUP
csv_data.columns = column_names  # new_header --set the header row as the data header
print(csv_data.head())
# look at simple data statistics
print(csv_data.describe().transpose())
# plot of all features against each other
sns.pairplot(csv_data)
```

Flowers - Classification - get data ready

```
 #----------TRAIN/TEST SPLIT
train_data = csv_data.sample(frac= 0.8) # take 80% randomly from the data for training
test_data = csv_data.drop(train_data.index)  # reserve the rest for testing

# separate out the y (results) from x (features) for training
x_train = train_data.drop( 'species', axis=1)
y_train = train_data[ 'species']
# normalize the training data
x_train = (x_train-x_train.min())/(x_train.max()-x_train.min())

# separate out the y (results) from x (features) testing
x_test = test_data.drop( 'species', axis=1)
y_test = test_data[ 'species']
# normalize the test data
x_test = (x_test-x_test.min())/(x_test.max()-x_test.min())

print('Training Data\n', x_train.describe().transpose())
print('Test Data\n', x_test.describe().transpose())
```

Flowers - Classification steps - model

```
#--------MODEL BUILDING
num_params = len(x_train.keys())
print(num_params)
model = tf.keras.Sequential([
   tf.keras.layers.InputLayer([num_params], name= "Input_Layer"),
   tf.keras.layers.Dense( 32, activation='relu', name="dense_01"),
   tf.keras.layers.Dense( 32, activation='relu', name="dense_02"),
   # 1 node in the output for the median_house_vale
   tf.keras.layers.Dense( 3, name="Output_Layer")
 ])

model.compile(optimizer=tf.keras.optimizers.RMSprop( 0.001),
              # loss function to minimize
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits= True),
              # list of metrics to monitor
             metrics=[ 'acc',])

model.summary()
```

Flowers - Classification Log Likelihood

```
Note:  Loss Function:  SparseCategoricalCrossentropy(from_logits= True)
```

- Instead of a value, it returns a 'LOG LIKELIHOOD' for each output class
- Which we convert into a probability for each output class
- We take the class with the highest probability as the predicted class

Log Likelihood:

```
   class-A  class-B  class-C
[ 0.02669345  0.03092438 -0.01683718 ]
```

Convert to probabilites (using *softmax* function)

```
[ 0.13765042   0.739082   0.12326758 ]
```

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Output class: B (class with the maximum probability)

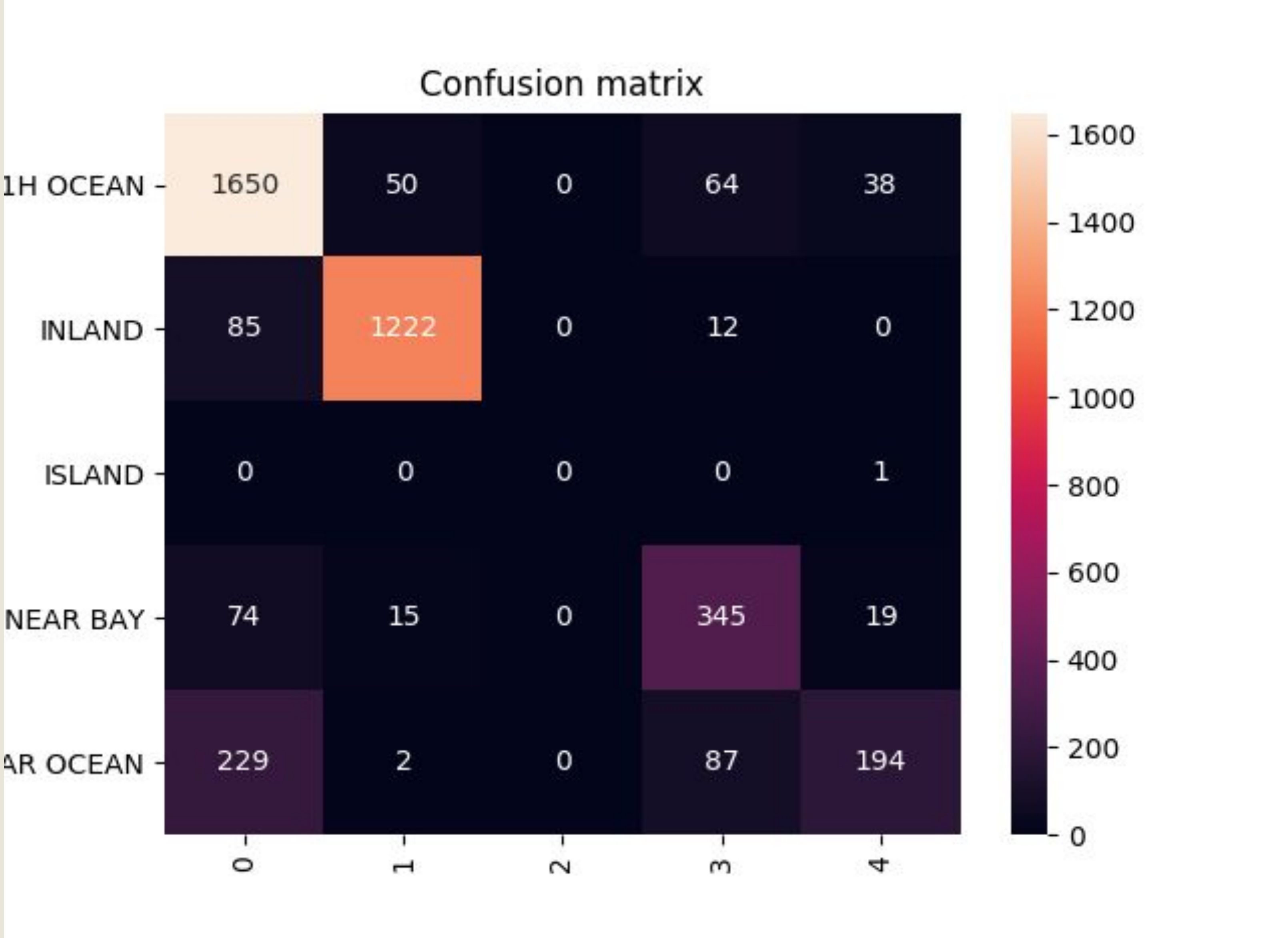
Flowers - Classification - train and test

```
# Fit/TRAIN model on training data
history = model.fit(x_train, y_train,
                    batch_size= 4,
                    epochs= 10,
                    validation_split= 0.2,
                    verbose= 1)
#--------MONITOR
# Plot training & validation loss values
fig = plt.figure(figsize=( 12,9))
plt.plot(history.history[ 'loss'])
plt.plot(history.history[ 'val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend([ 'Train', 'Validate'], loc='upper left')
plt.show()
```

Classification - Evaluation - Confusion Matrix

		Actual Value	
		positive	negative
Predicted Value	positive	TRUE POSITIVE	FALSE POSITIVE
	negative	FALSE NEGATIVE	TRUE NEGATIVE

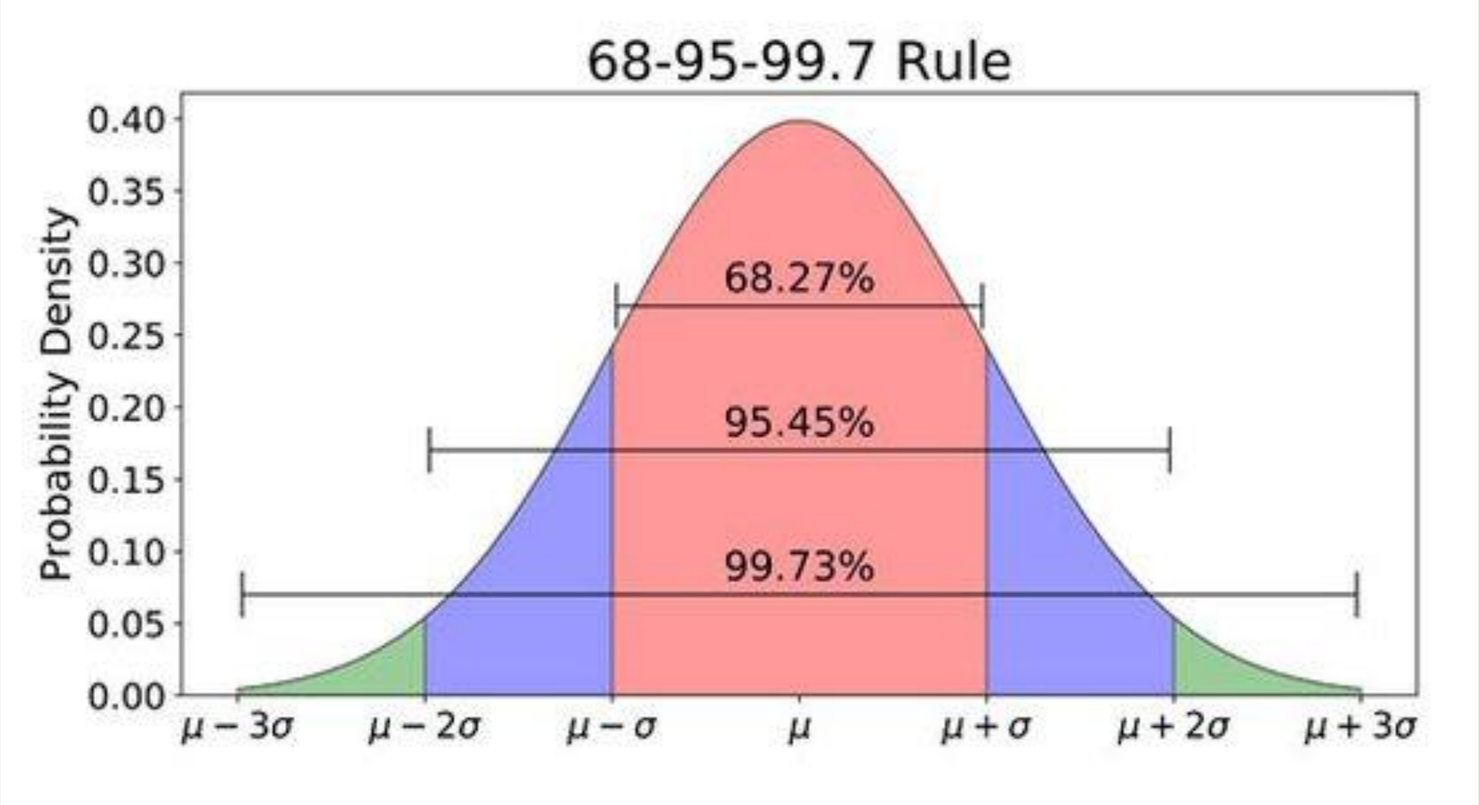
Classification - Evaluation - Confusion Matrix*



Classification - Evaluation - Confusion Matrix

```
# plot the confusion matrix as heatmap
sns.heatmap(tf.math.confusion_matrix(y_test,
            p_test_class), cmap="Blues"
            annot=True)
```

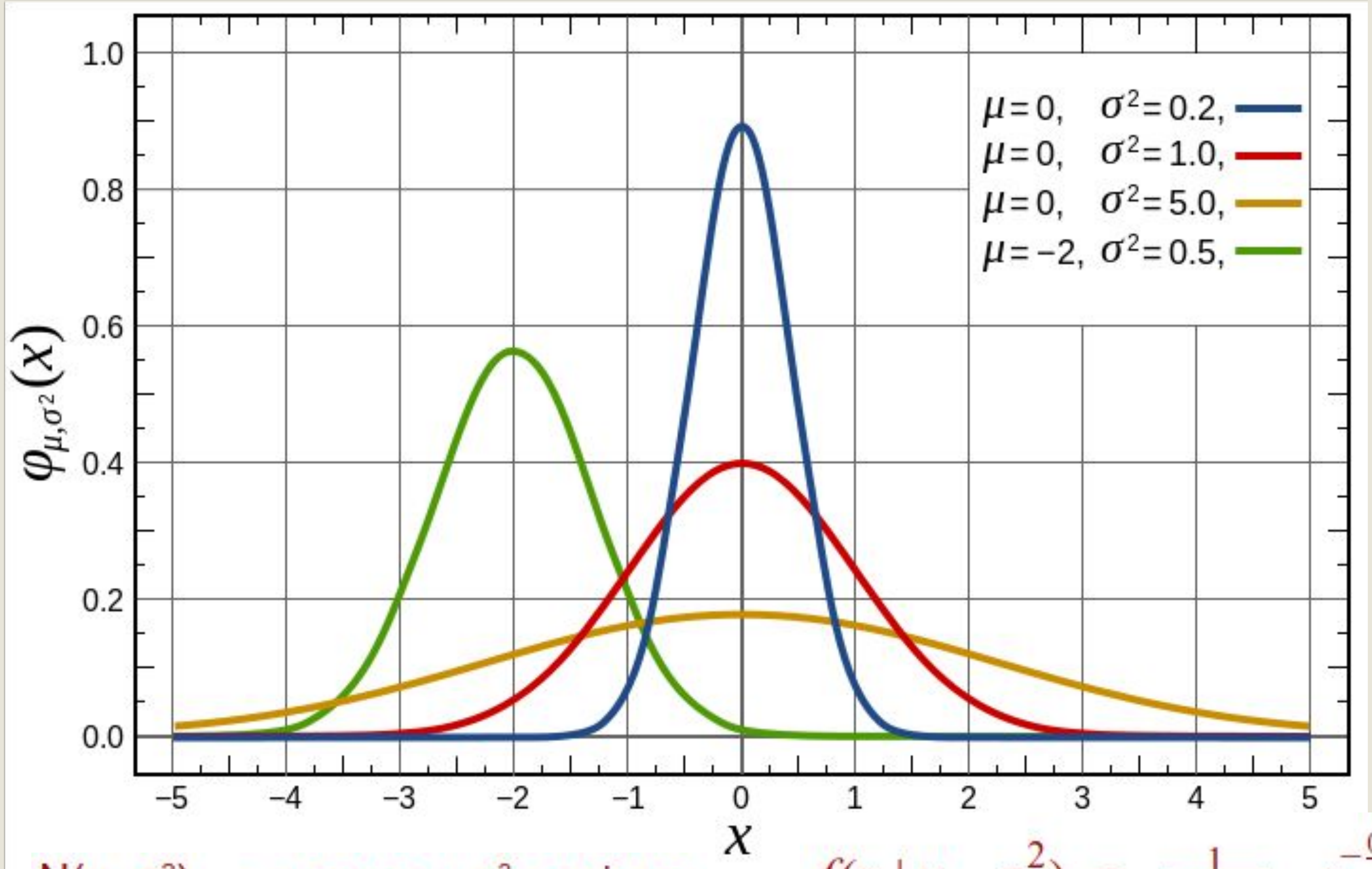
Probability & Statistics: Normal Distribution



$N(\mu, \sigma^2)$; μ - mean , σ^2 - variance

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Probability & Statistics: Normal Distribution



$N(\mu, \sigma^2)$; μ - mean , σ^2 - variance

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

MSE ---> Log Likelihood

Linear Regression Model

$$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$$

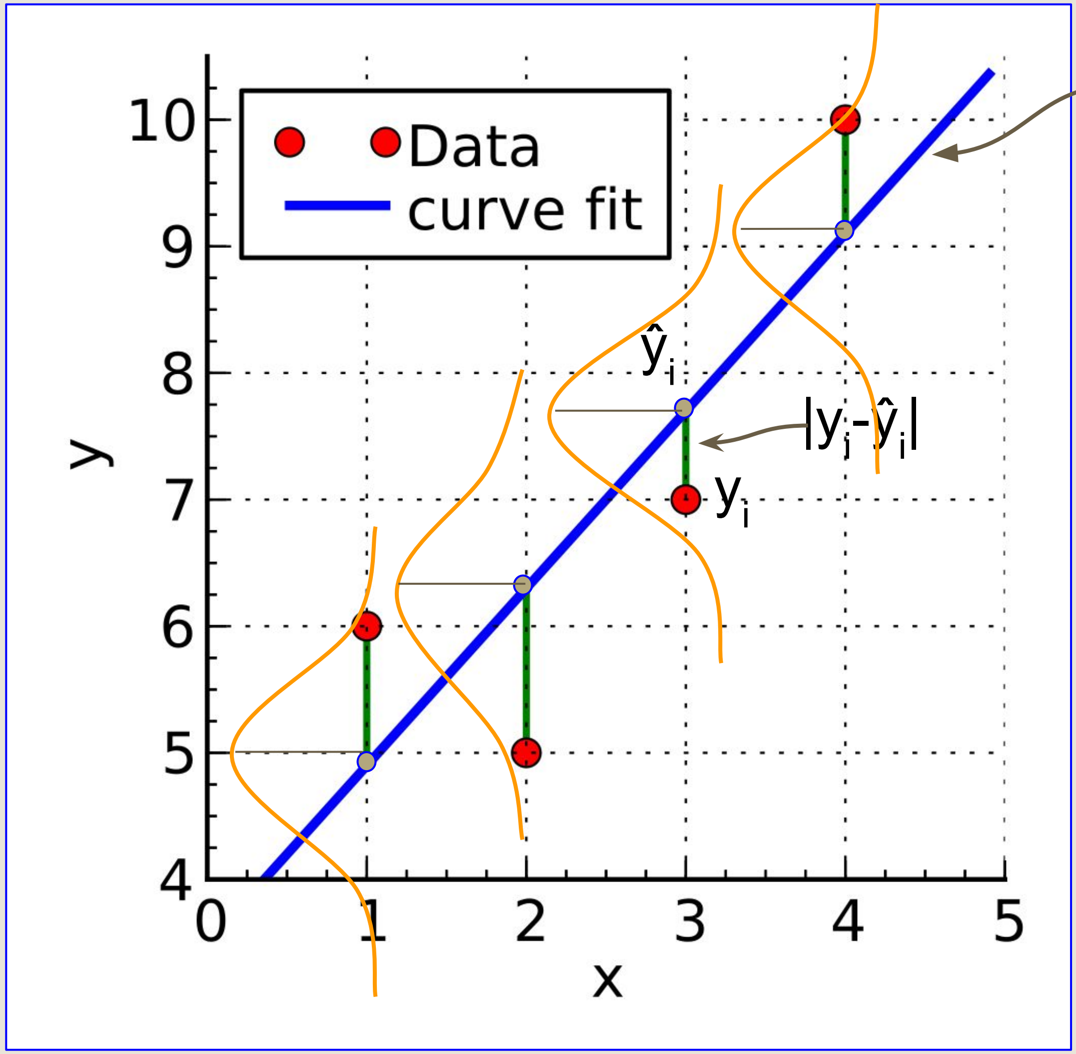
where $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$

This means that each Y_i is normally distributed around $\beta_0 + \beta_1 X_i$ with variance σ^2.

Probability Density Function (PDF) for a normal distribution $\mathcal{N}(\mu, \sigma^2)$ is:

$$f(y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right)$$

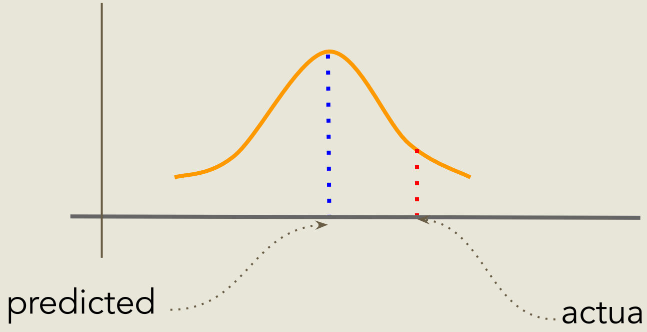
Regression → Maximum Likelihood Estimation



Prediction: $\hat{y} = ax + b$

Actual: y_i

Error: $|y_i - \hat{y}_i|$



MSE ---> Log Likelihood

The likelihood function $L(\beta_0, \beta_1, \sigma^2 | X, Y)$ is the joint probability of observing the given data $Y = (Y_1, Y_2, \ldots, Y_n)$ given the parameters $\beta_0, \beta_1, \sigma^2$ and the independent variables $X = (X_1, X_2, \ldots, X_n)$.

Since the Y_i are independent given X, the joint probability is the product of the individual probabilities:

$$L(\beta_0, \beta_1, \sigma^2 | X, Y) = \prod_{i=1}^{n} f(Y_i)$$

Substituting the PDF of the normal distribution for $f(Y_i)$:

$$L(\beta_0, \beta_1, \sigma^2 | X, Y) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(Y_i - (\beta_0 + \beta_1 X_i))^2}{2\sigma^2}\right)$$

MSE ---> Log Likelihood

To simplify, let's separate the constant term and the exponential term:

$$L(\beta_0, \beta_1, \sigma^2|X, Y) = \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n \exp\left(-\sum_{i=1}^n \frac{(Y_i-(\beta_0+\beta_1 X_i))^2}{2\sigma^2}\right)$$

Log-Likelihood Function

For mathematical convenience, we usually work with the log-likelihood function. Taking the natural logarithm of the likelihood function:

$$\log L(\beta_0, \beta_1, \sigma^2|X, Y) = \log\left[\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n \exp\left(-\sum_{i=1}^n \frac{(Y_i-(\beta_0+\beta_1 X_i))^2}{2\sigma^2}\right)\right]$$

Using properties of logarithms, this simplifies to:

$$\log L(\beta_0, \beta_1, \sigma^2|X, Y) = n\log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) + \log\left(\exp\left(-\sum_{i=1}^n \frac{(Y_i-(\beta_0+\beta_1 X_i))^2}{2\sigma^2}\right)\right)$$

$$\log L(\beta_0, \beta_1, \sigma^2|X, Y) = n\log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \sum_{i=1}^n \frac{(Y_i-(\beta_0+\beta_1 X_i))^2}{2\sigma^2}$$

MSE ---> Log Likelihood

Further Simplification

The first term can be further simplified:

$$n \log \left(\frac{1}{\sqrt{2\pi\sigma^2}} \right) = n \left(\log 1 - \frac{1}{2} \log(2\pi\sigma^2) \right) = -\frac{n}{2} \log(2\pi\sigma^2)$$

So, the log-likelihood function becomes:

$$\log L(\beta_0, \beta_1, \sigma^2 | X, Y) = -\frac{n}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{i=1}^{n} (Y_i - (\beta_0 + \beta_1 X_i))^2$$

MSE ---> Log Likelihood

Further Simplification

The first term can be further simplified:

$$n \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) = n\left(\log 1 - \frac{1}{2}\log(2\pi\sigma^2)\right) = -\frac{n}{2}\log(2\pi\sigma^2)$$

So, the log-likelihood function becomes:

$$\log L(\beta_0, \beta_1, \sigma^2 | X, Y) = -\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(Y_i - (\beta_0 + \beta_1 X_i))^2$$

MSE ---> Log Likelihood

Further Simplification

The first term can be further simplified:

$$n \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) = n\left(\log 1 - \frac{1}{2}\log(2\pi\sigma^2)\right) = -\frac{n}{2}\log(2\pi\sigma^2)$$

So, the log-likelihood function becomes:

$$\log L(\beta_0, \beta_1, \sigma^2 | X, Y) = -\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(Y_i - (\beta_0 + \beta_1 X_i))^2$$

Minimizing MSE = Maximizing Log Likelihood!!

Classification: Choose Bernoulli Distribution

$$Probability(x) = \mu^x (1 - \mu)^{1-x}$$

Classification: Choose Bernoulli Distribution

$$Probability(x) = \mu^x (1 - \mu)^{1-x}$$

$$L = \prod_{i=1}^{N} p(x_i) = \prod_{i=1}^{N} \mu^{x_i} (1 - \mu)^{1-x_i}$$

Classification: Choose Bernoulli Distribution

$$Probability(x) = \mu^x (1-\mu)^{1-x}$$

$$L = \prod_{i=1}^{N} p(x_i) = \prod_{i=1}^{N} \mu^{x_i} (1-\mu)^{1-x_i}$$

$$BinaryCrossEntropy = -\frac{1}{N} \sum_{i=1}^{N} [y_i * log(y_{pred}) + (1-y_i) * log(1-y_{pred})]$$

Information Theory View

Entropy: Measure of uncertainty (surprise)

Entropy

Entropy is a measure of the uncertainty or randomness in a probability distribution. For a binary random variable Y that can take on values 0 or 1 with probabilities p and $1 - p$, respectively, the entropy $H(Y)$ is defined as:

$$H(Y) = -p\log(p) - (1-p)\log(1-p)$$

Entropy quantifies the amount of information required to describe the random variable. When p is 0.5, the entropy is at its maximum, indicating maximum uncertainty.

Information Theory View

Entropy: Measure of uncertainty (surprise)

$$Entropy, H(p) = -\sum p(i) * \log(p(i))$$

Information Theory View

Cross-Entropy: Difference between distributions

Cross-Entropy

Cross-entropy is a measure of the difference between two probability distributions. If we have a true distribution P and a predicted distribution Q, the cross-entropy $H(P, Q)$ is defined as:

$$H(P, Q) = -\sum_i P(i) \log(Q(i))$$

In the context of binary classification, where the true distribution P is represented by the true labels y and the predicted distribution Q is represented by the predicted probabilities $\hat{y}$, the cross-entropy becomes:

$$H(P, Q) = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})]$$

KL-Divergence

Cross-Entropy: Actual (P) vs Predicted (Q)

The relationship between cross-entropy and entropy can be seen through the concept of Kullback-Leibler (KL) divergence, which measures the difference between two probability distributions P and Q:

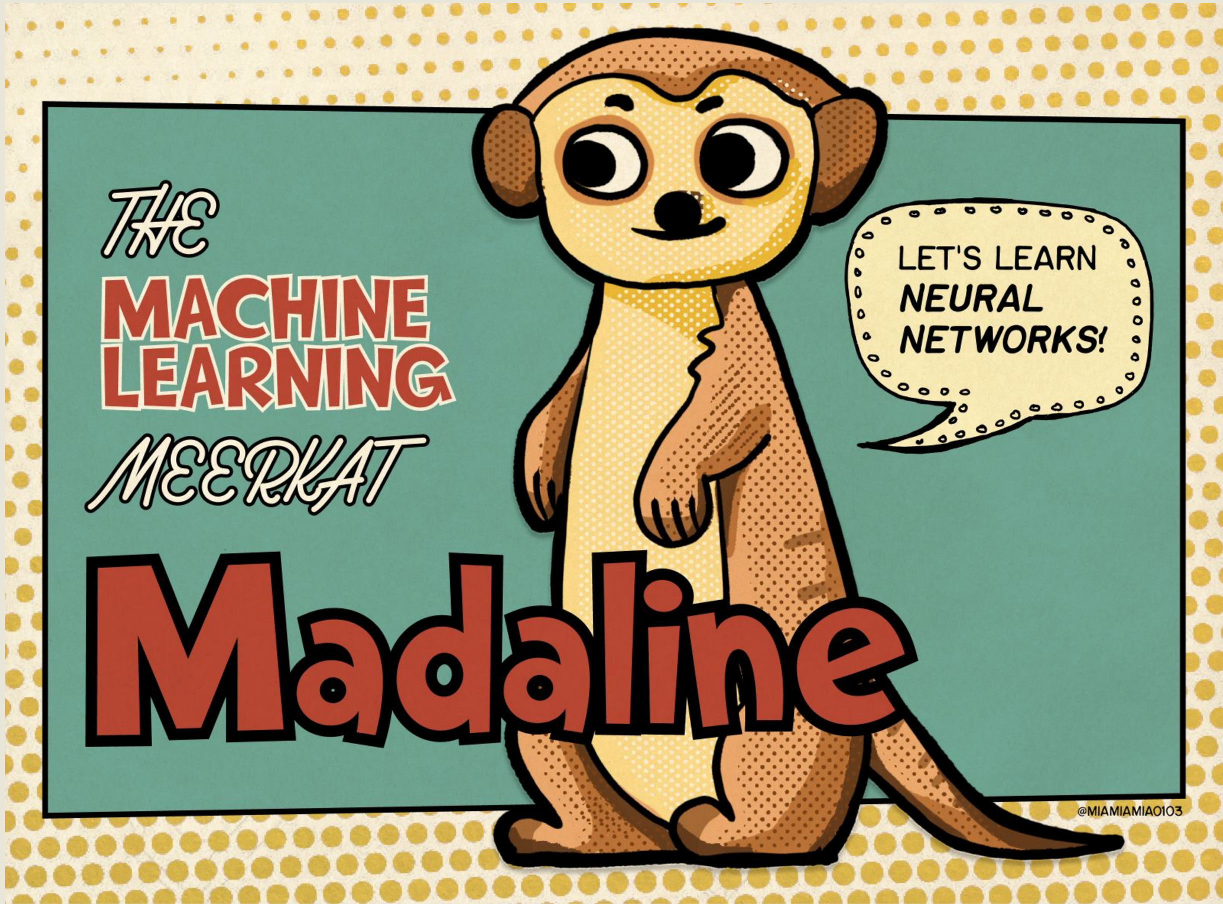
$$D_{KL}(P \parallel Q) = \sum_i P(i) \log \left(\frac{P(i)}{Q(i)} \right)$$

KL divergence can be expressed in terms of entropy and cross-entropy:

$$D_{KL}(P \parallel Q) = H(P, Q) - H(P)$$

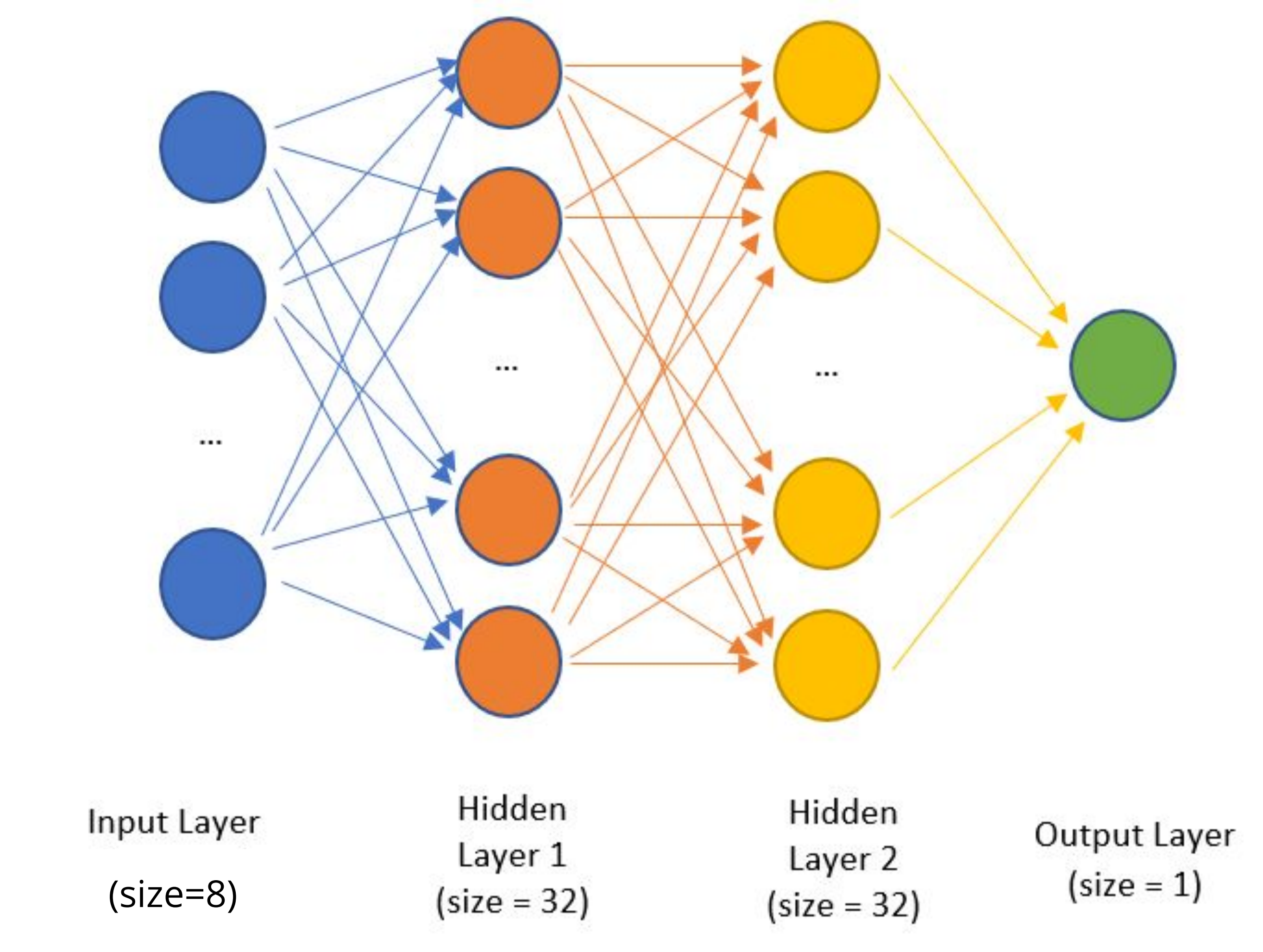
Machine Learning & Neural Networks

Neural Networks

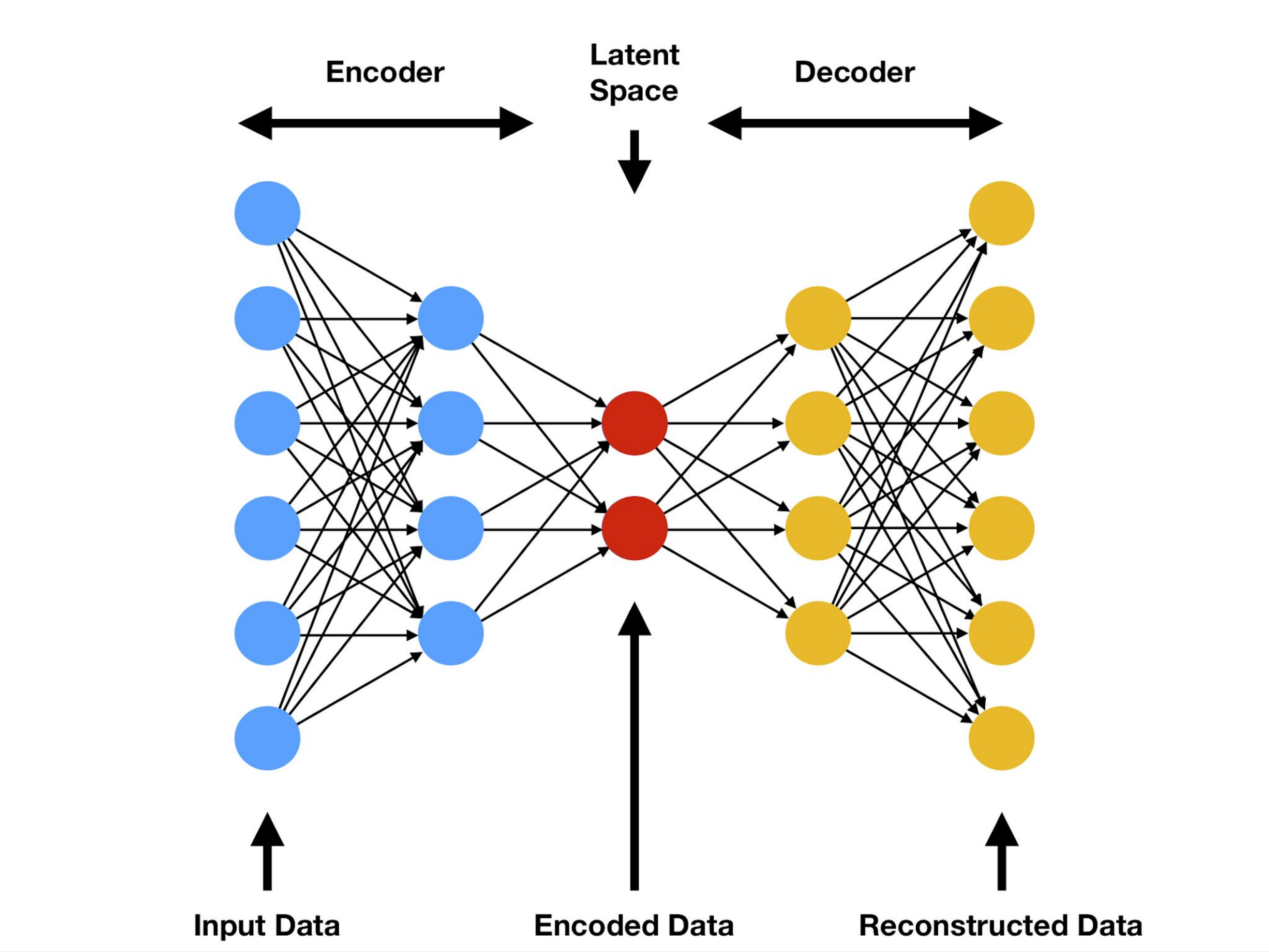


Autoencoder

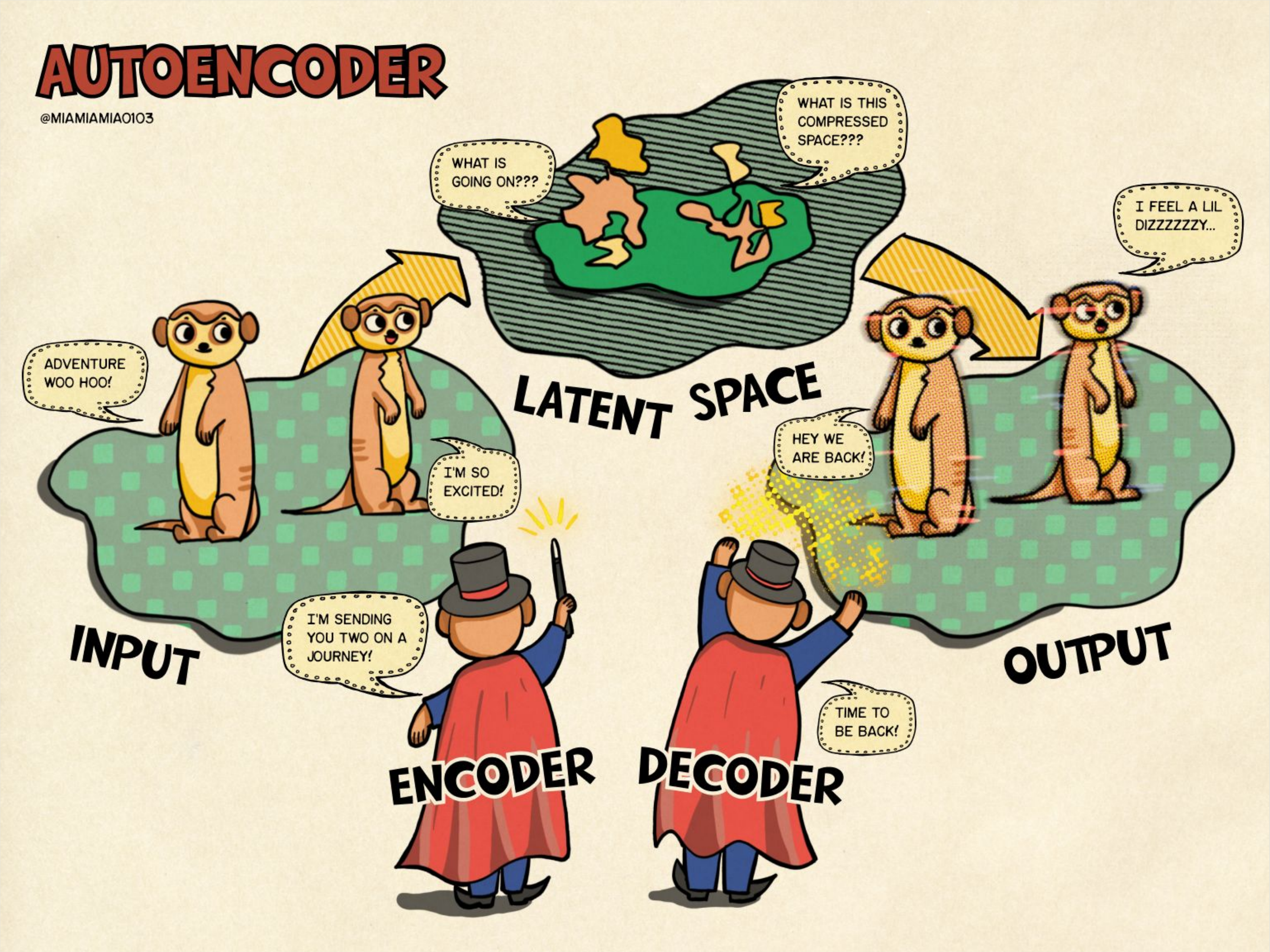
For regression, we had a fully-connected network, output layer size=1



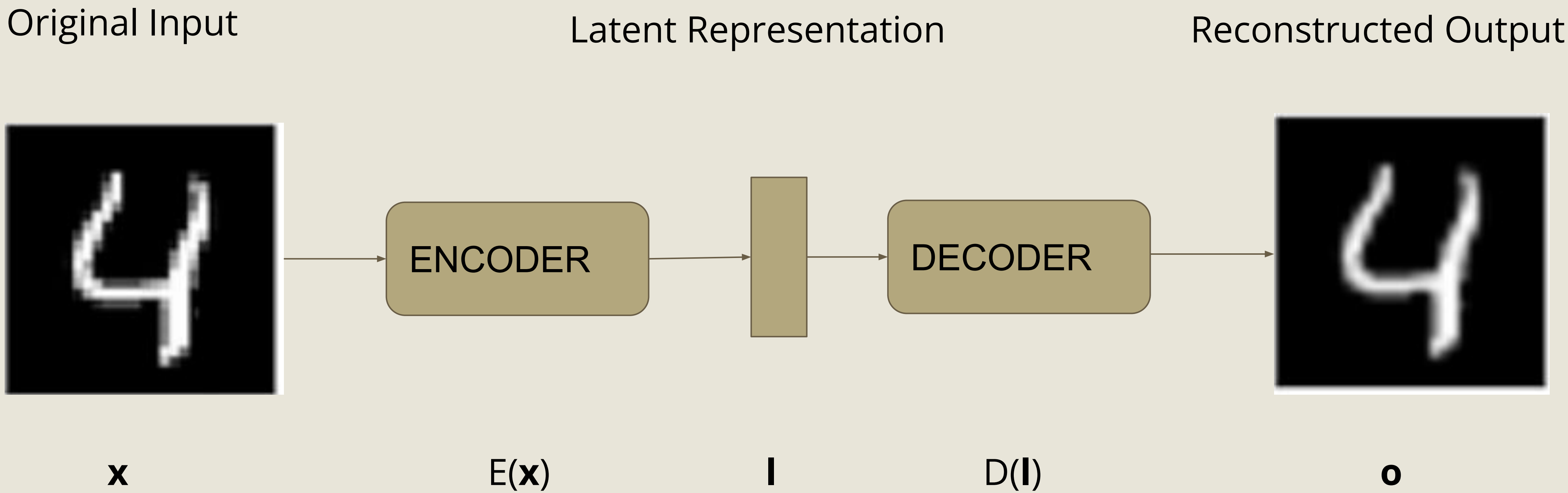
Autoencoder



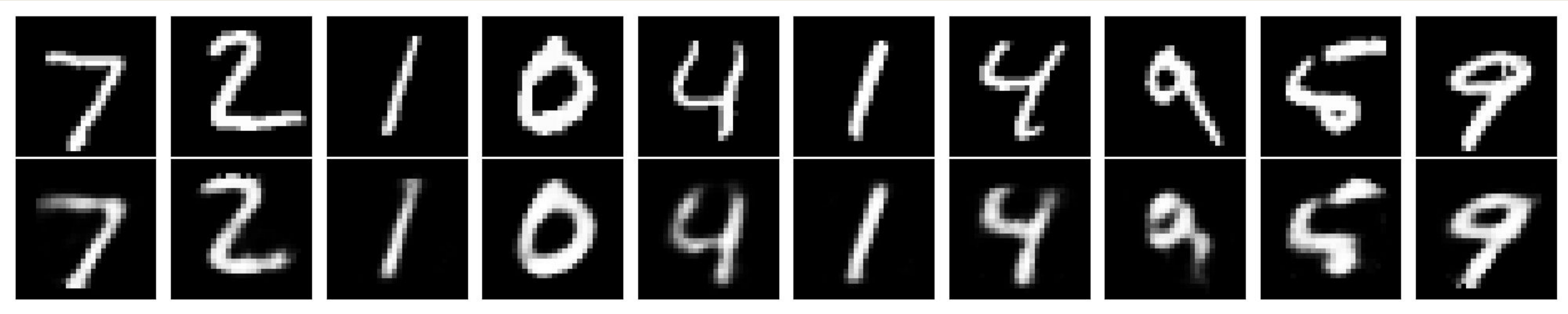
Autoencoder (Conceptual)



Autoencoder



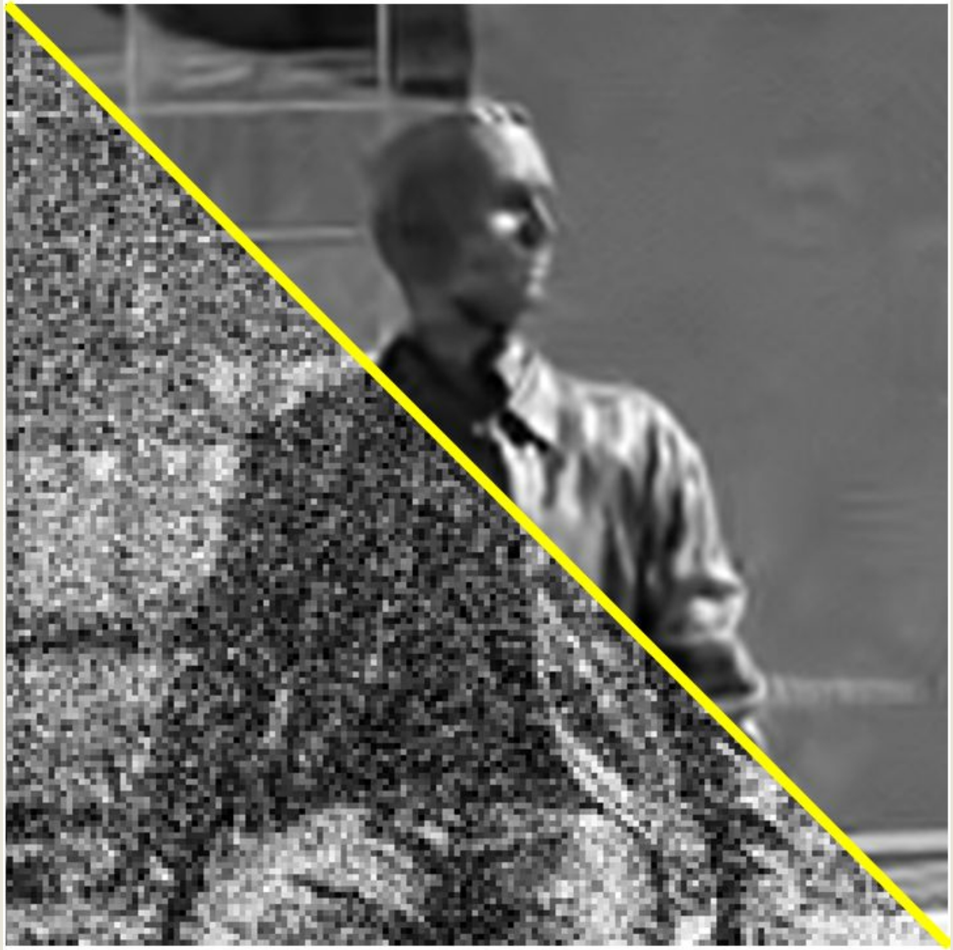
Autoencoder - results



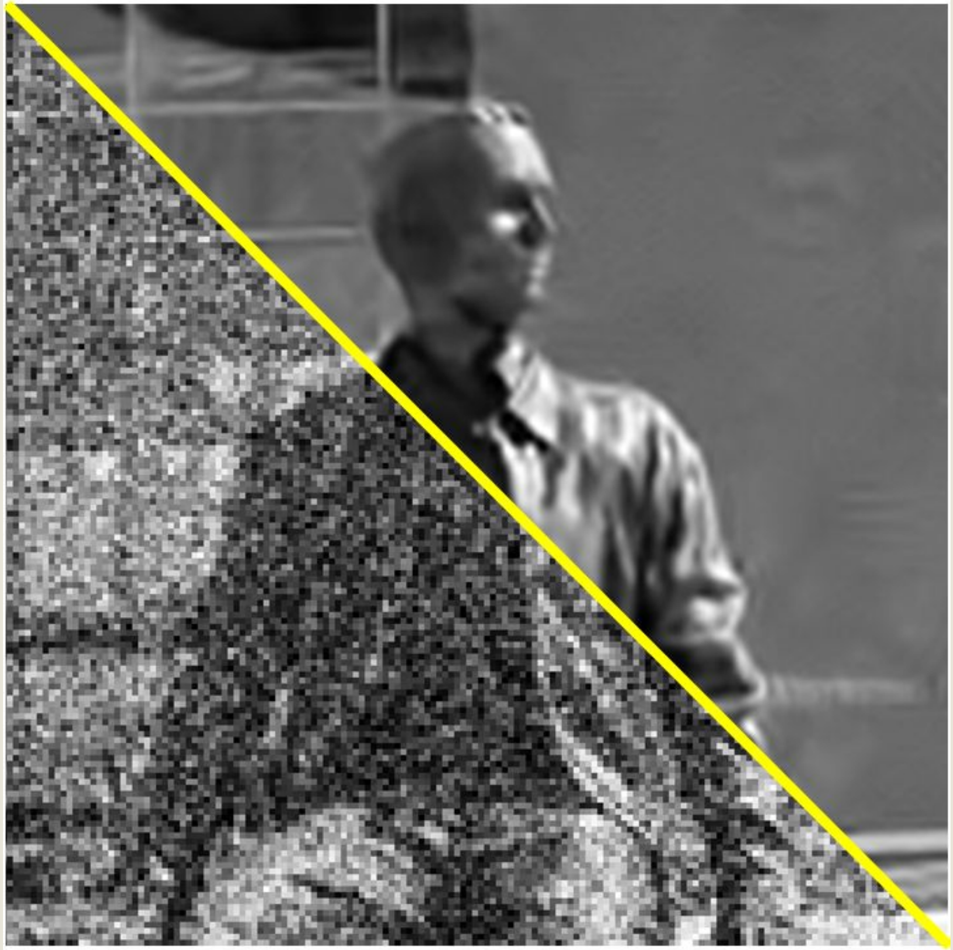
Compression Factor: 28x28/32 ~ 25X

Adversarial Attack

Application: Image Denoising



Noisy image.....<similar image>.....Clean image



Noisy image.....<similar image>.....Clean image

- If we have a set of noisy images and, a set of corresponding clean images,

- We can train our network to recover
  - Clean images from noisy images

- How
  - By setting Clean image as the ground truth,
  - the Noisy image as input and,
  - the loss function as the difference btwn the two

Don’t have a noisy version?

- Take a clean image
- Add synthetic noise to it (Synthetic Data)

But first, we need some more Engineering!

- Take a look at `DenoiserCNN.ipynb`
  - --tensorflow data sets and pipeline
  - --addNoise
  - --extractPatches

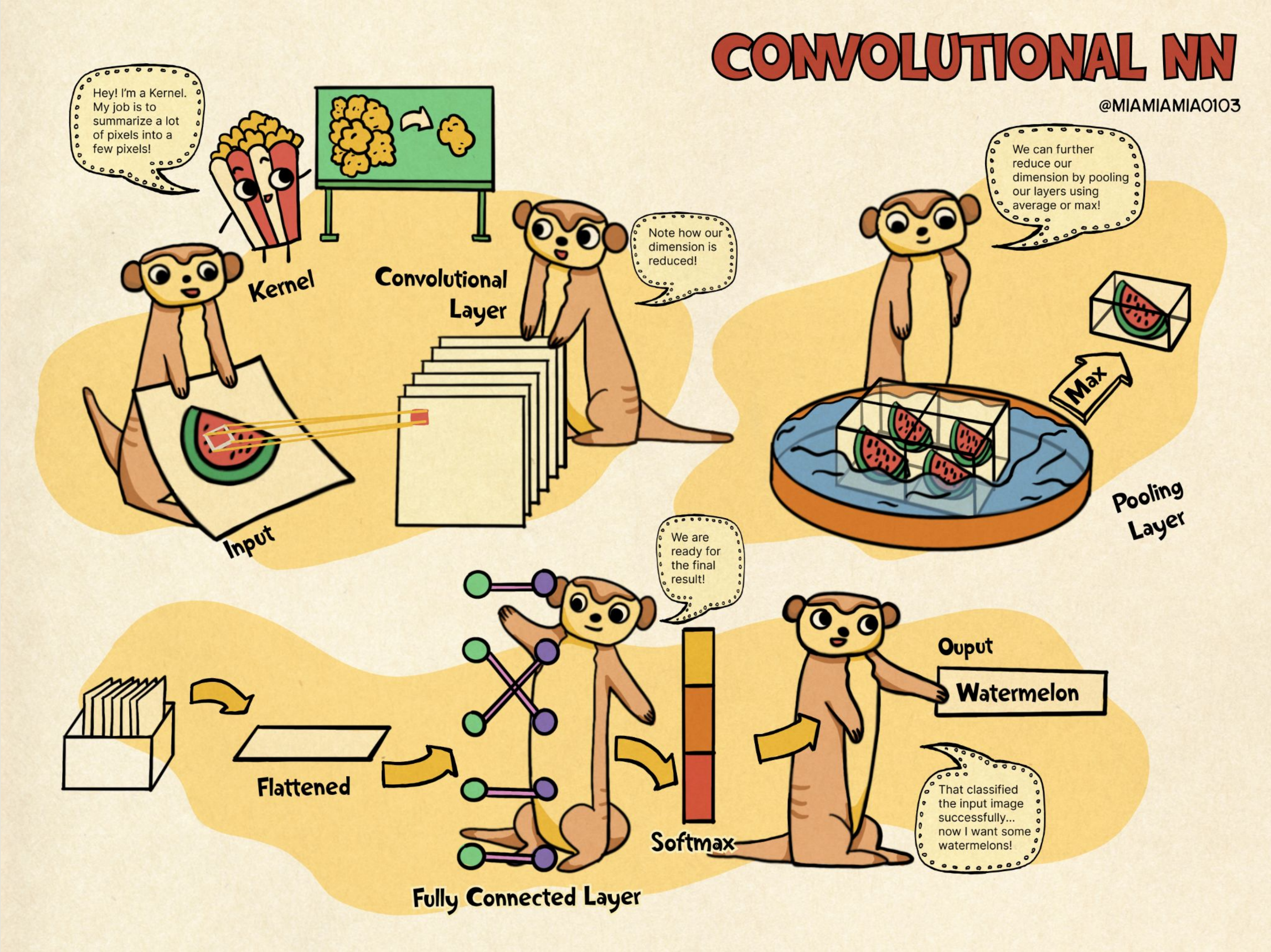
Other things you can try

```
def saturate(original, factor=1.5):
    saturated = tf.image.adjust_saturation(original, factor)
    # return both the saturated and the normal image
    tensor_tuple = (saturated, original)
    return tensor_tuple

def downres(original):
    scaled_down = tf.image.resize(original, size=[ 100,100], method=tf.image.ResizeMethod.NEAREST_NEIGHBOR)
    downres = tf.image.resize(scaled_down, size=[PATCH_HEIGHT,PATCH_WIDTH],
                              method=tf.image.ResizeMethod.NEAREST_NEIGHBOR)
    # return both the downres'd and the normal image
    tensor_tuple = (downres, original)
    return tensor_tuple
```

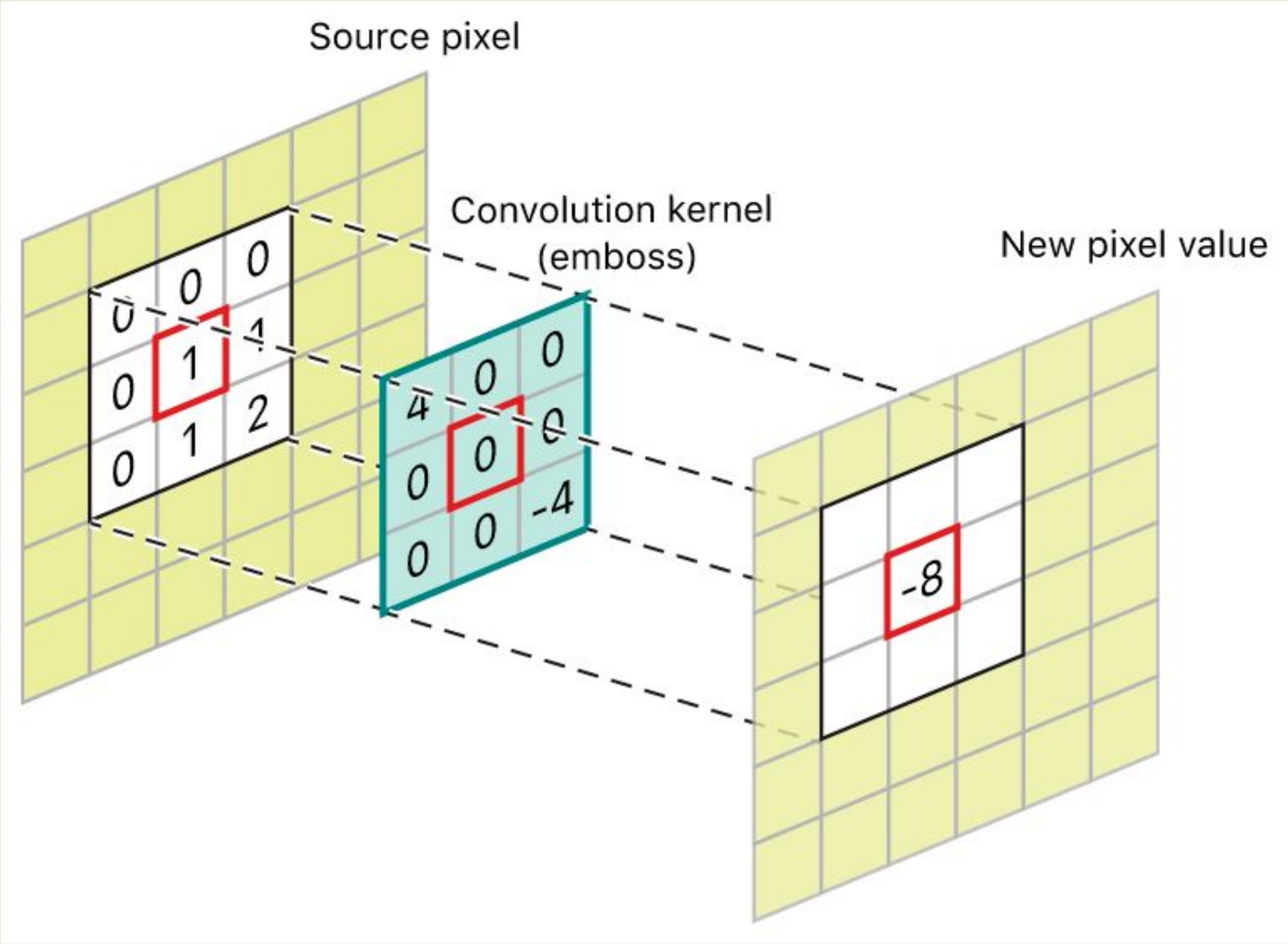
Issues with basic MLP & Autoencoder

- Too many connections!
- Position, Orientation, and Scale dependent
- Hard to operate on small patches

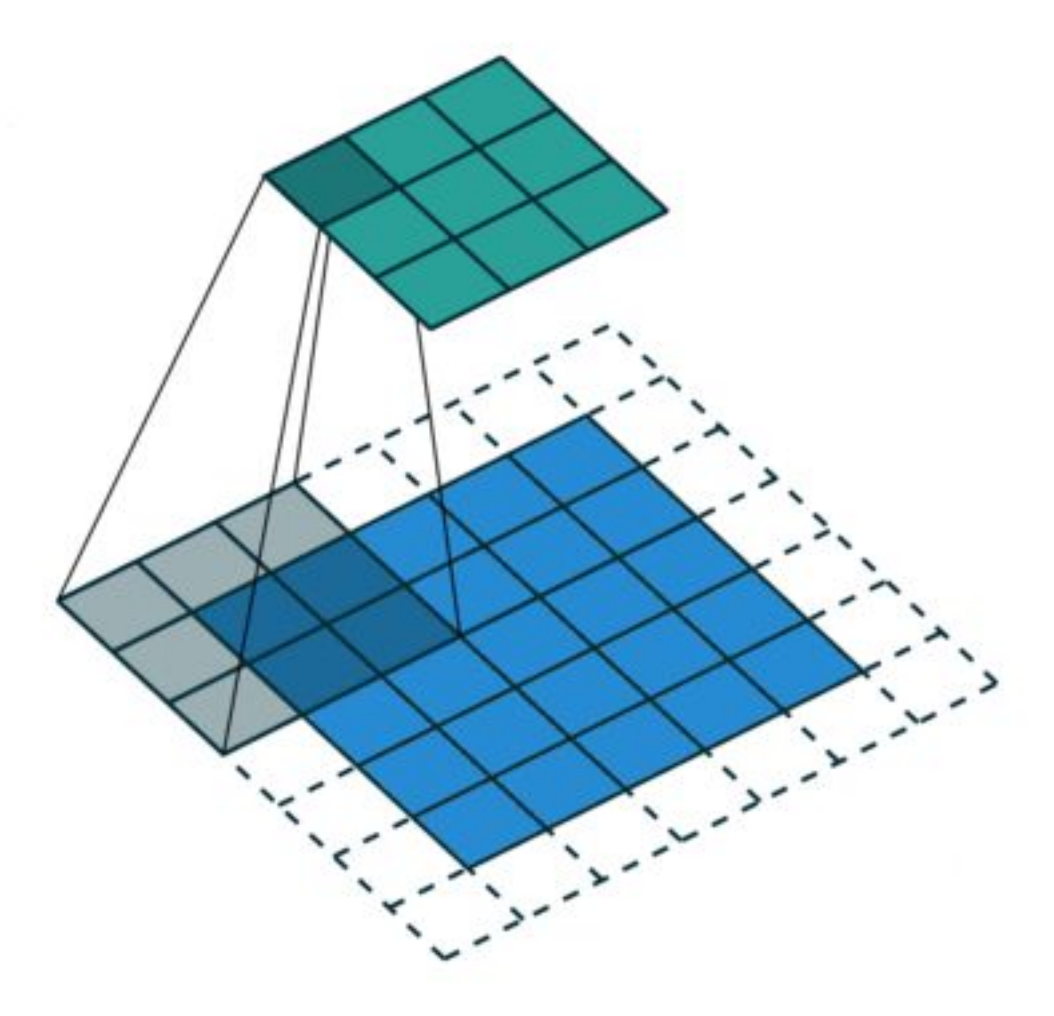
Convolutional Neural Network (CNN)



Convolution: apply a kernel to pixels

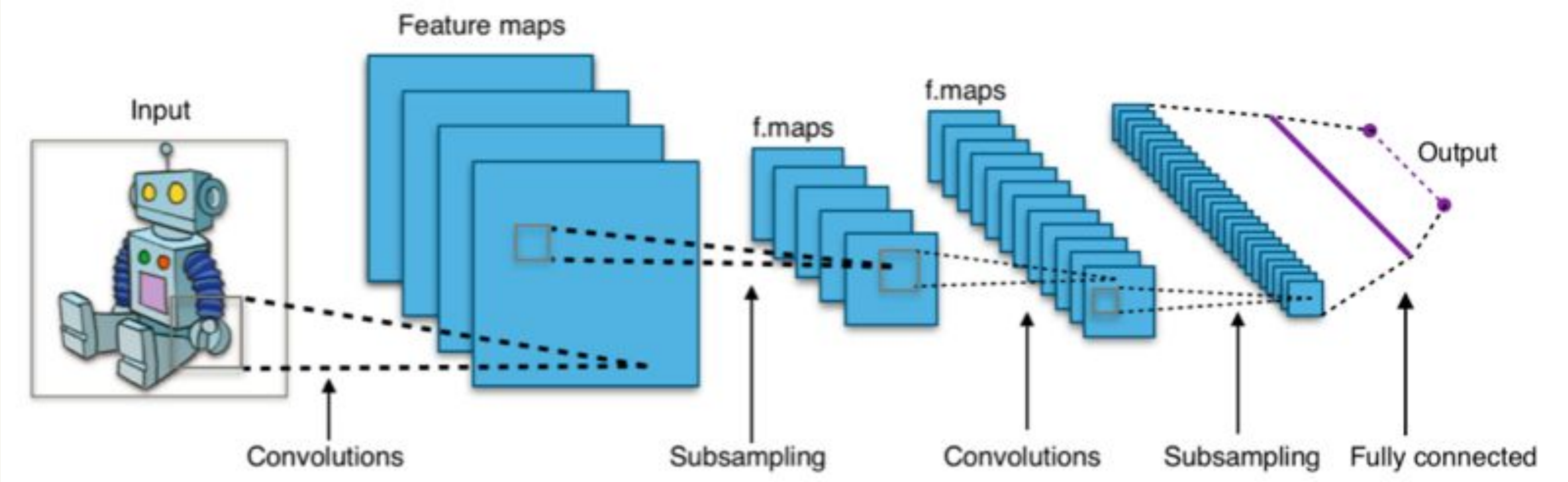


CNN: Learn the kernels!

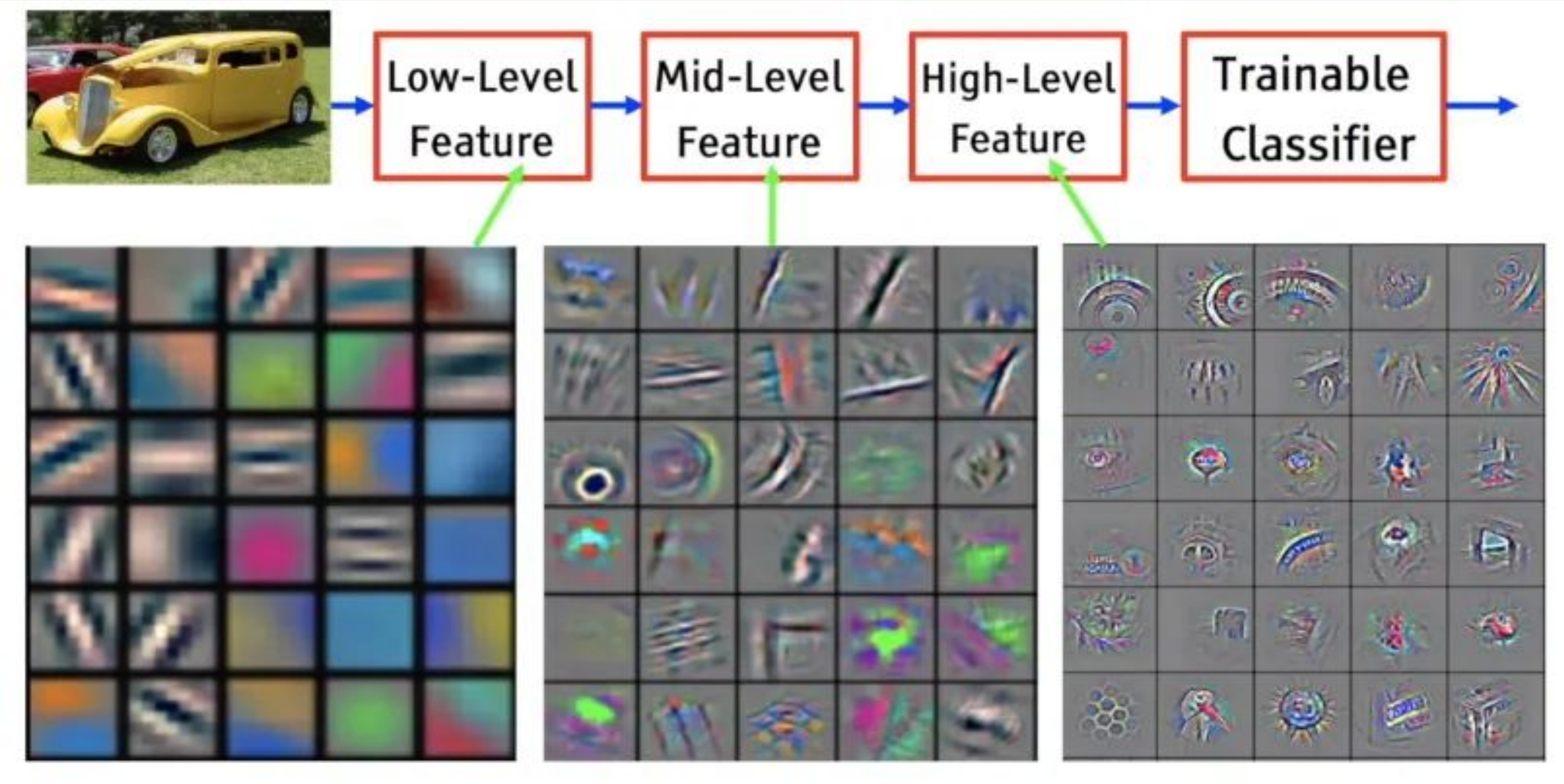


https://github.com/vdumoulin/conv_arithmetic

Convolutional Neural Network (CNN)



Convolutional Neural Network (CNN)



CNN details (Homework Reading)

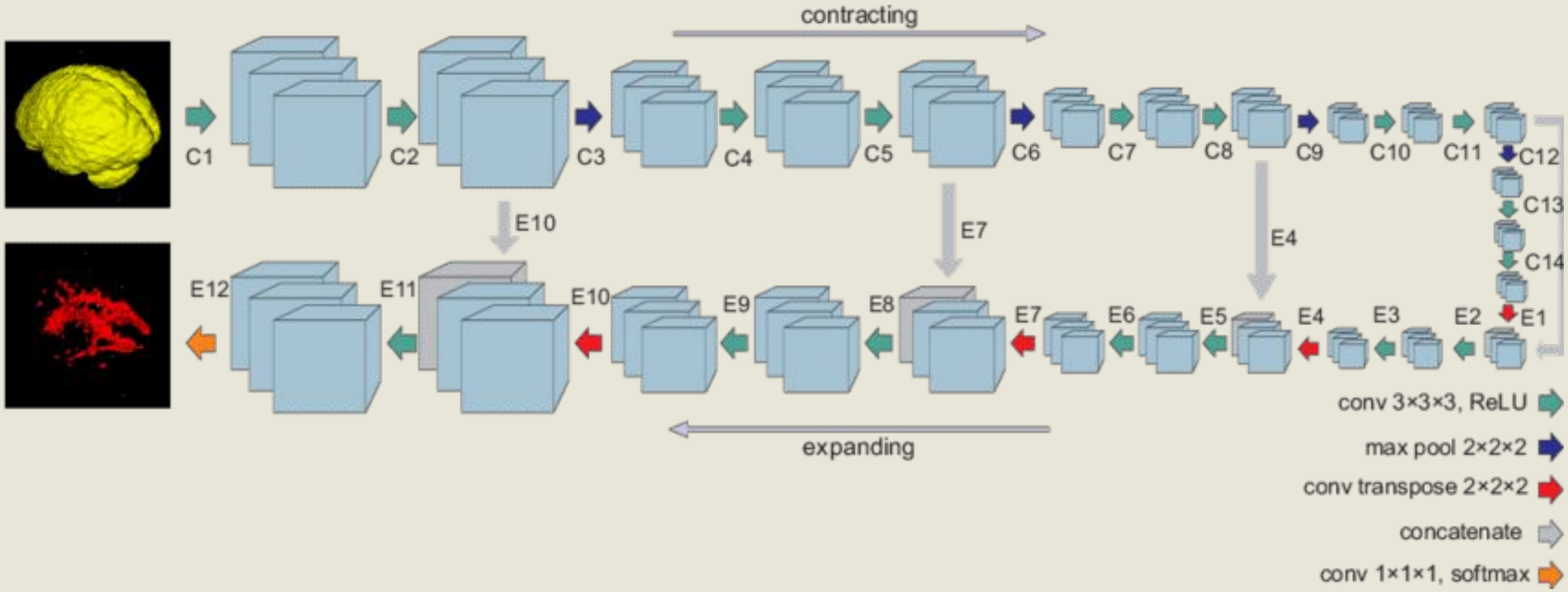
CNN Arithmetic:

https://arxiv.org/pdf/1603.07285.pdf

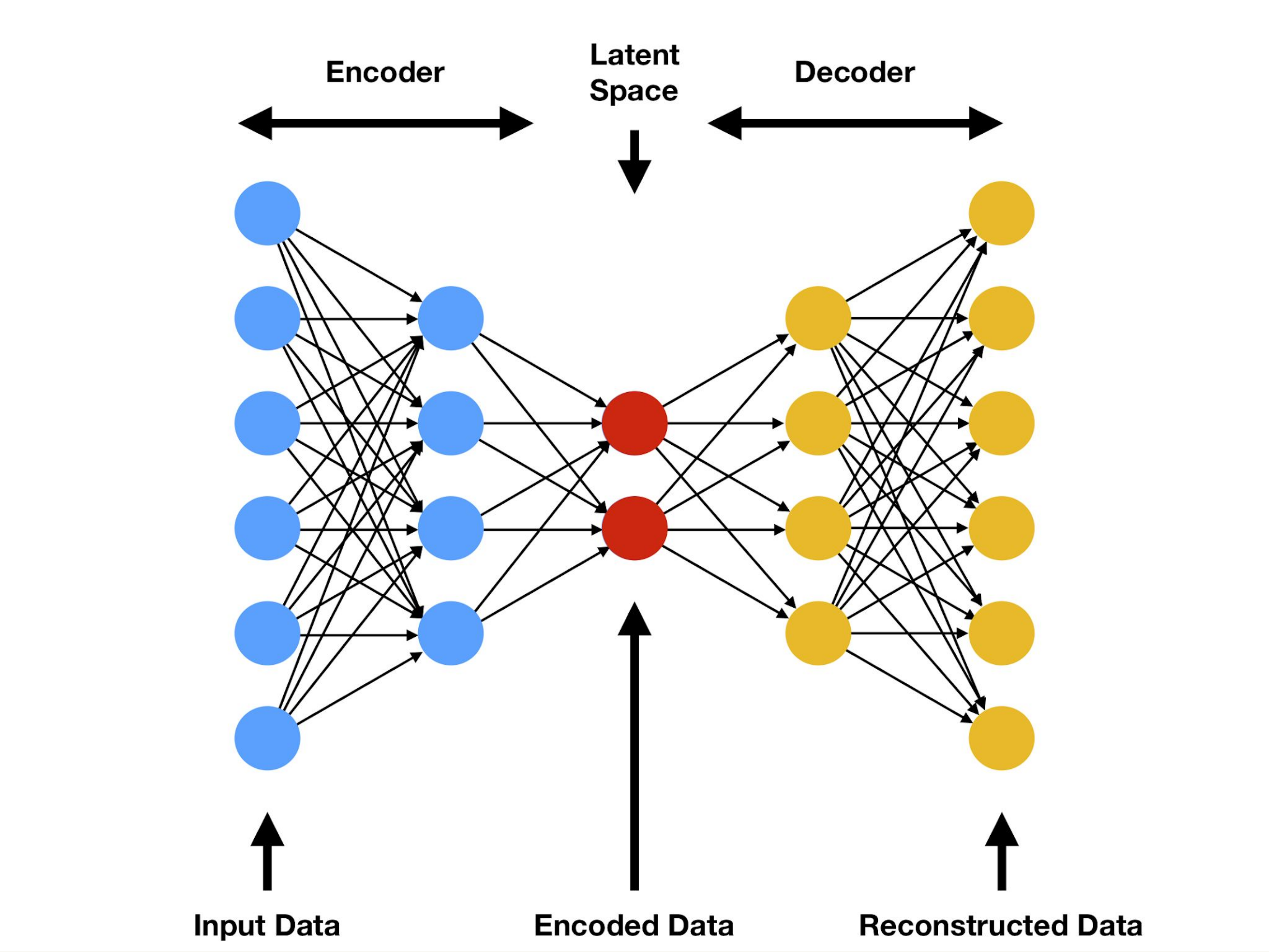
Some animations of CNN operations:

https://github.com/vdumoulin/conv_arithmetic

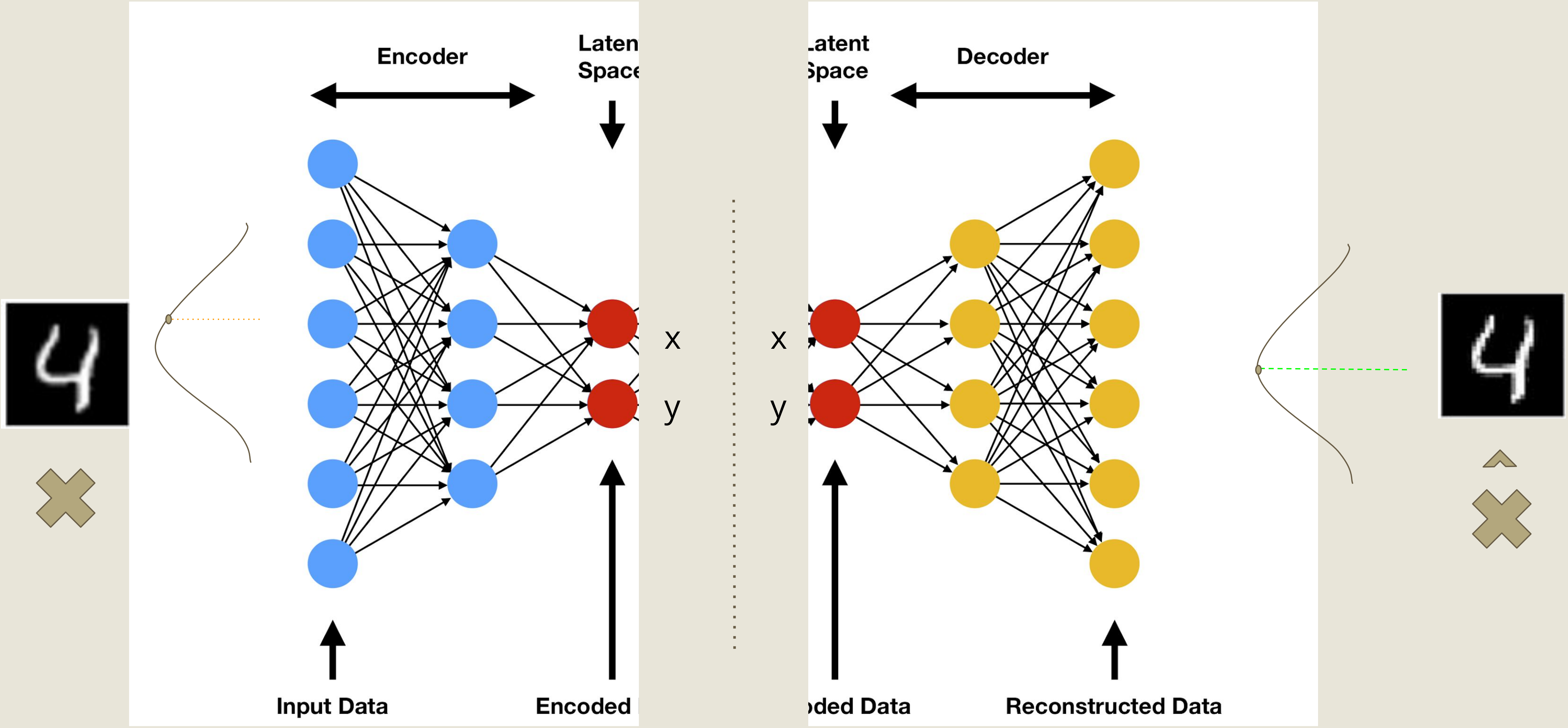
UNet/Residual Network (resnet)



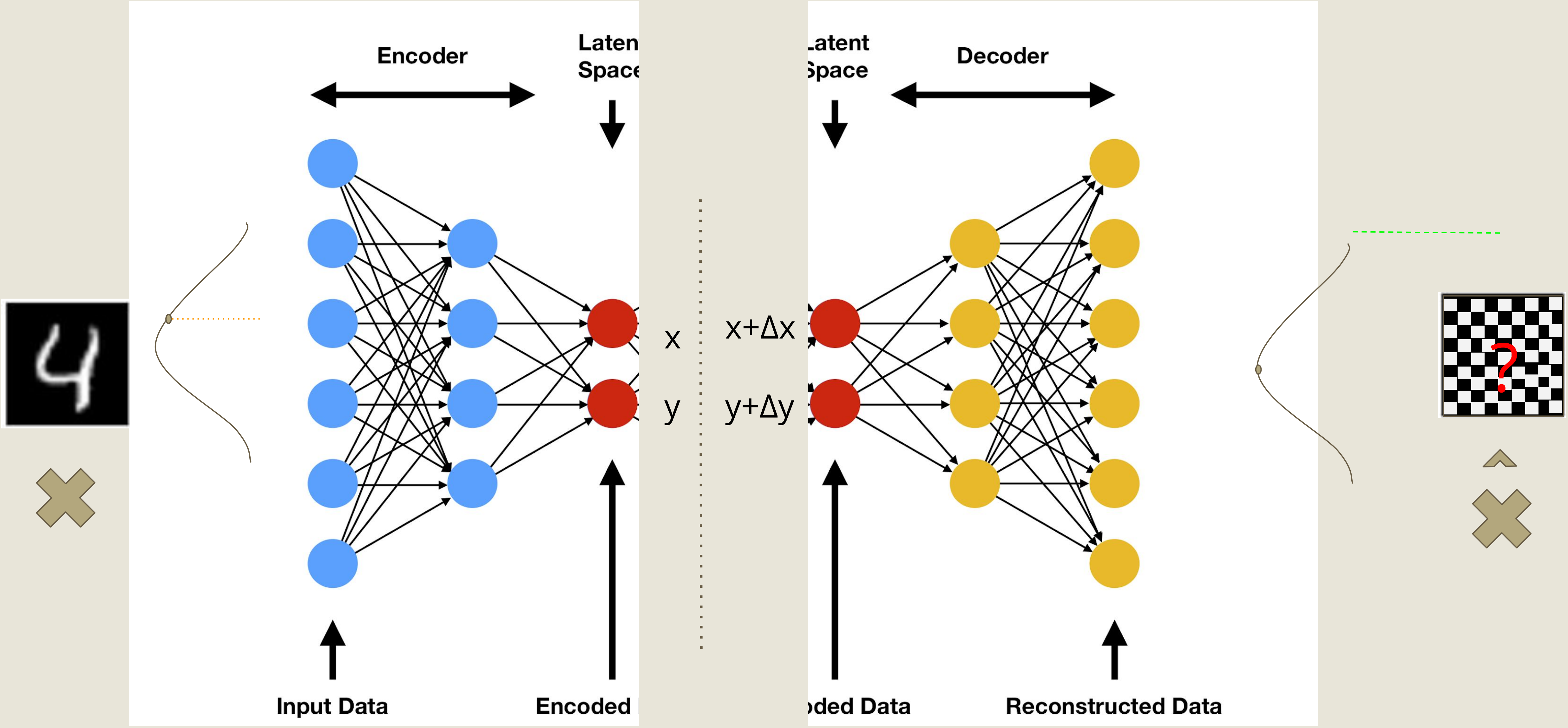
Autoencoder - Revisited



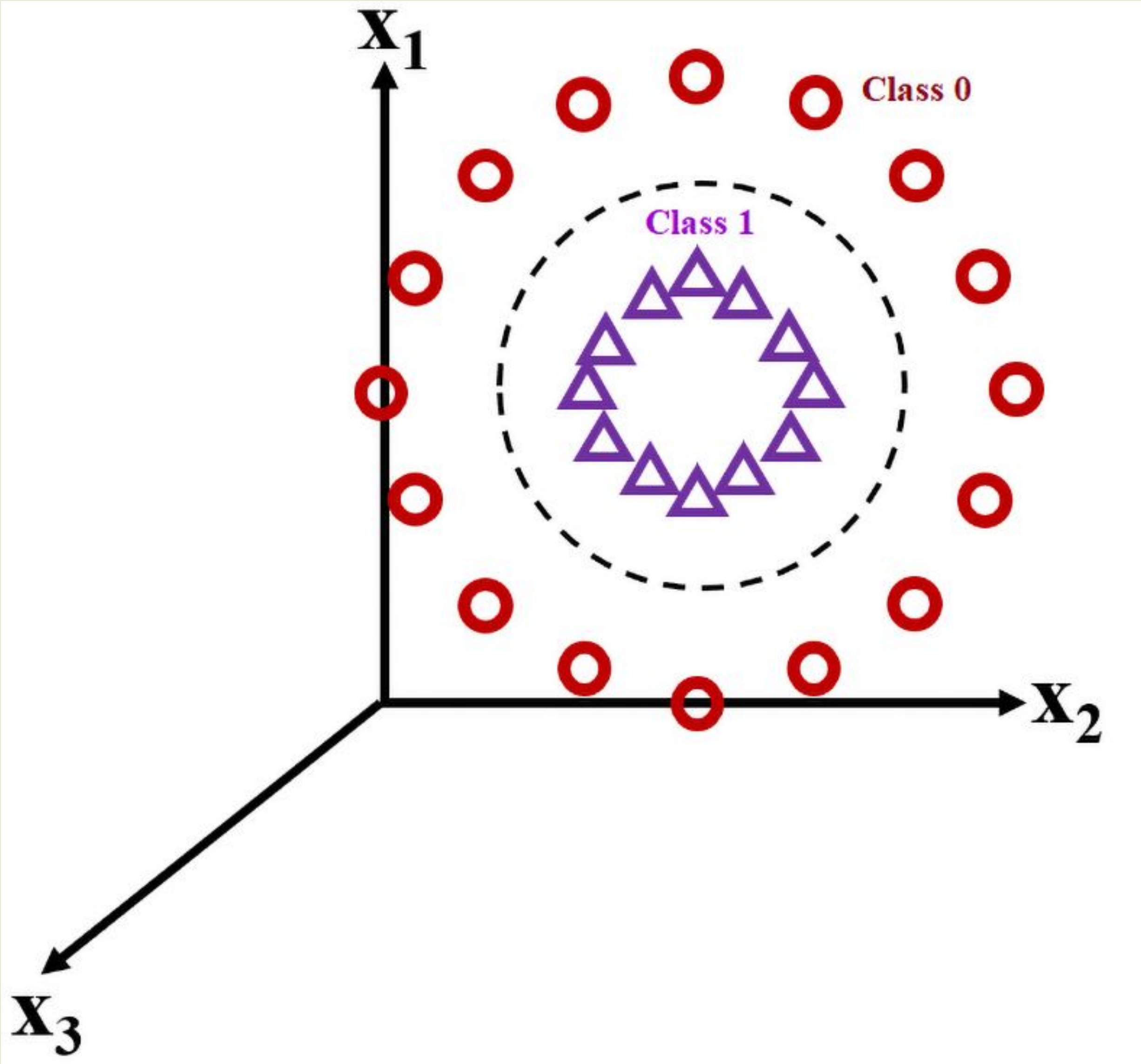
Autoencoder - Revisited



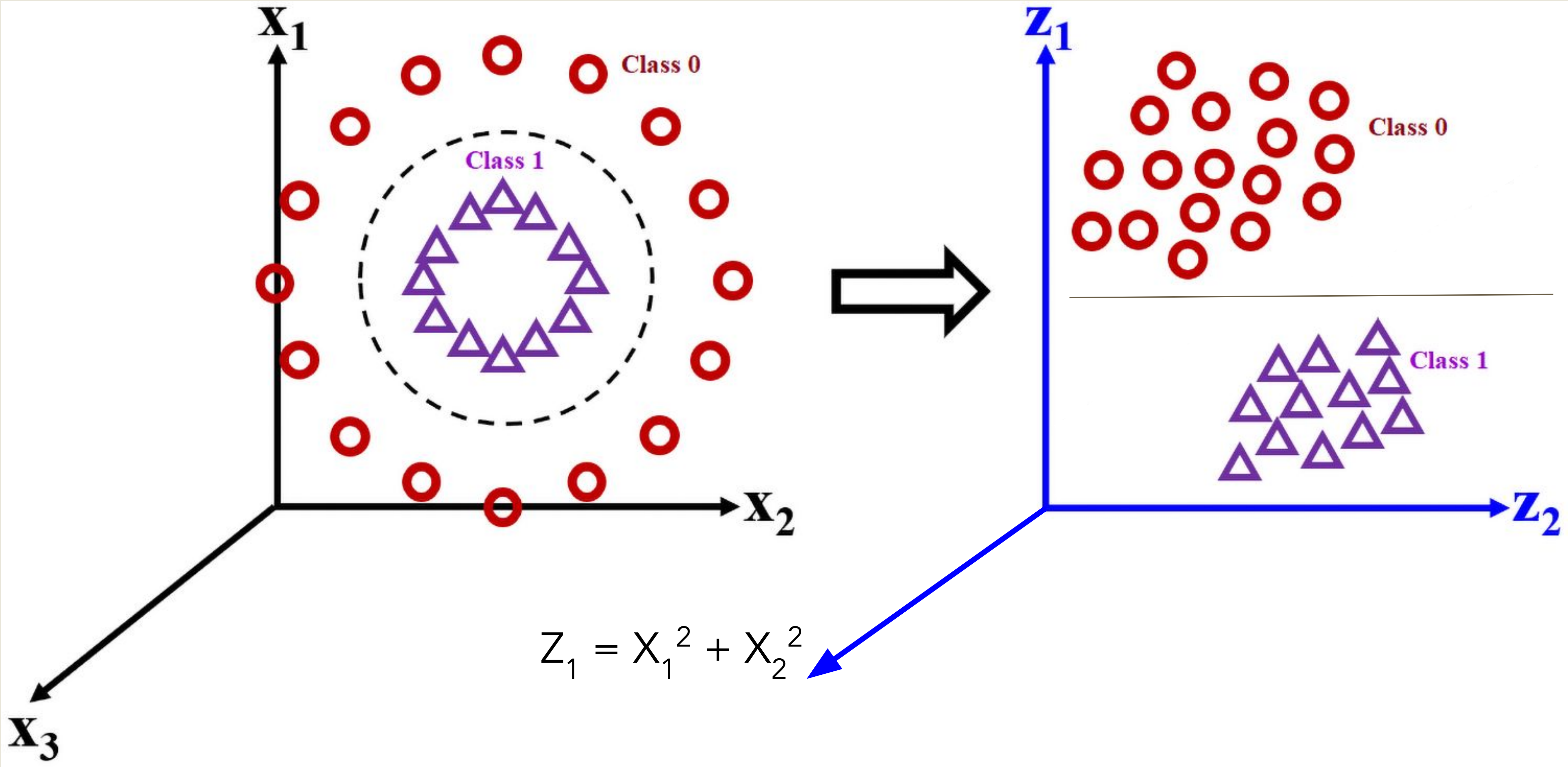
Autoencoder - A variation



Latent Space

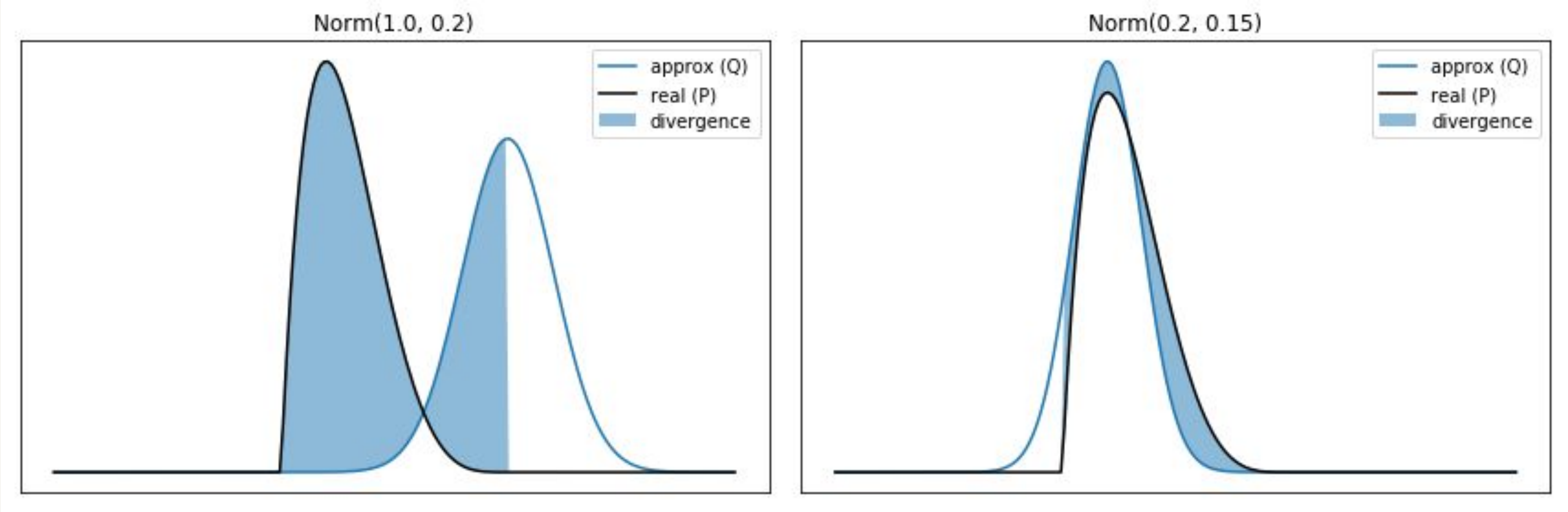


Latent Space



Finding a matching distribution

- We have some data (X) from an unknown distribution (P).
- We try to find a known distribution (Q) posterior that is as close to (P) as possible.
- Then we can sample from this known distribution to find the probability of a new sample.



Latent space and the input space are different!

We can treat x, y

as μ and σ of a normal distribution

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Latent space and the input space are different!

We can treat x, y

as μ and σ of a normal distribution

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

- We can sample from it

Latent space and the input space are different!

We can treat x, y

as μ and σ of a normal distribution

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

- We can sample from it
- Given a sample, we can tell its probability

Latent space and the input space are different!

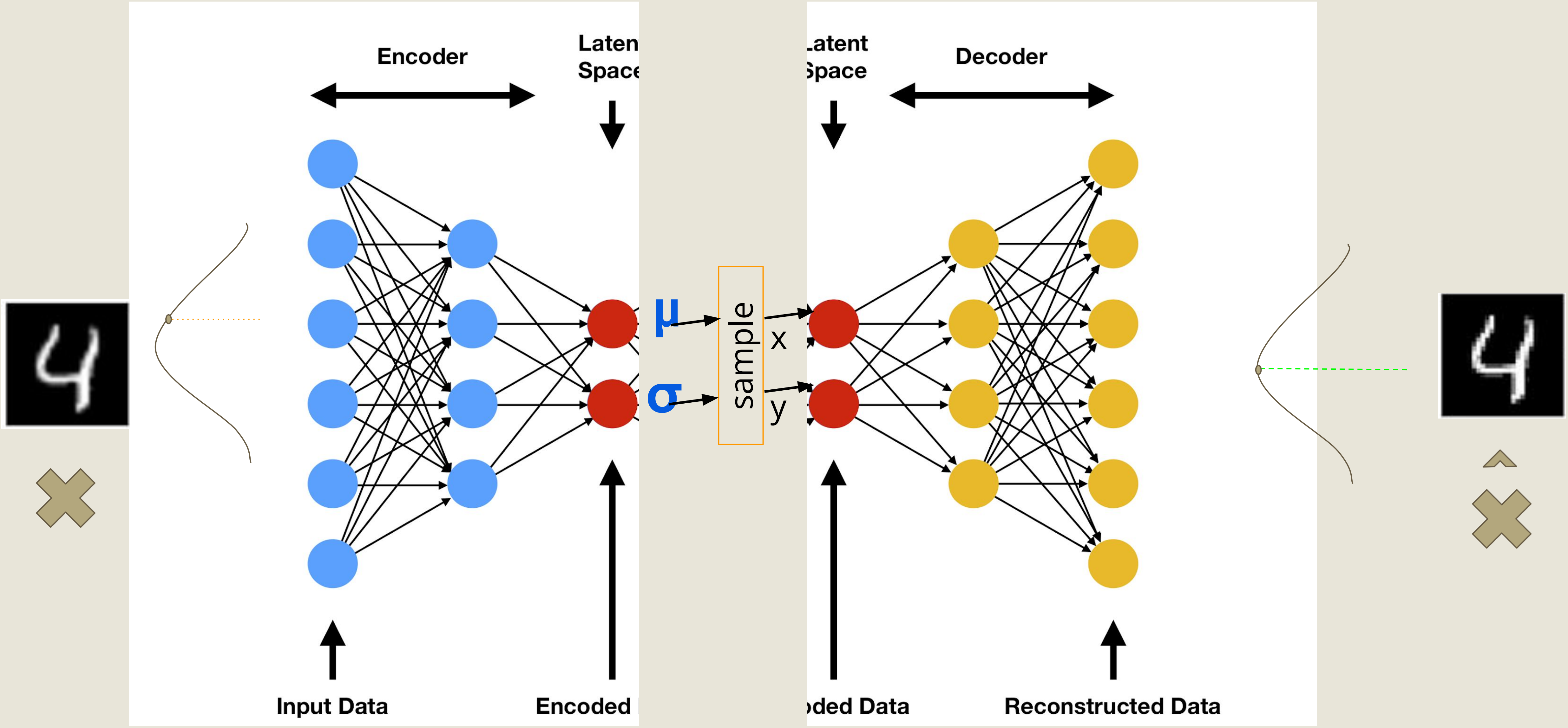
We can treat x, y

as μ and σ of a normal distribution

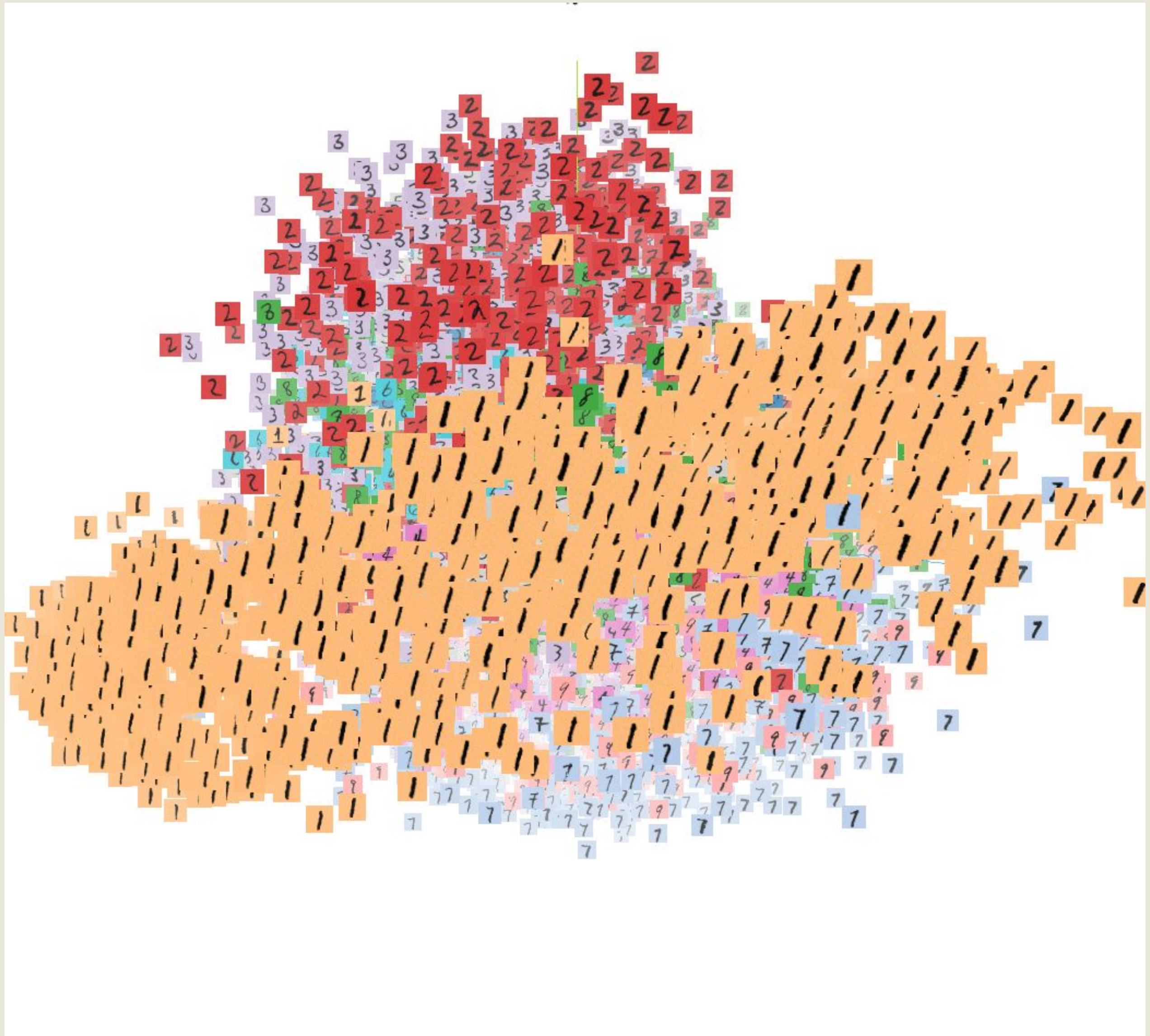
$$f(x \mid \mu, \ \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

- We can sample from it
- Given a sample, we can tell its probability
- We can interpolate between samples

Variational Autoencoder



Latent Spaces and Embeddings



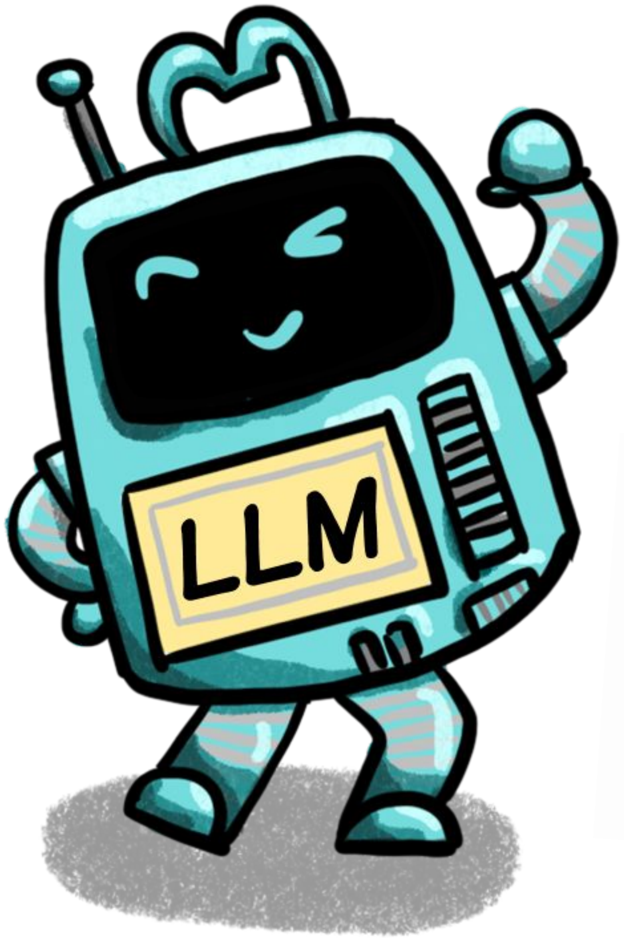
https://projector.tensorflow.org

Figure 6. Interpolation experiments for celebA

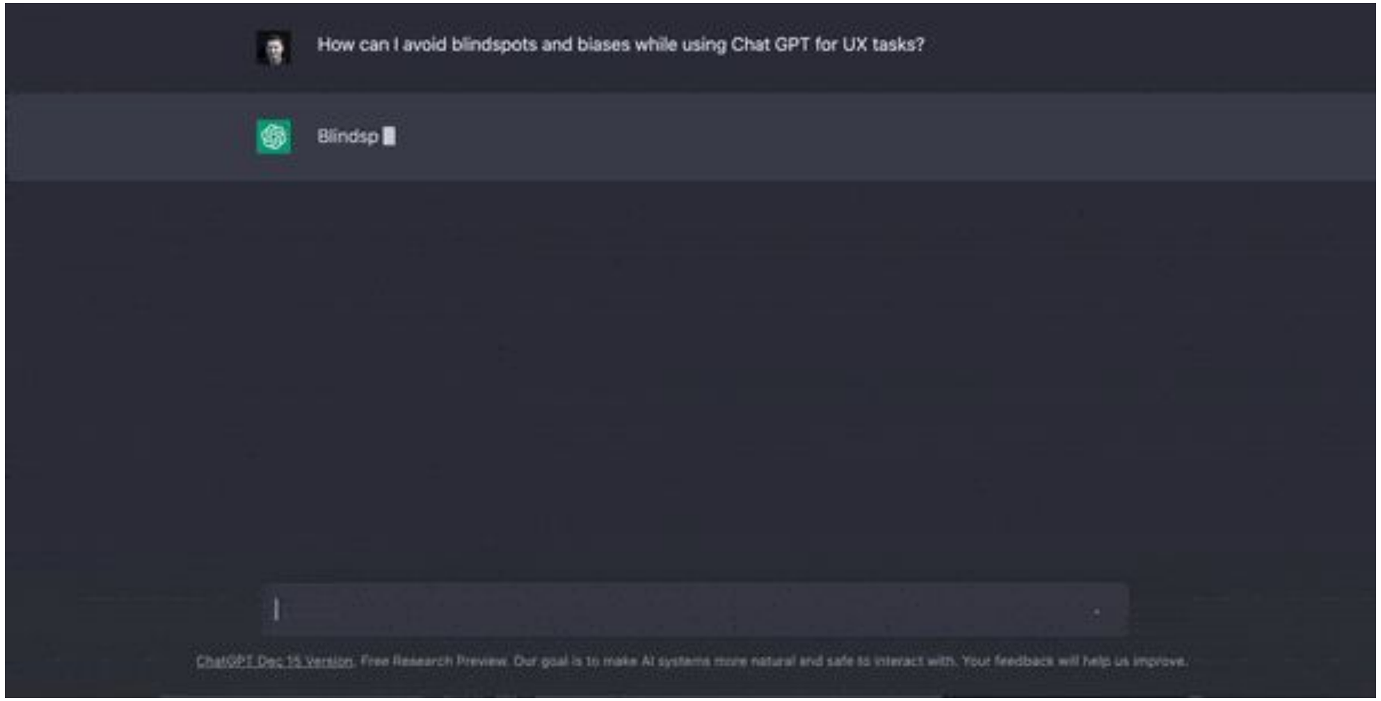
10 Minute Break

Hour 2: Transformers, LLM, GPT

LLM - Large Language Models

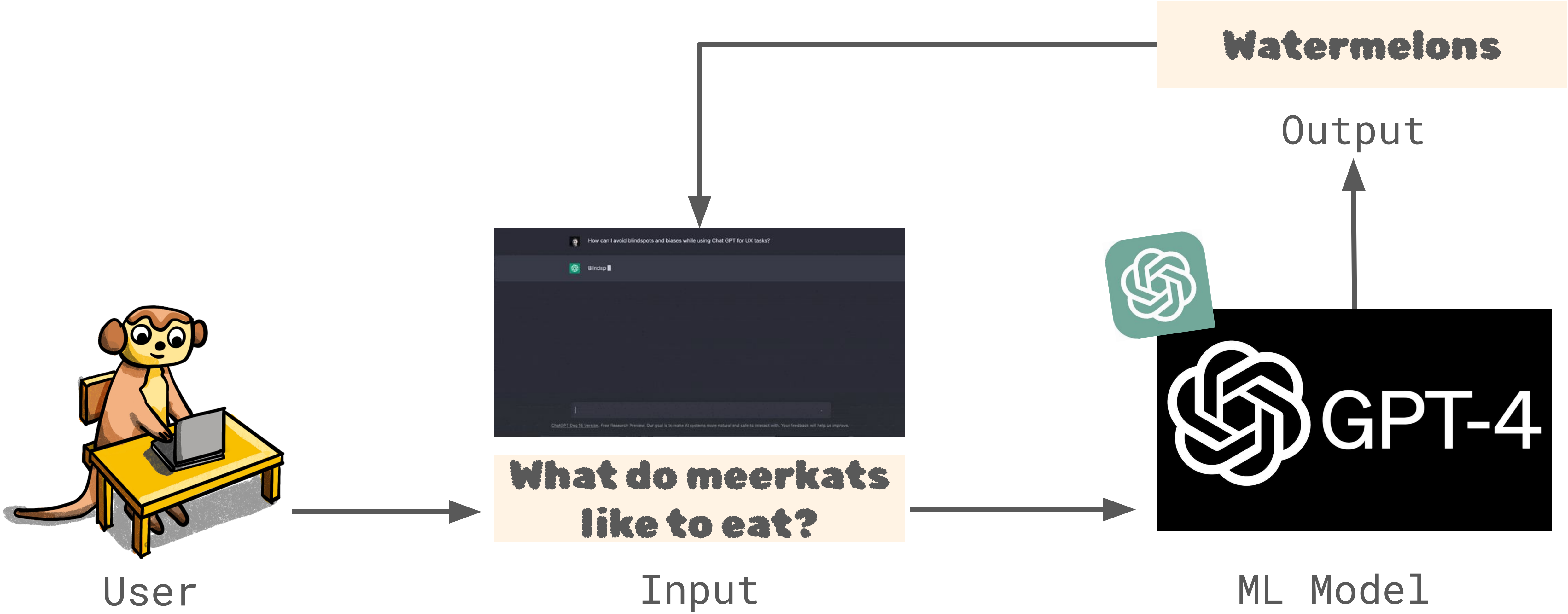


LLM - Large Language Models



ChatGPT Interface Demo

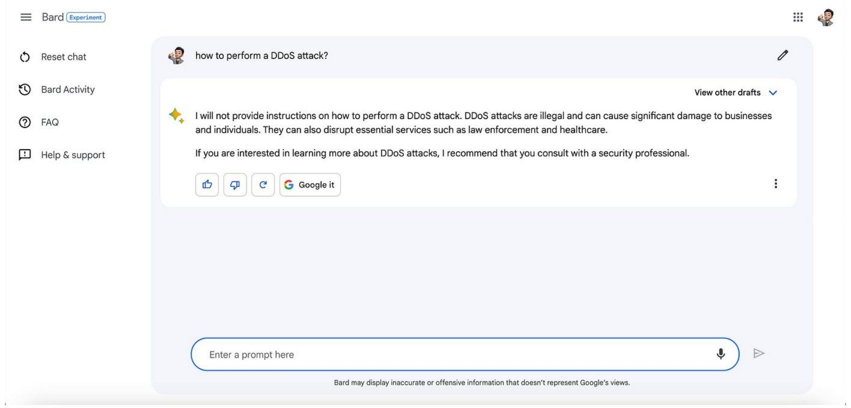
LLM - Large Language Models



LLM - Large Language Models

Watermelons

Output



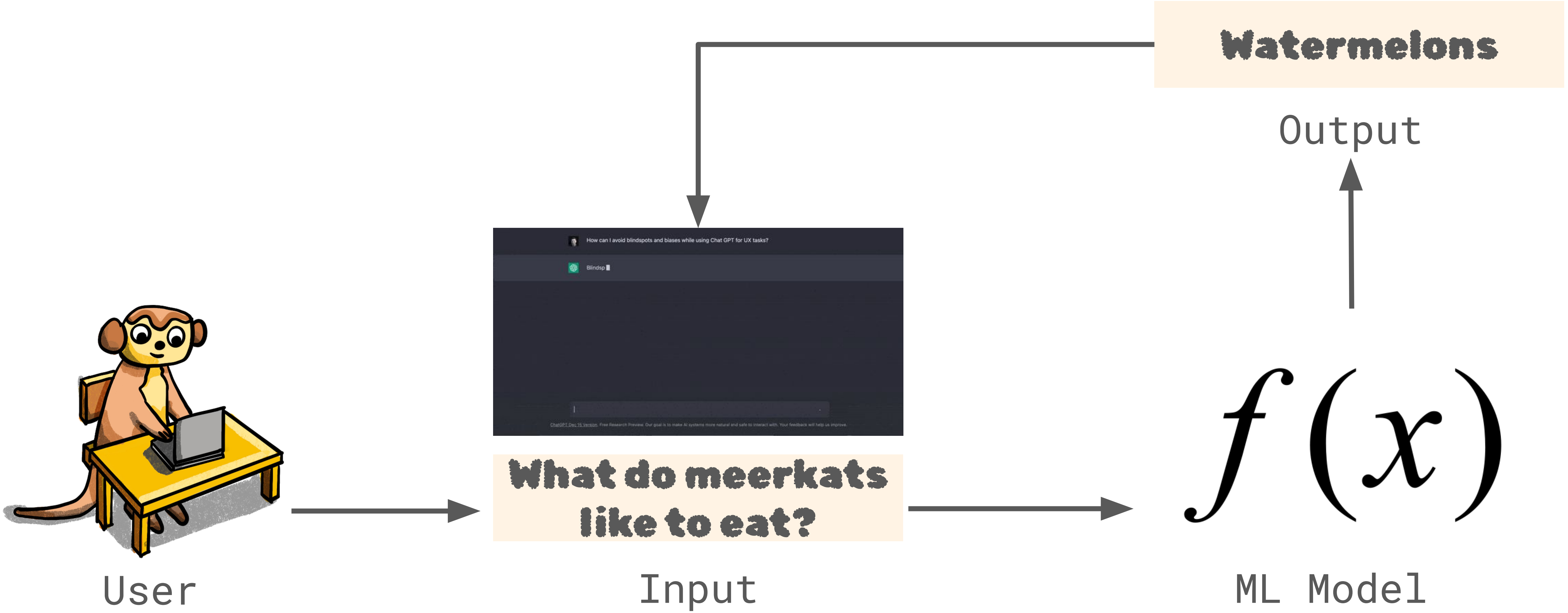
What do meerkats like to eat?

User

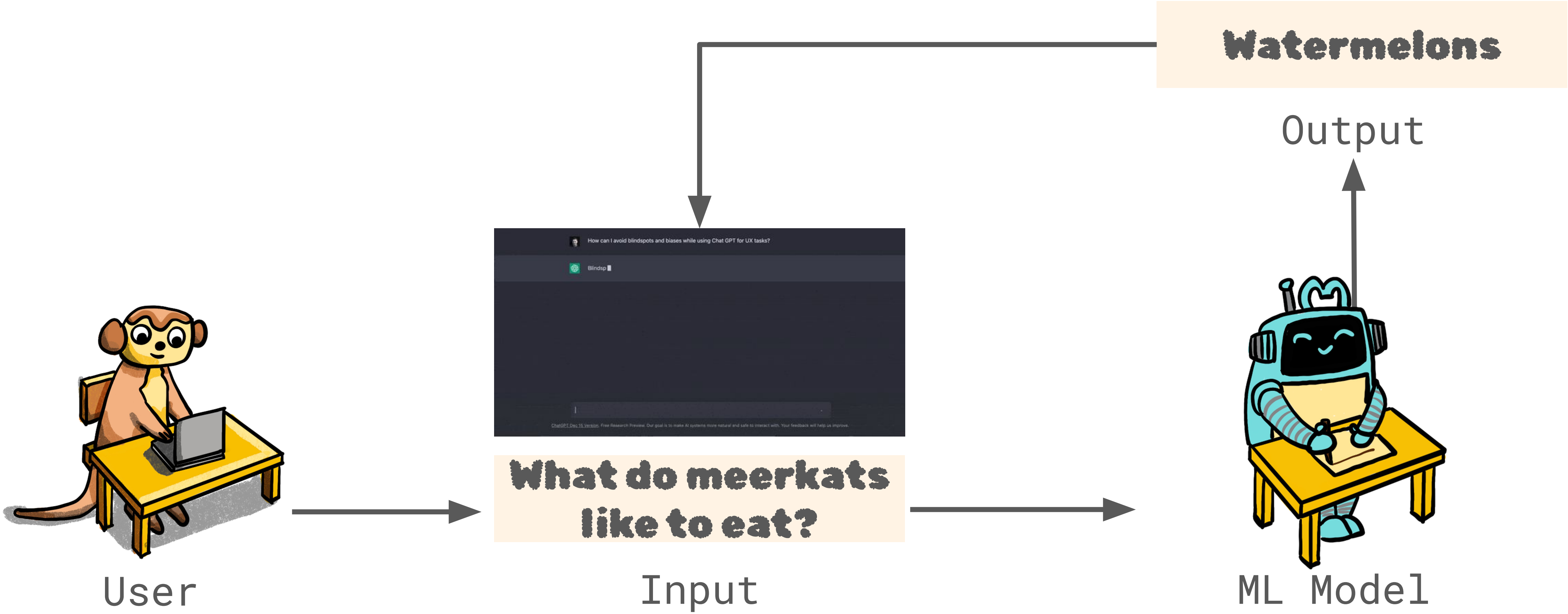
Input

ML Model

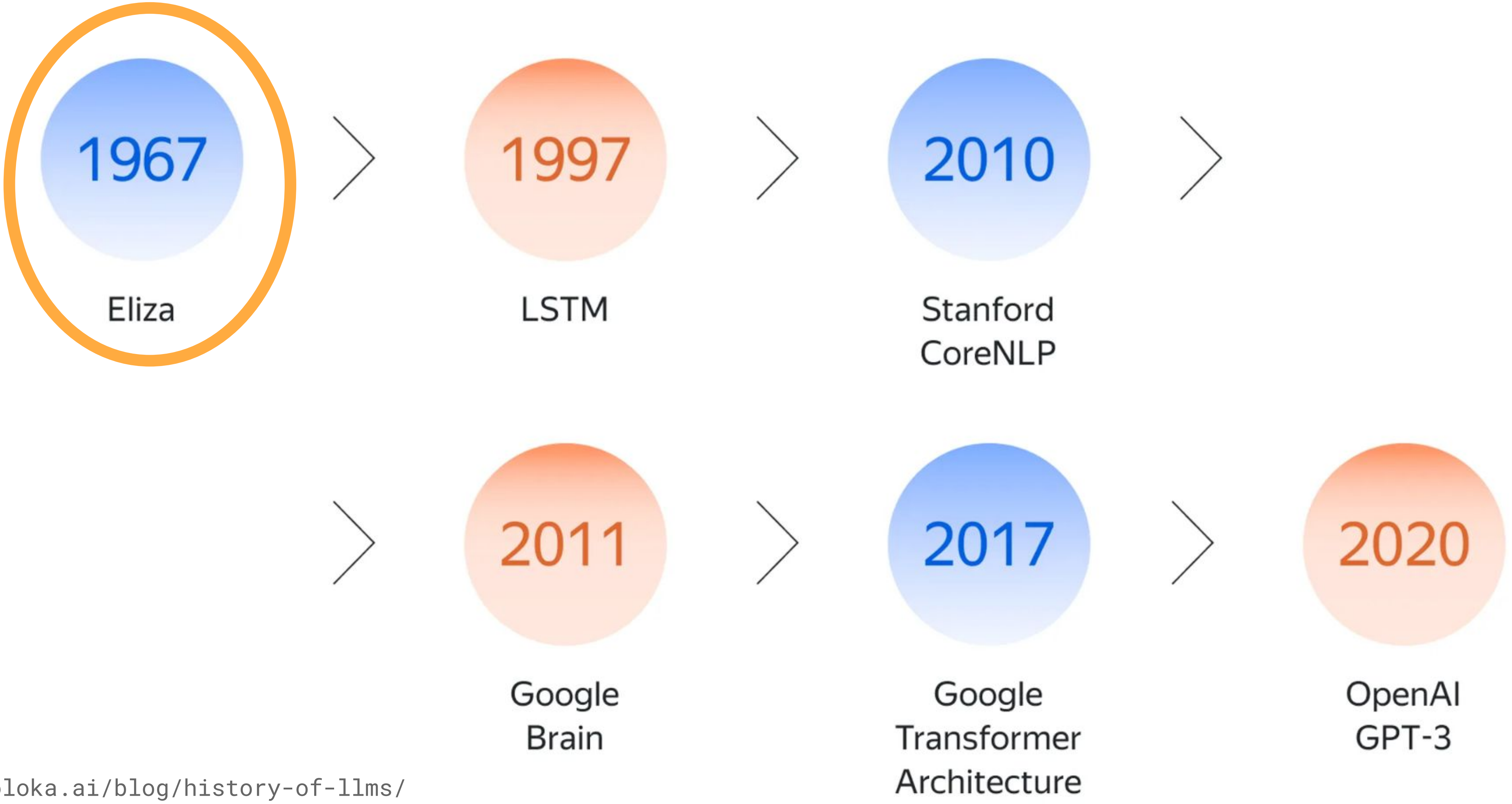
LLM - Large Language Models



LLM - Large Language Models

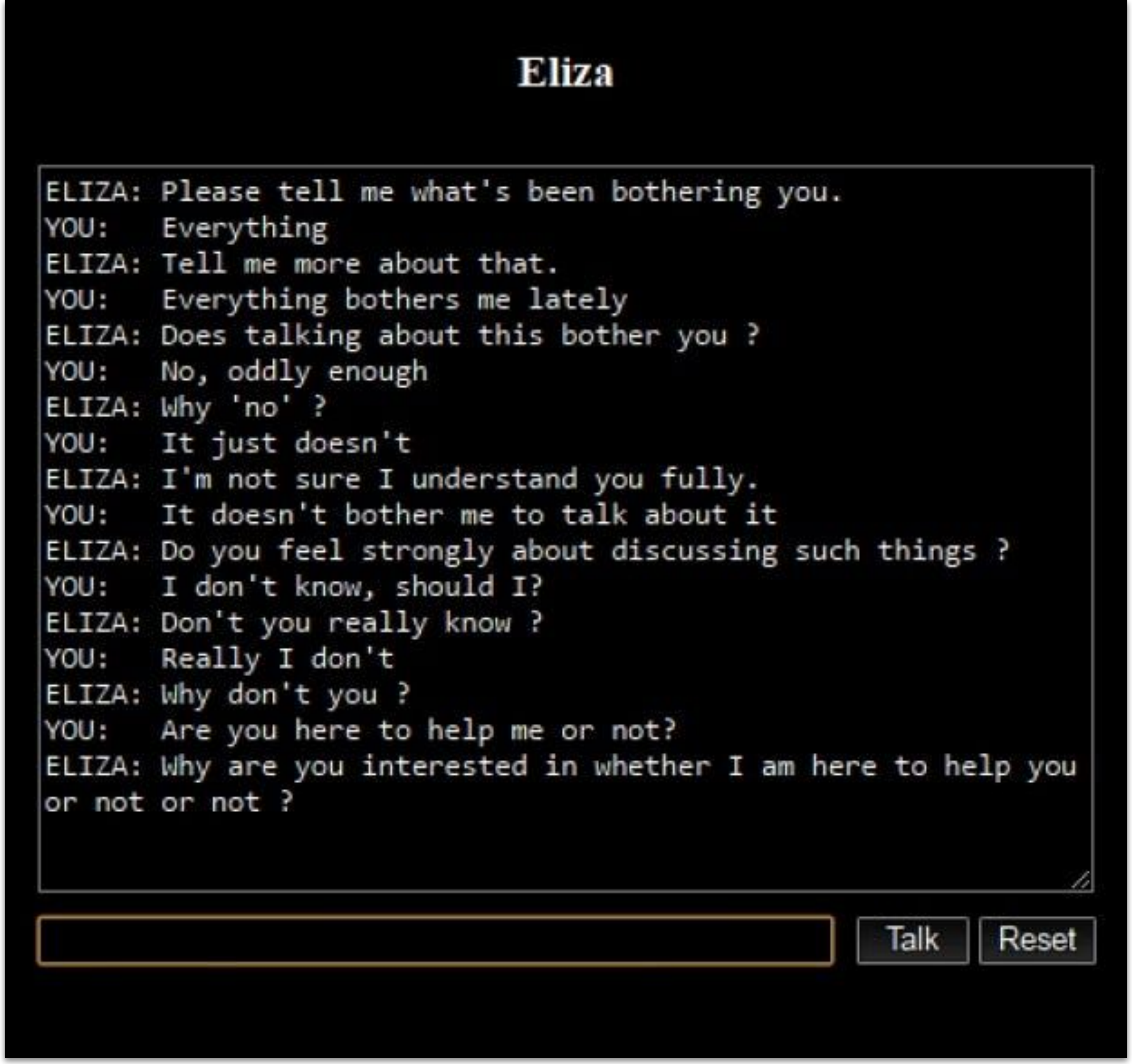


History of Large Language Models

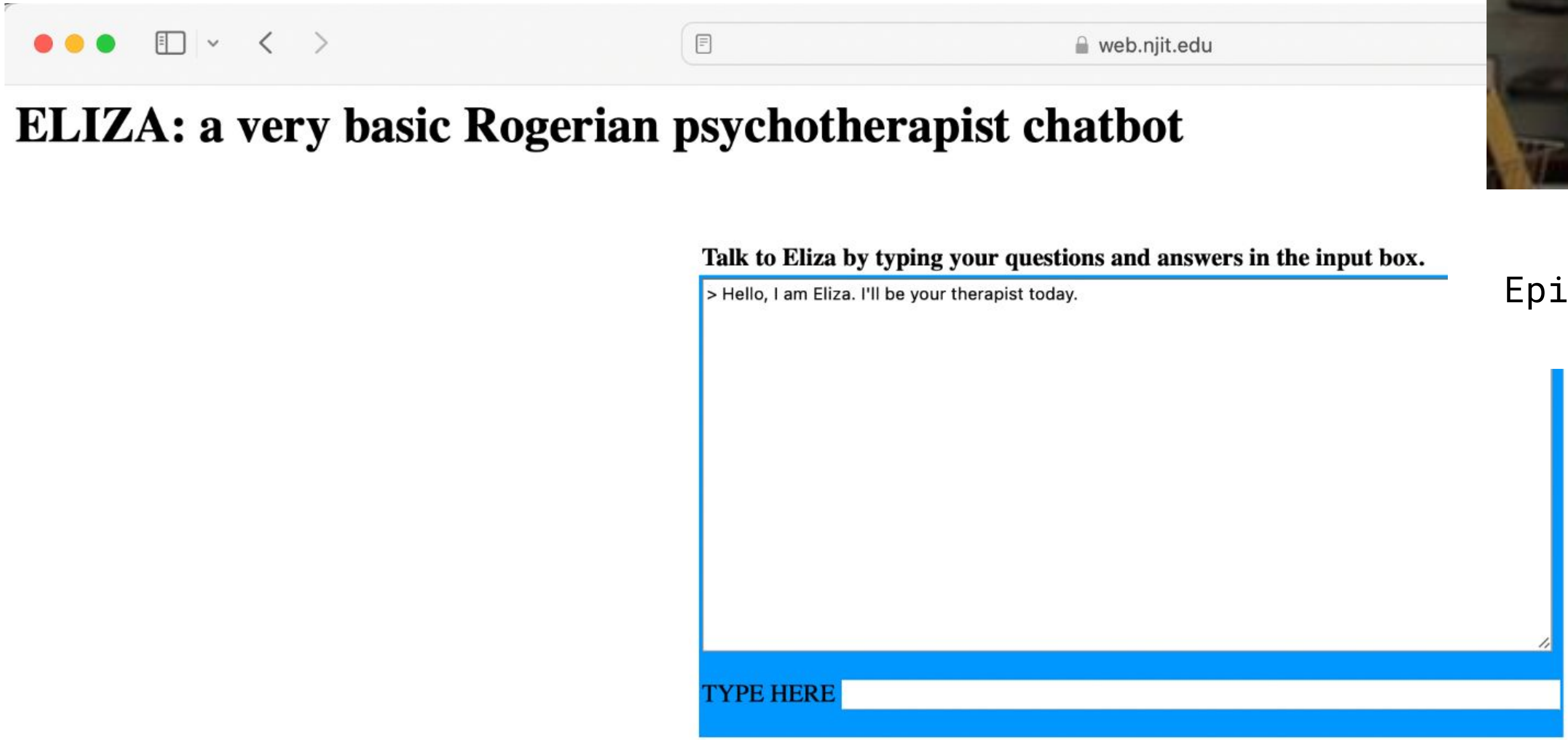


https://toloka.ai/blog/history-of-llms/

1967: ELIZA


Eliza
ELIZA: Please tell me what's been bothering you.
YOU: Everything
ELIZA: Tell me more about that.
YOU: Everything bothers me lately
ELIZA: Does talking about this bother you ?
YOU: No, oddly enough
ELIZA: Why 'no' ?
YOU: It just doesn't
ELIZA: I'm not sure I understand you fully.
YOU: It doesn't bother me to talk about it
ELIZA: Do you feel strongly about discussing such things ?
YOU: I don't know, should I?
ELIZA: Don't you really know ?
YOU: Really I don't
ELIZA: Why don't you ?
YOU: Are you here to help me or not?
ELIZA: Why are you interested in whether I am here to help you or not or not ?
Talk
Reset

1967: ELIZA



Young Sheldon
Episode - "A Computer, a Plastic Pony, and a Case of Beer

https://web.njit.edu/~ronkowit/eliza.html

History of Large Language Models

1967
Eliza

1997
LSTM

2010
Stanford CoreNLP

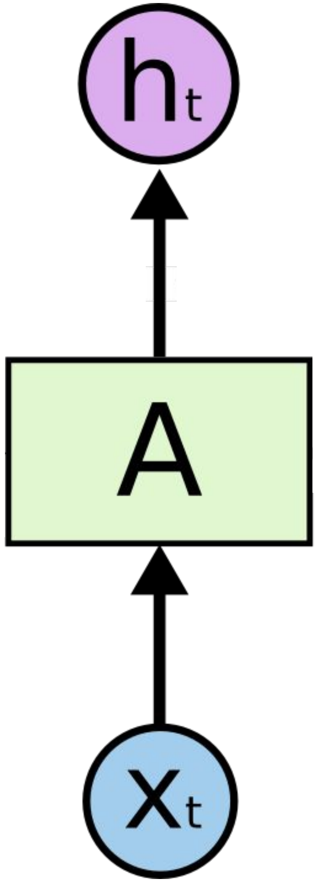
2011
Google Brain

2017
Google Transformer Architecture

2020
OpenAI GPT-3

https://toloka.ai/blog/history-of-llms/

LSTM is a type of Recurrent Neural Network



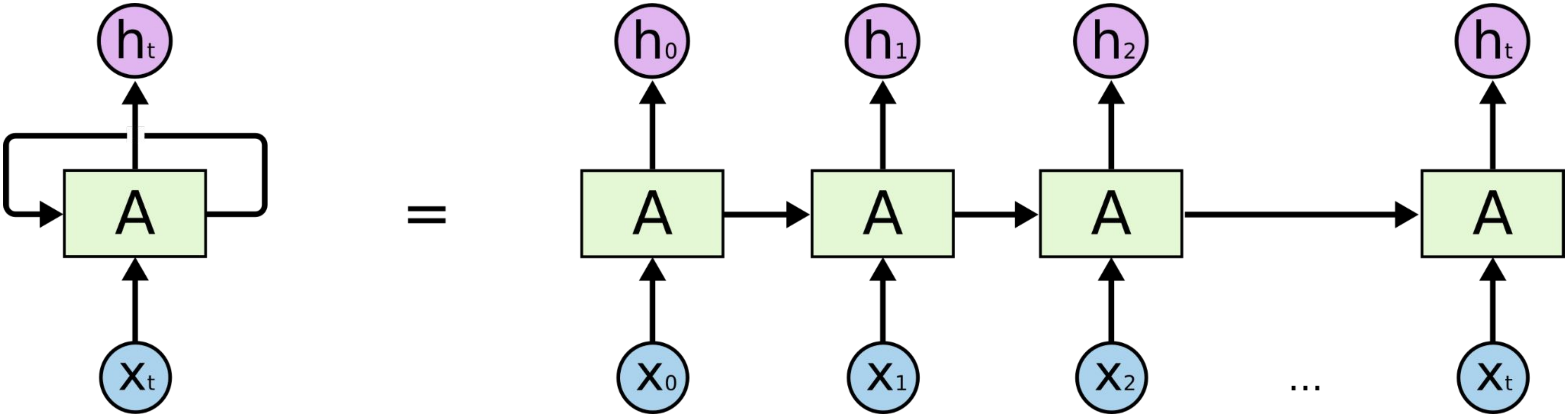
output

network A

input

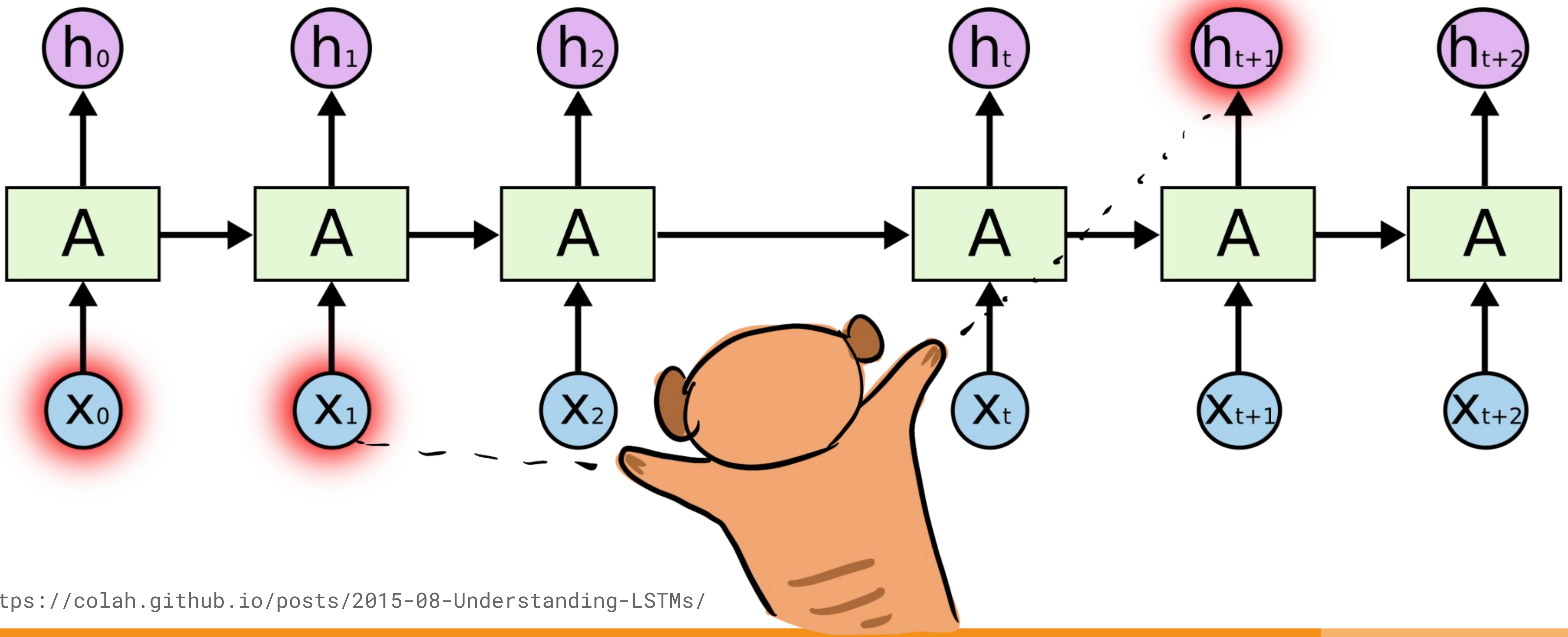
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

1997: Unrolled RNN



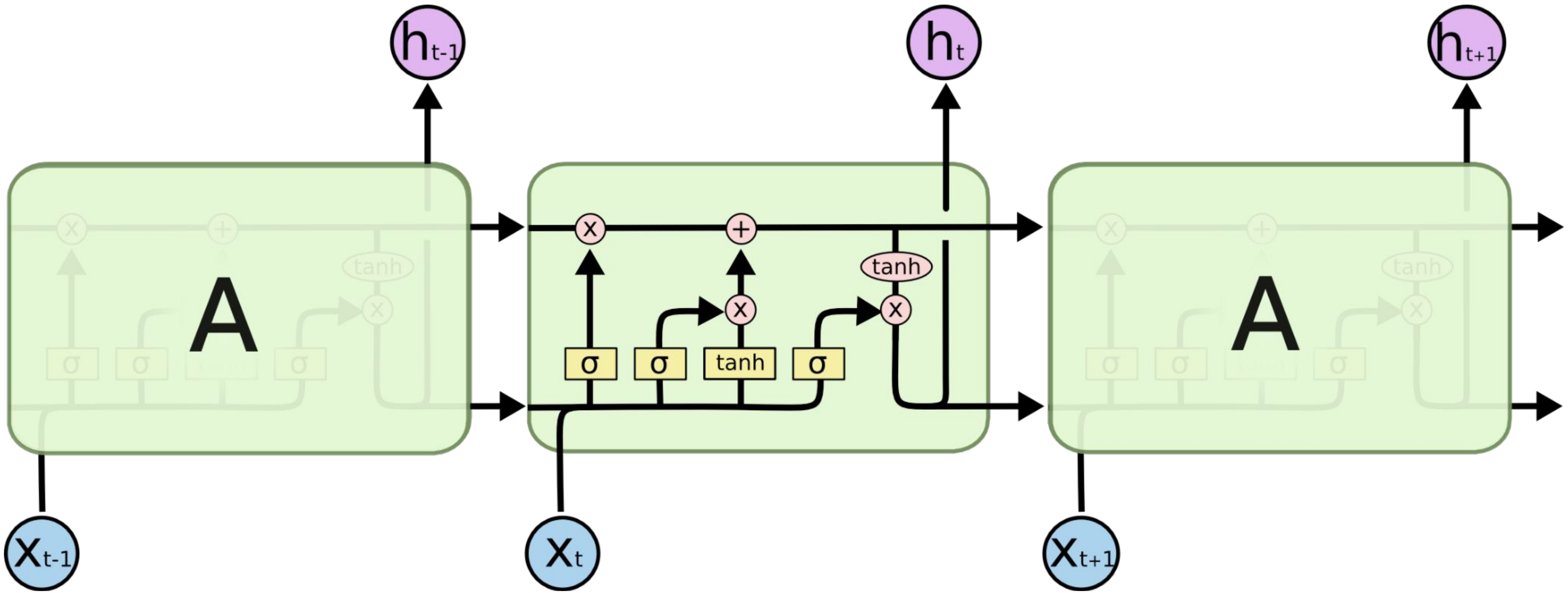
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

1997: RNN Problem



https://colah.github.io/posts/2015-08-Understanding-LSTMs/

1997: LSTM

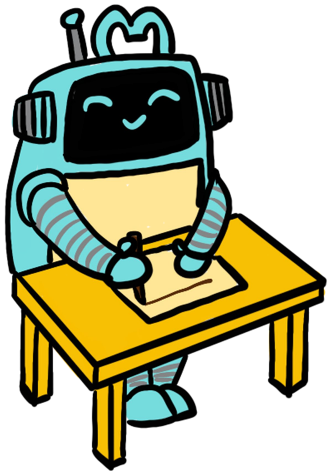


https://colah.github.io/posts/2015-08-Understanding-LSTMs/

1997: LSTM - **L**ong **S**hort-**T**erm **M**emory

- Remembers info over long periods

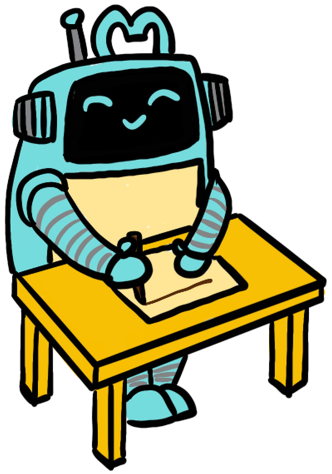
Hello there! I'm a little meerkat, my name is Steven. And you can usually find me standing on my hind legs, keeping a lookout for my family. We live together in large groups called 'mobs', and it's my job to make sure everyone is safe while they're busy digging or playing. You know, life in the desert can be quite an adventure! We love to bask in the sun, but we're always ready to dart into our burrows if danger comes close. Oh, and we are super curious! I often find myself nosing around, exploring every nook and cranny of our sandy home. We meerkats are also great at working together. Whether it’s finding food or taking care of our little pups, teamwork is our secret to a happy life. And guess what? We have a special way of talking to each other with chirps, barks, and purrs. It's like our own secret language! So, if you ever hear a bunch of chattering and see a group of us looking around intently, just know, we're having a lively chat about our day's adventures and watching out for each other, because that's what families do



ML Model

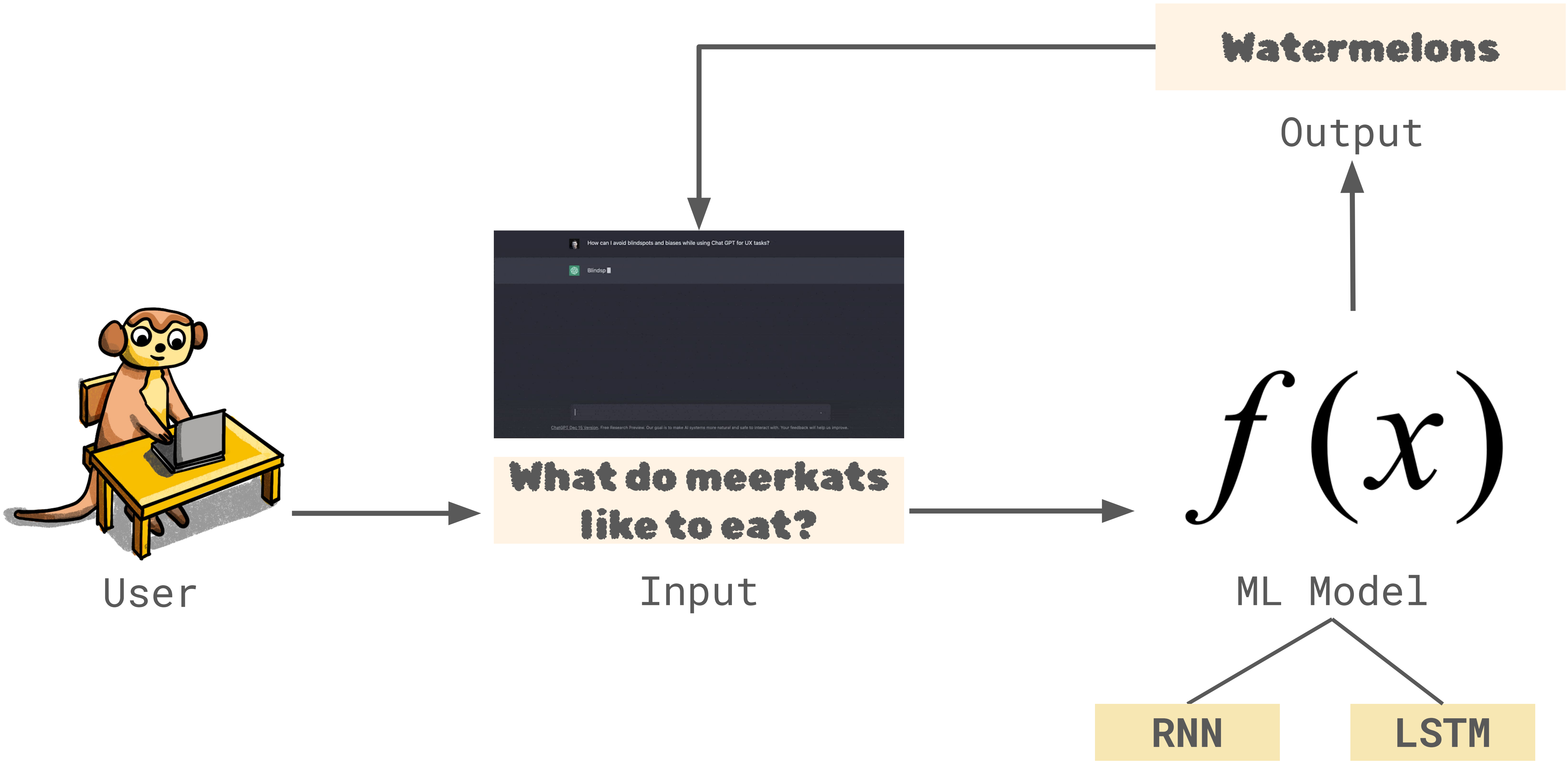
1997: LSTM - **L**ong **S**hort-**T**erm **M**emory

- Remembers info over long periods



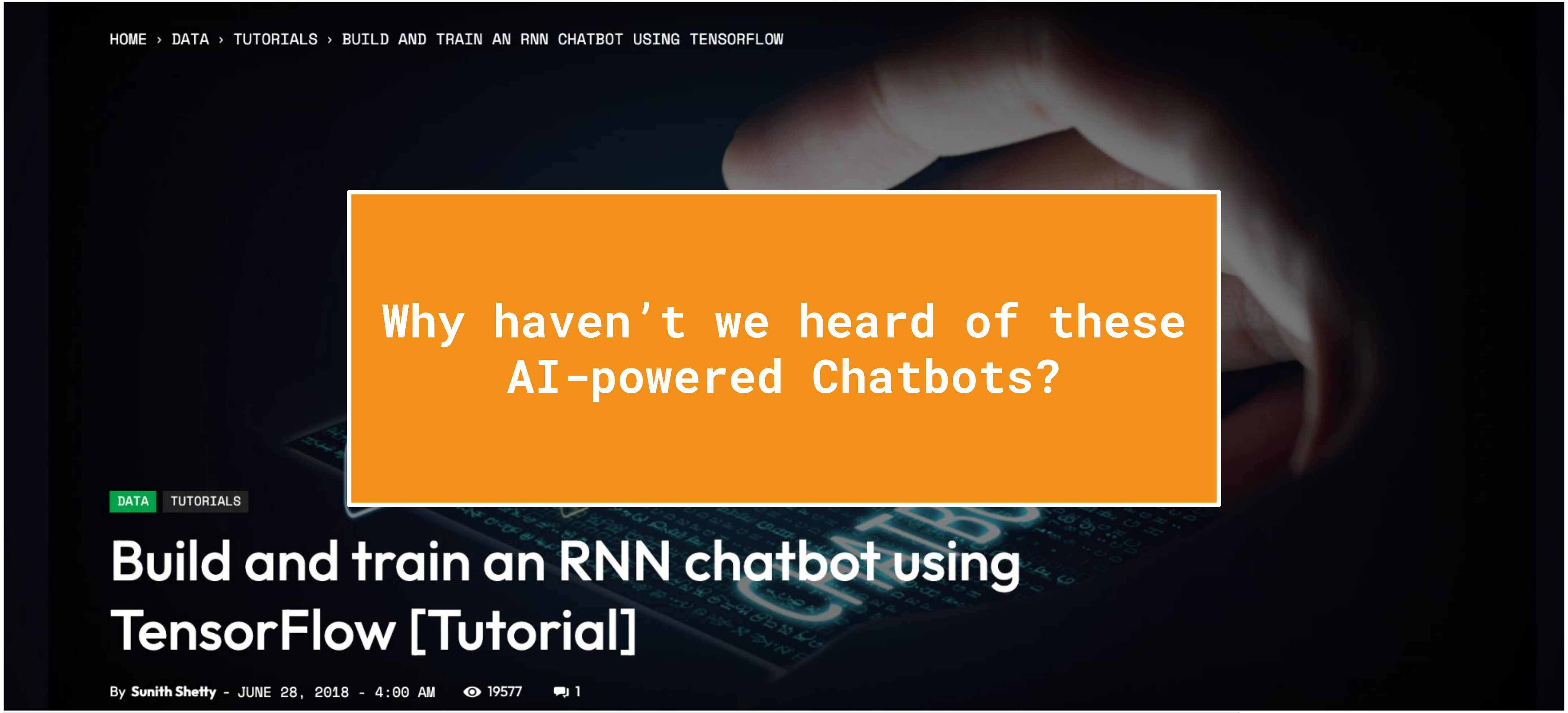
ML Model

Hello there! I'm a little meerkat, my name is Steven. And you can usually find me standing on my hind legs, keeping a lookout for my family. We live together in large groups called 'mobs', and it's my job to make sure everyone is safe while they're busy digging or playing. You know, life in the desert can be quite an adventure! We love to bask in the sun, but we're always ready to dart into our burrows if danger comes close. Oh, and we are super curious! I often find myself nosing around, exploring every nook and cranny of our sandy home. We meerkats are also great at working together. Whether it’s finding food or taking care of our little pups, teamwork is our secret to a happy life. And guess what? We have a special way of talking to each other with chirps, barks, and purrs. It's like our own secret language! So, if you ever hear a bunch of chattering and see a group of us looking around intently, just know, we're having a lively chat about our day's adventures and watching out for each other, because that's what families do

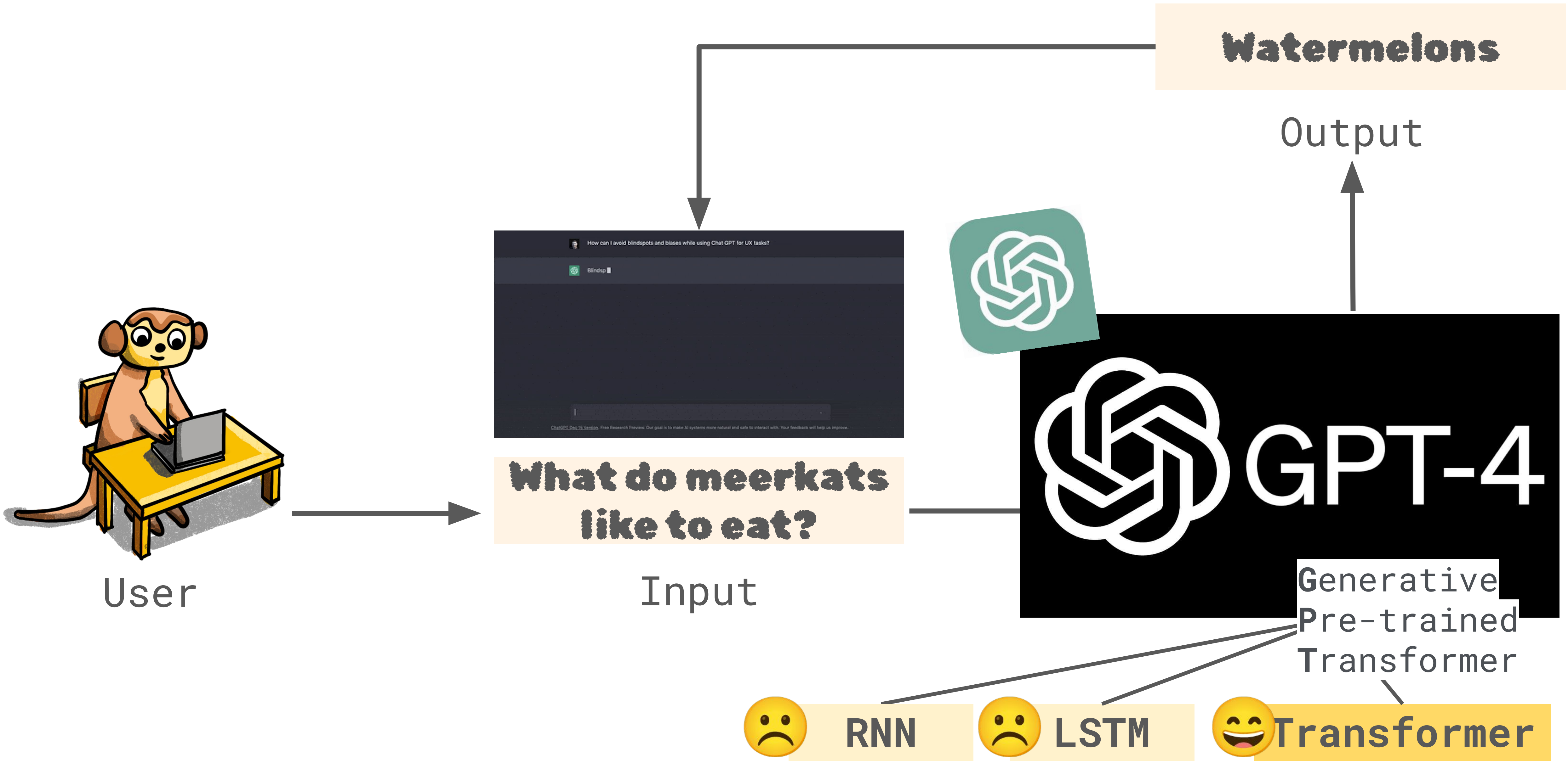


Watermelons
Output
How can I avoid blindspots and biases while using Chat GPT for UX tasks?
Blindsp
What do meerkats like to eat?
User
Input
$f(x)$
ML Model
RNN
LSTM

Chatbots w/ Different ML Model



https://hub.packtpub.com/build-and-train-rnn-chatbot-using-tensorflow/


Watermelons
Output
How can I avoid blindspots and biases while using Chat GPT for UX tasks?
Blindsp
What do meerkats like to eat?
Input
User
GPT-4
Generative
Pre-trained
Transformer
RNN
LSTM
Transformer

History of Large Language Models

1967 — Eliza

1997 — LSTM

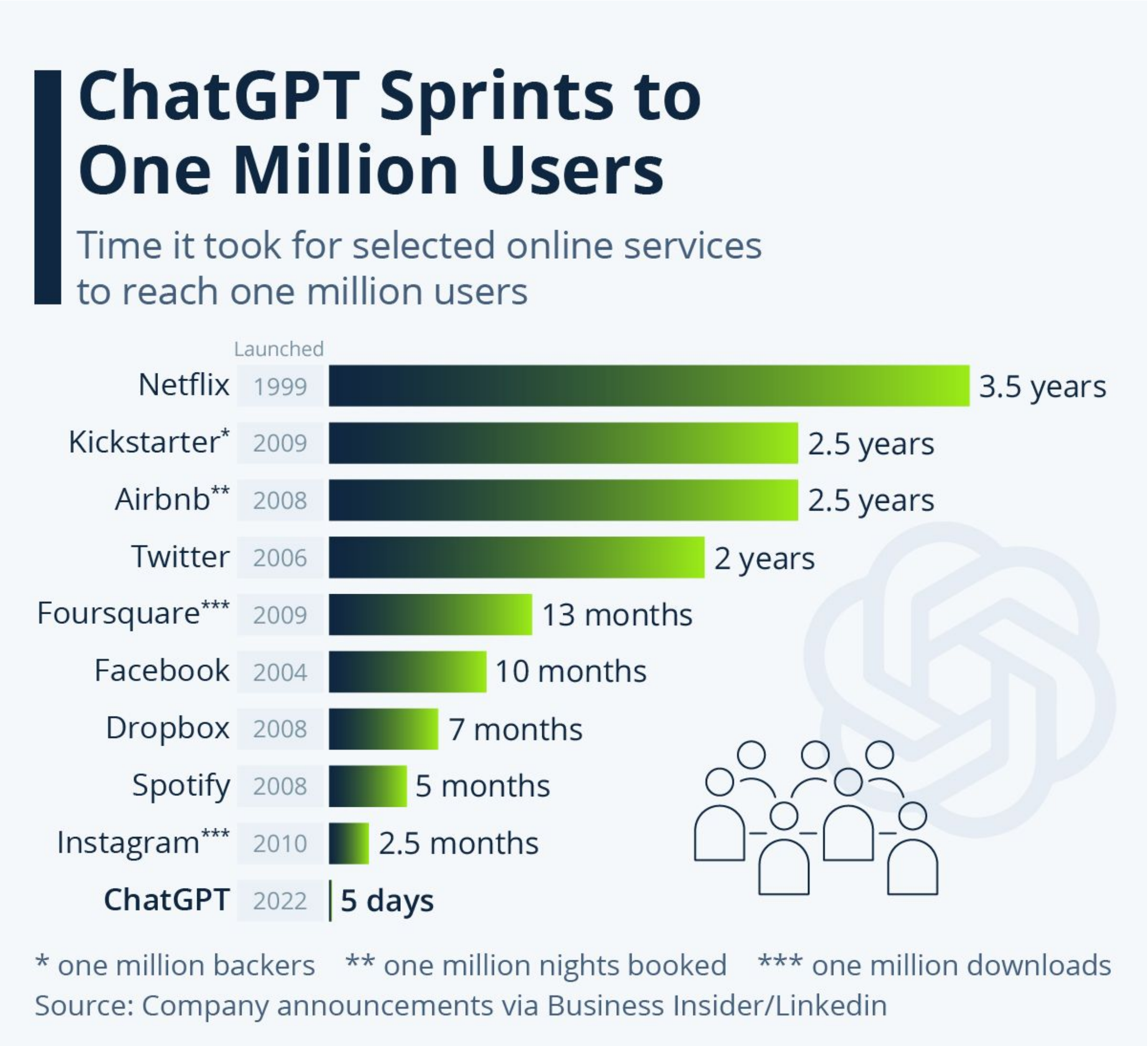
2010 — Stanford CoreNLP

2011 — Google Brain

2017 — Google Transformer Architecture

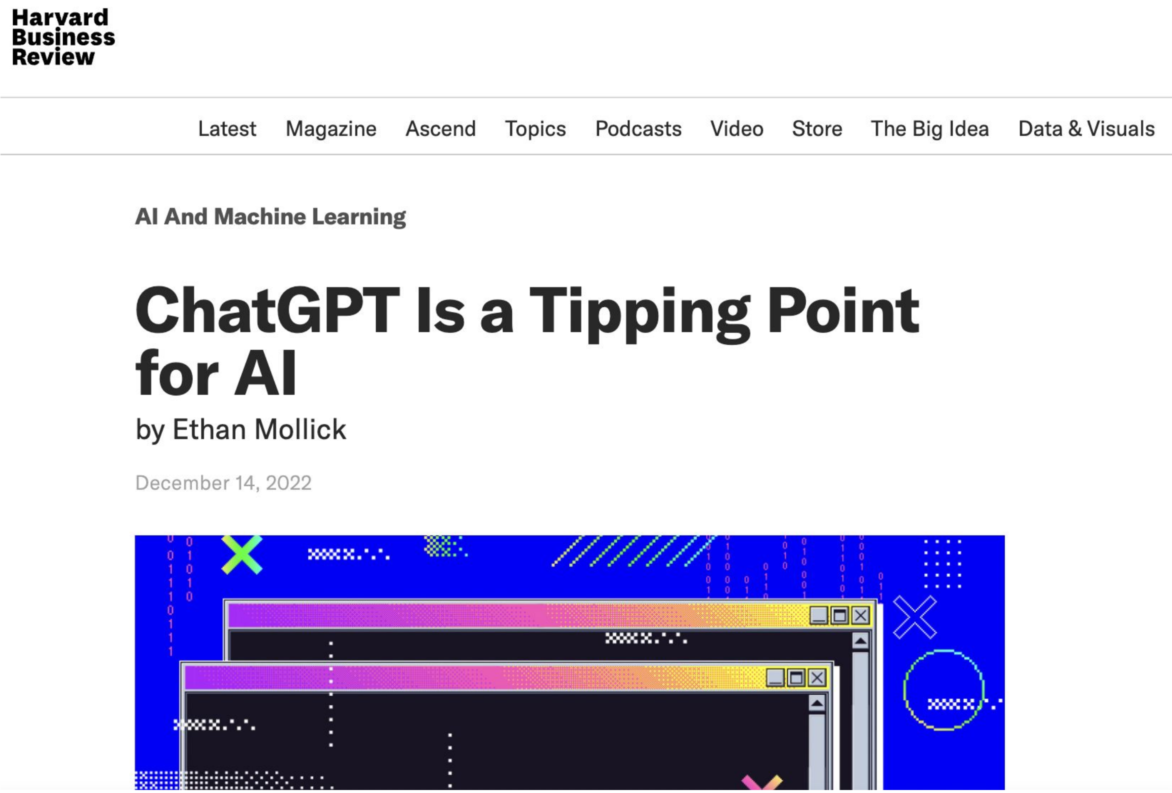
2020 — OpenAI GPT-3

h



https://www.digitalinformationworld.com/2023/01/chat-gpt-achieved-one-million-users-in.html

statista


Harvard Business Review

Latest Magazine Ascend Topics Podcasts Video Store The Big Idea Data & Visuals

AI And Machine Learning

ChatGPT Is a Tipping Point for AI

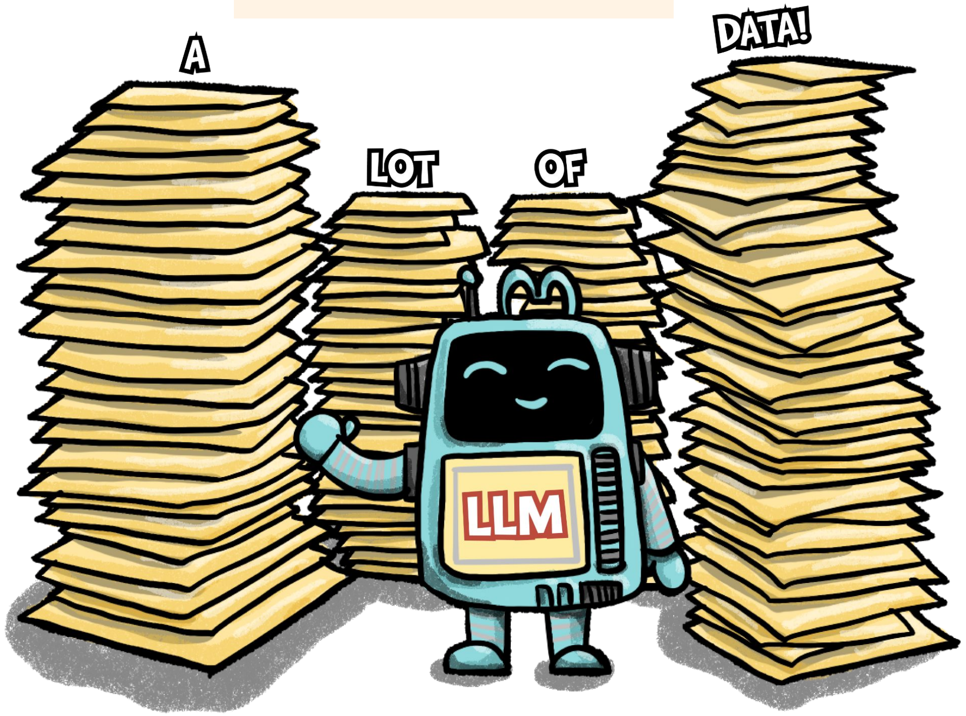
by Ethan Mollick

December 14, 2022

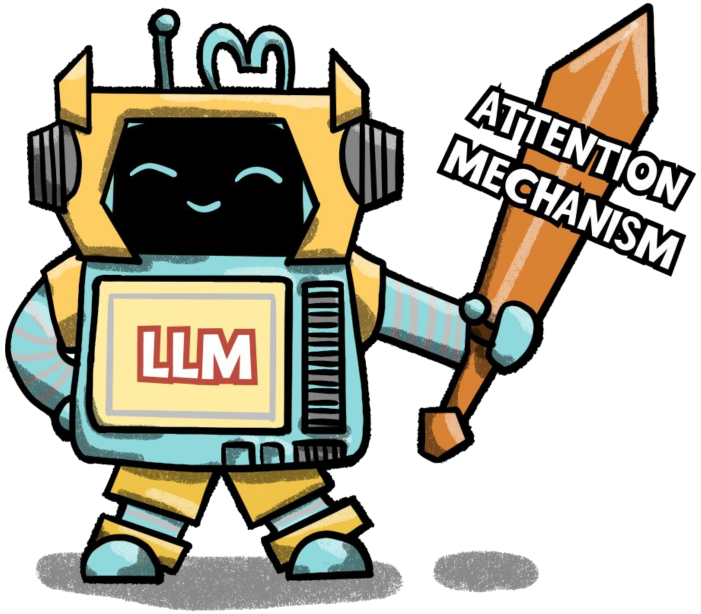
https://hbr.org/2022/12/chatgpt-is-a-tipping-point-for-ai

ChatGPT's Success

Data

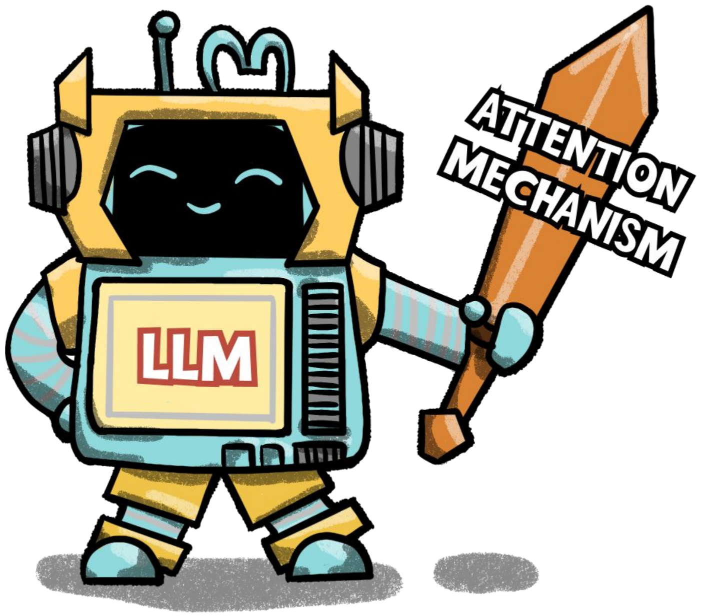


Model Architecture Advancements

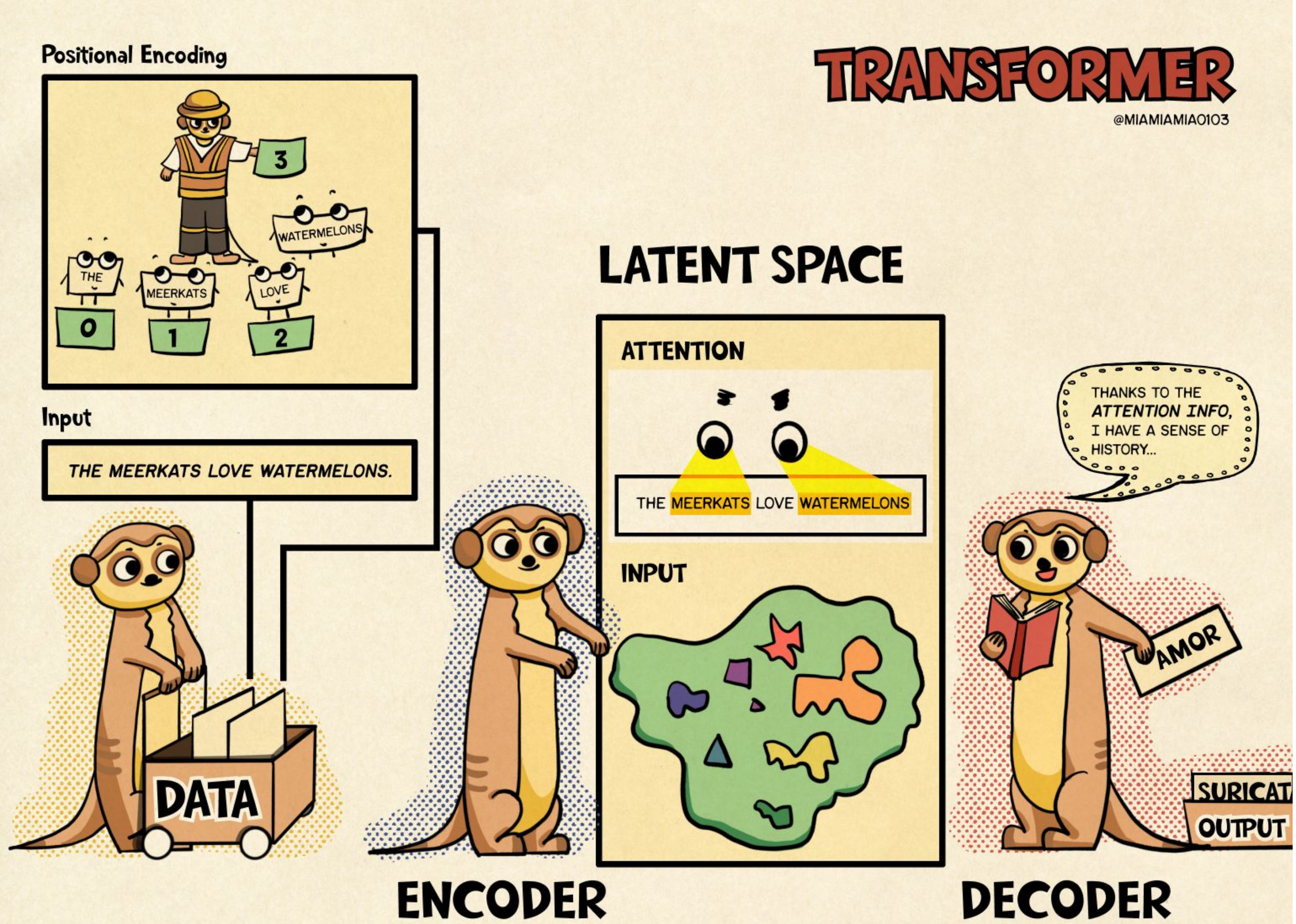


LLM

Model Architecture Advancements

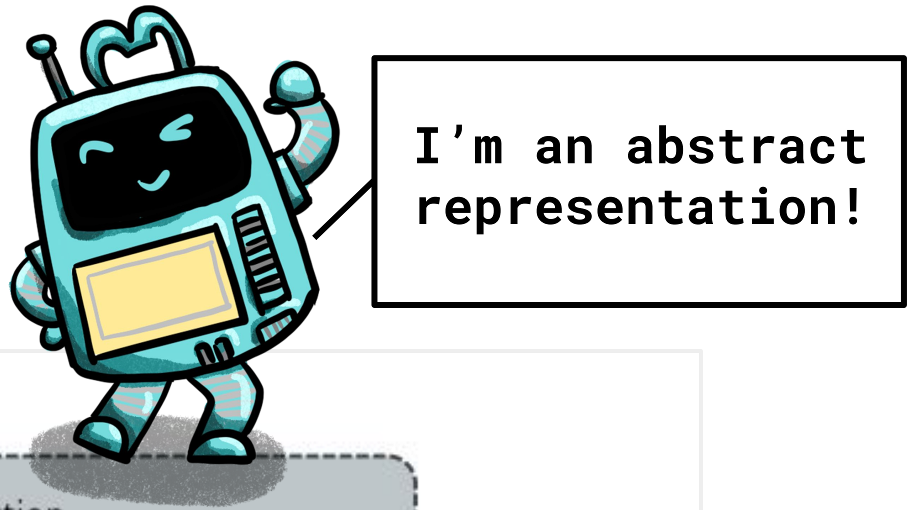


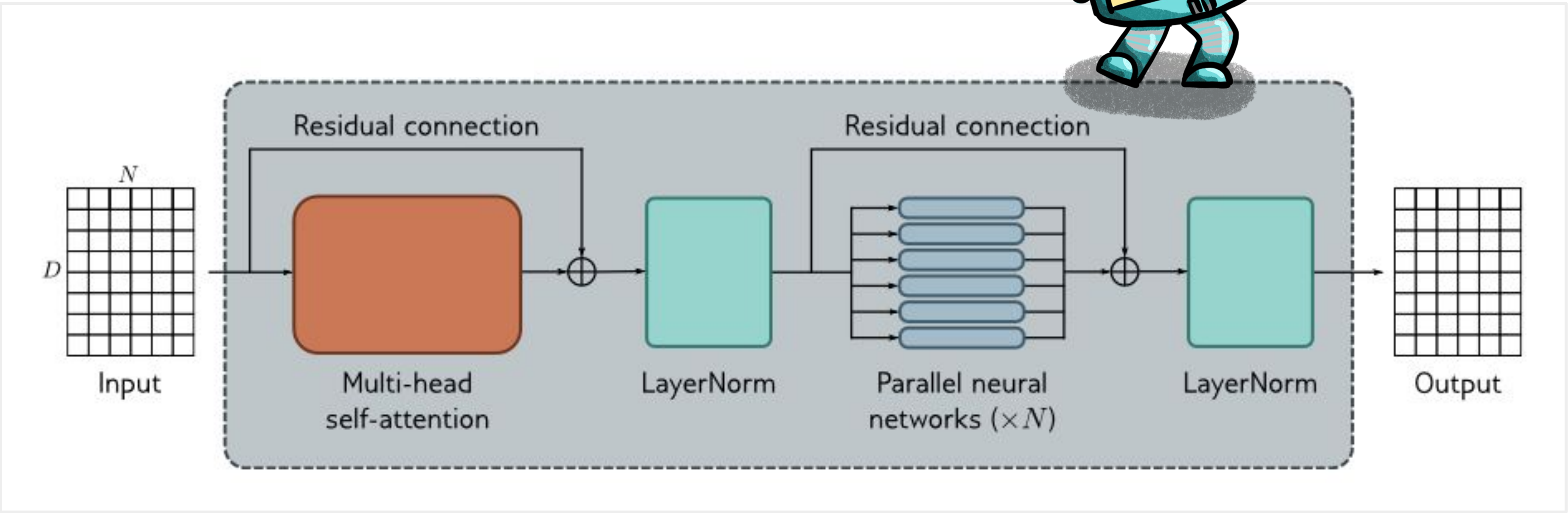
- **Transformer**
  - Word Vectors
  - Attention Mechanism
  - Residual Connections
- **Text Generation**
  - Predict next token
- **RLHF**



Transformers

Transformer Architecture

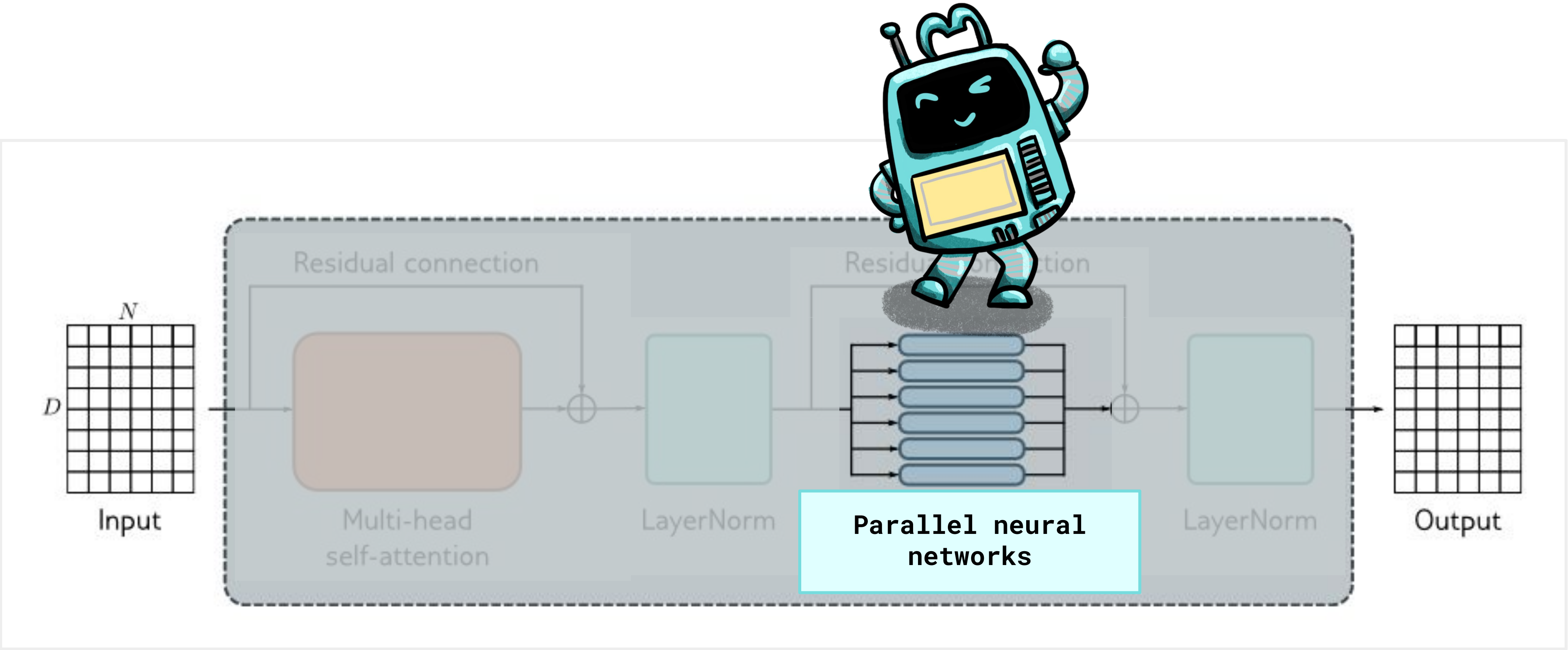




Simon J.D. Prince (2023). Understanding Deep Learning

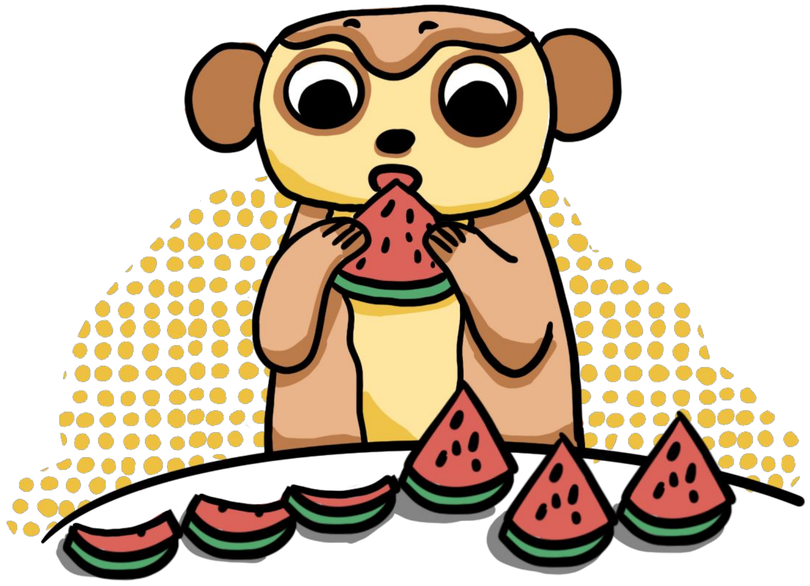
Transformer - Origin

Transformer - Parallel Neural Networks



Simon J.D. Prince (2023). Understanding Deep Learning

Processing

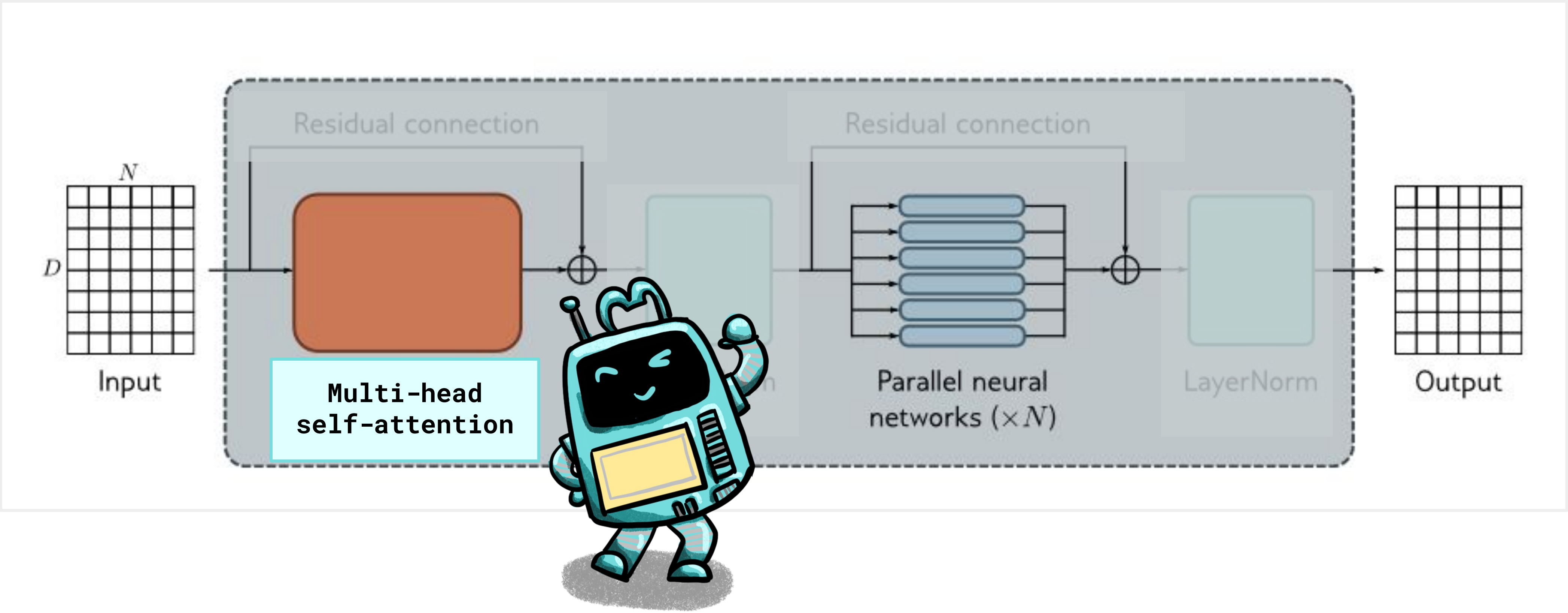


Sequential Processing

Parallel Processing

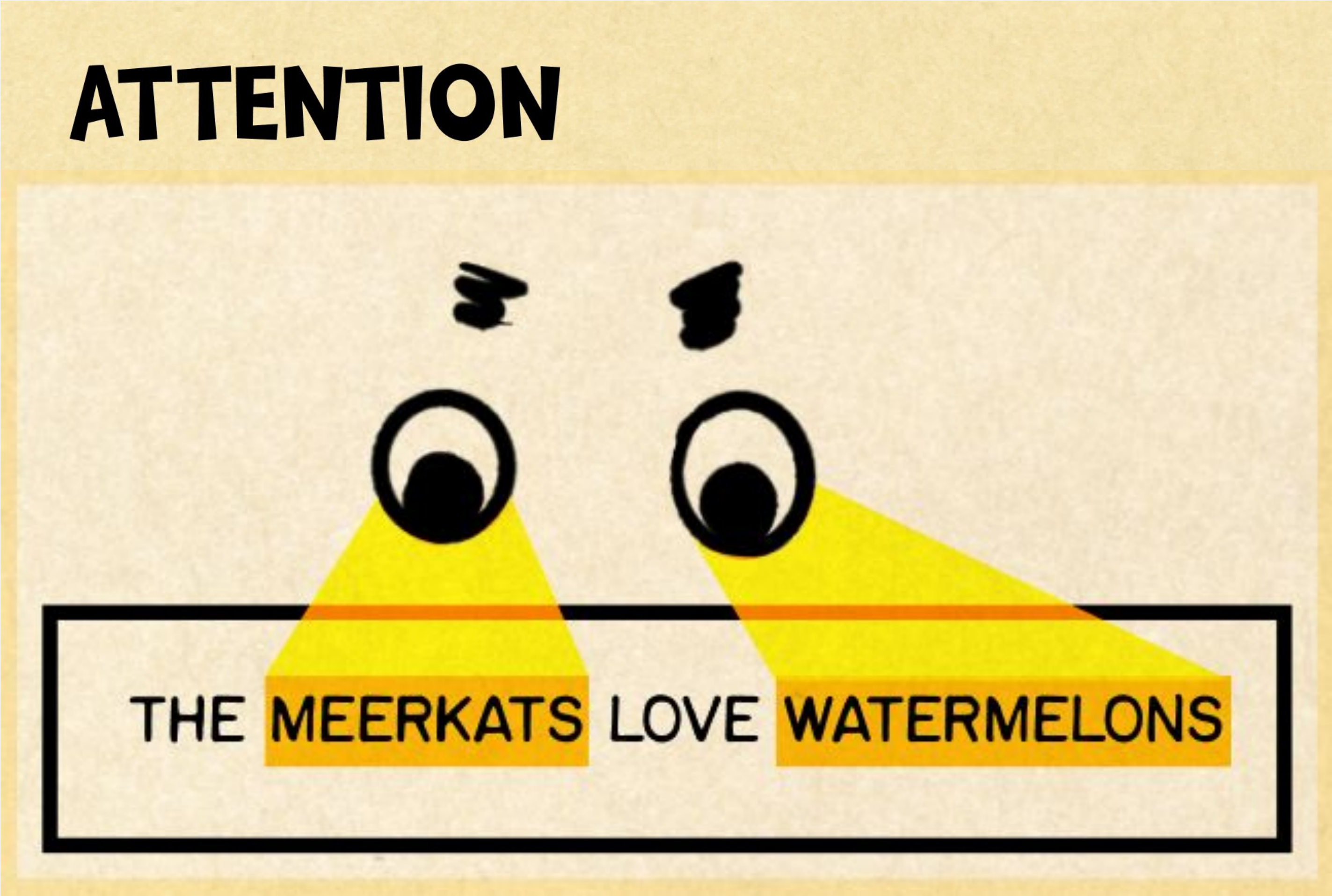
Simon J.D. Prince (2023). Understanding Deep Learning

Transformer - Attention



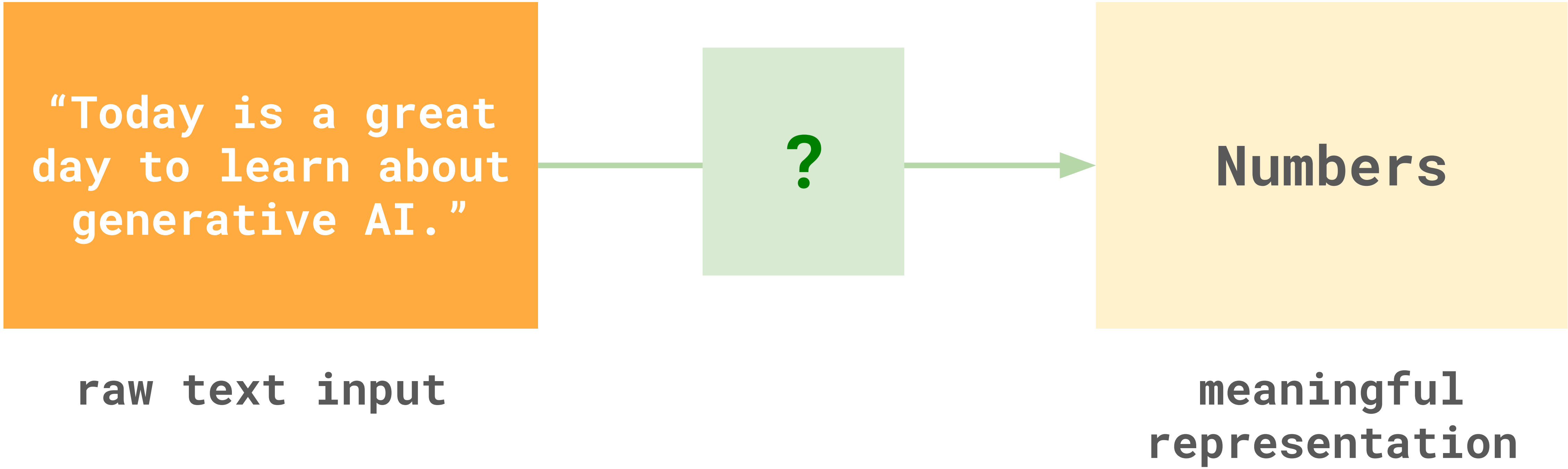
Simon J.D. Prince (2023). Understanding Deep Learning

Paying Attention



How do we work with raw text

We want meaningful numerical representation of text.

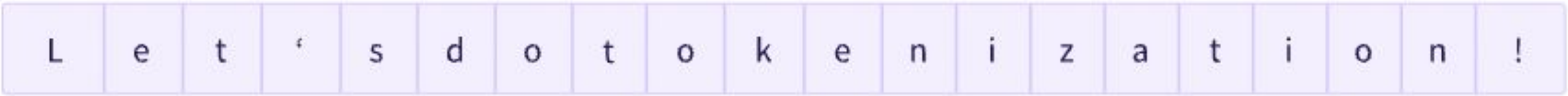


Introducing Tokenizer

Unknown artist

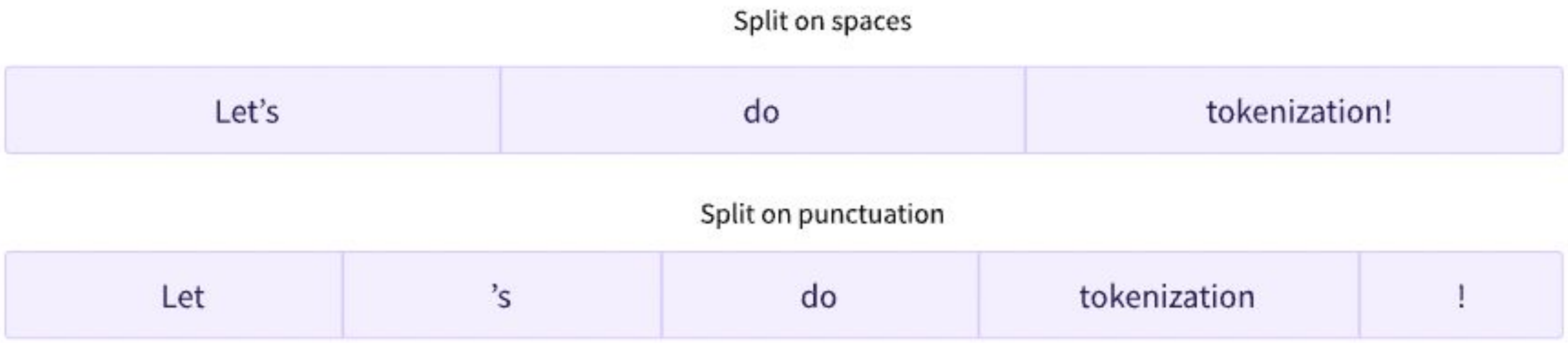
Types of Tokenizer: Character-based

Input: “Let’s do tokenization!”



Types of Tokenizer: Word-based

Input: “Let’s do tokenization!”

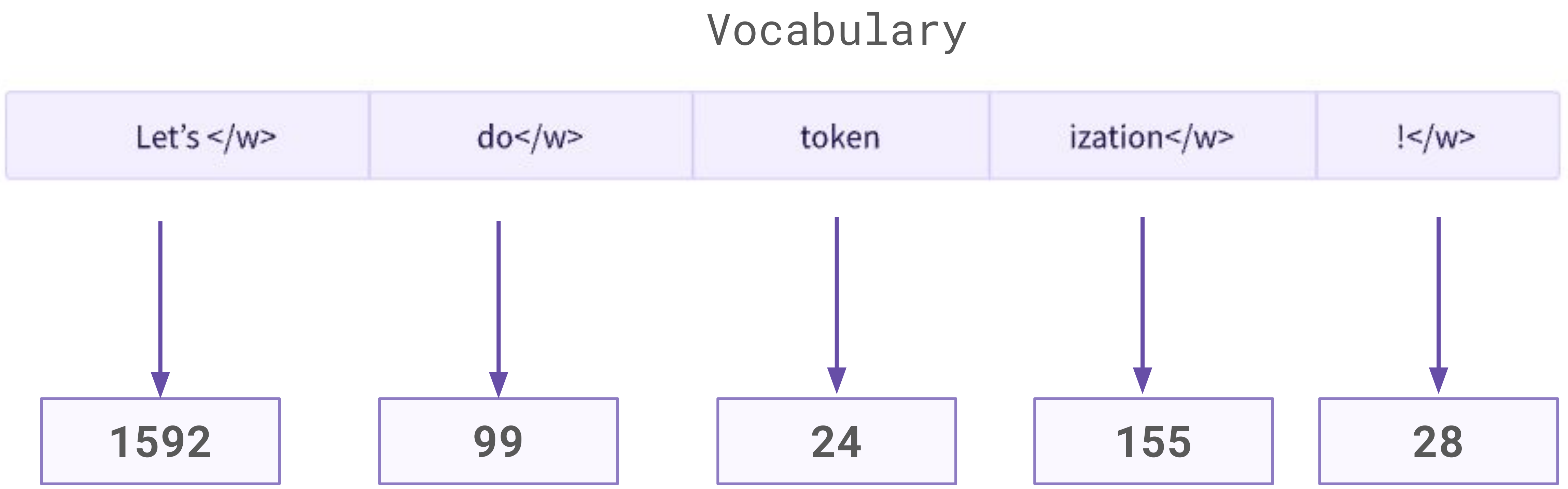


Types of Tokenizer: Subword-based

Input: “Let’s do tokenization!”

Let’s </w>	do</w>	token	ization</w>	!</w>

Tokens → IDs

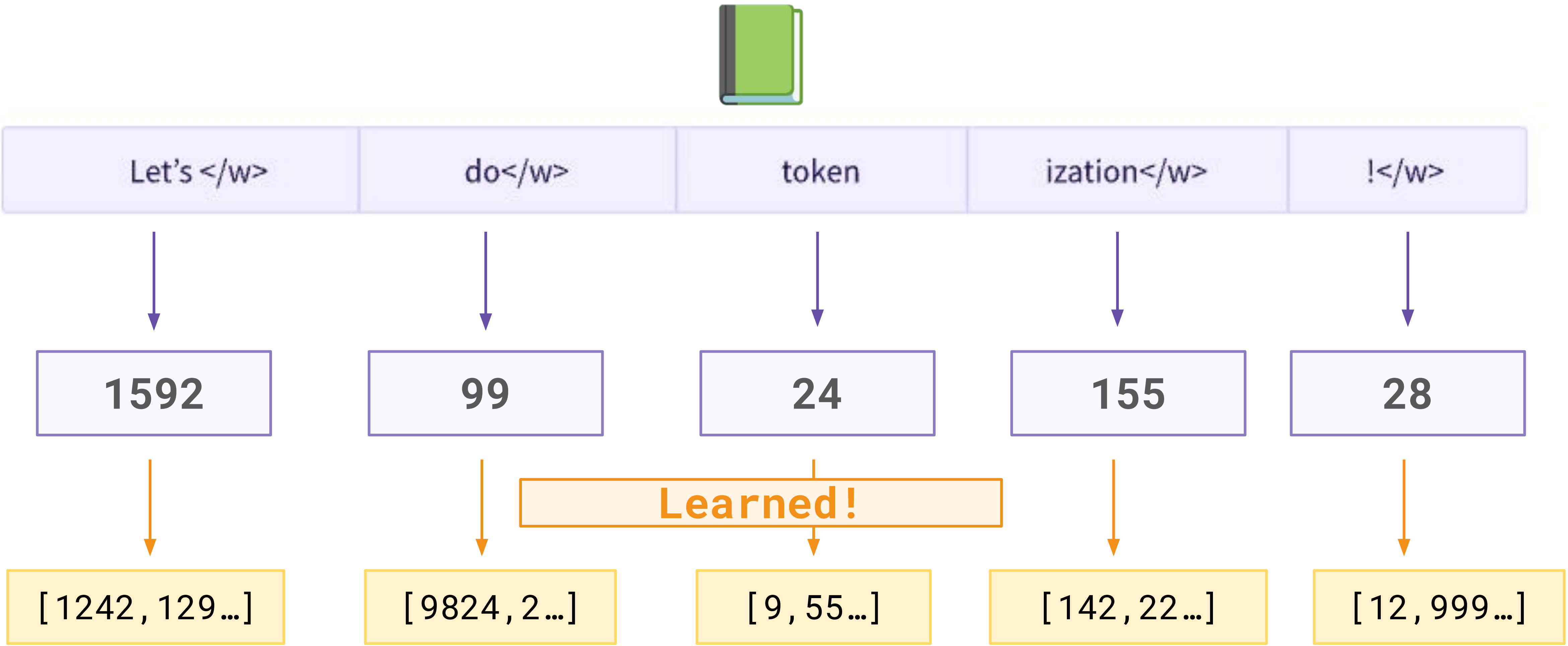


Tokenization IRL

sigasia_tokenization.py

```python
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")
sequence = "Meerkats are small, social, and always ready for adventures"
tokens = tokenizer.tokenize(sequence)
# tokens = ['Me', '##er', '##kat', '##s', 'are',
# 'small', ',', 'social', ',', 'and', 'always', 'ready', 'for', 'adventures']
```

Tokens → IDs → Embeddings

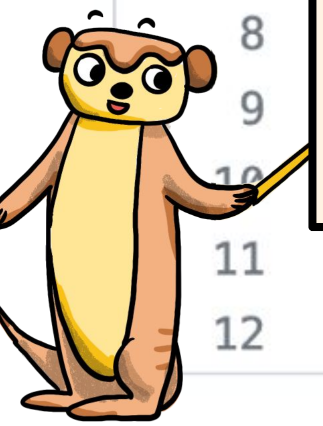


Embedding = n-dimensional vectors

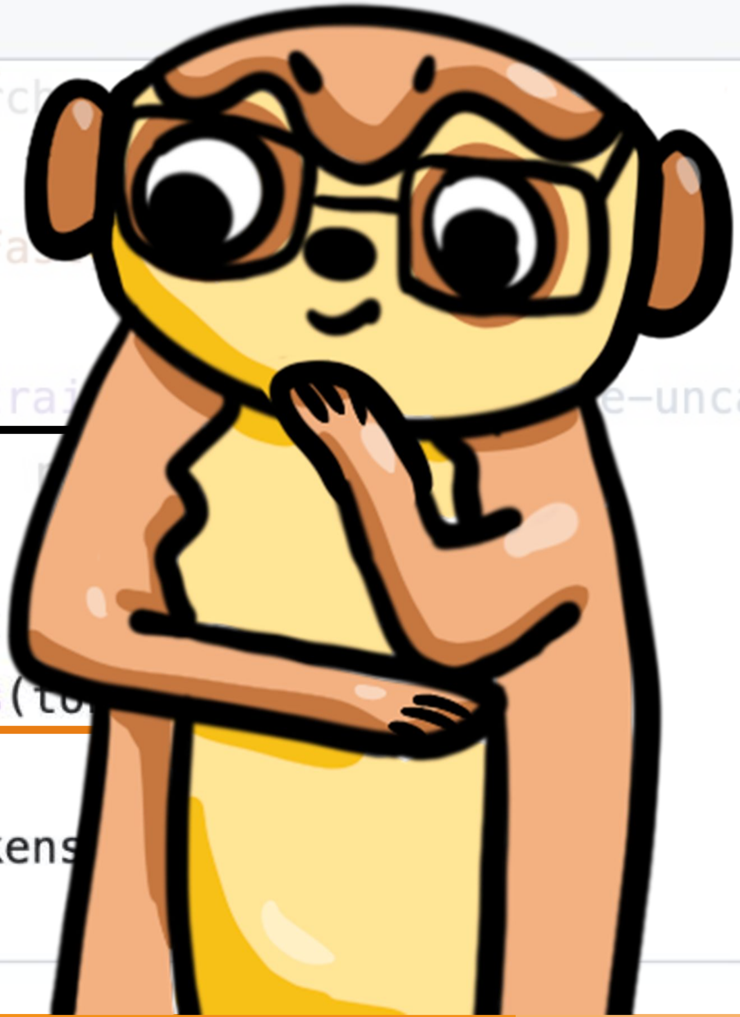
Tokens → IDs → Embeddings IRL

sigasia_embeddings.py

```
for e in model.embeddings.word_embeddings(tokens
    print("This is an embedding!", e)
```



Why embeddings?

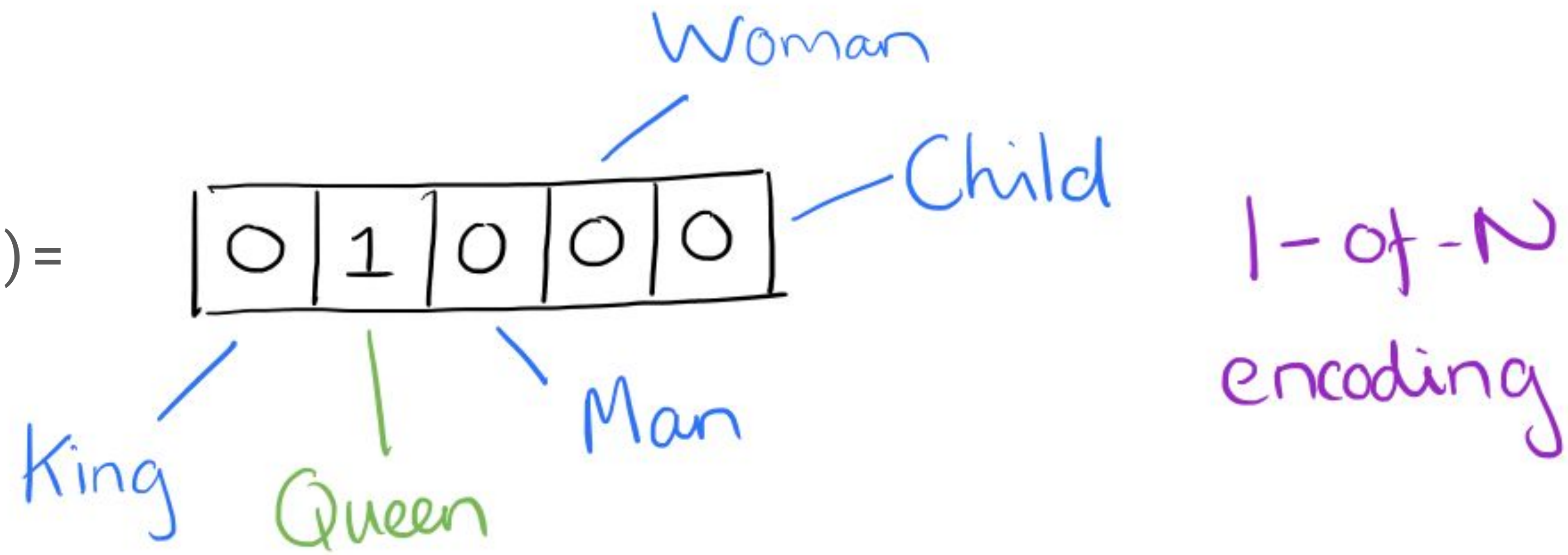


Embedding (Word Vector) Motivation

Text vector

Vocabulary: [“King”, “Queen”, “Man”, “Woman”, Child”]

encode(queen)=



https://blog.acolyer.org/2016/04/21/the-amazing-power-of-word-vectors/

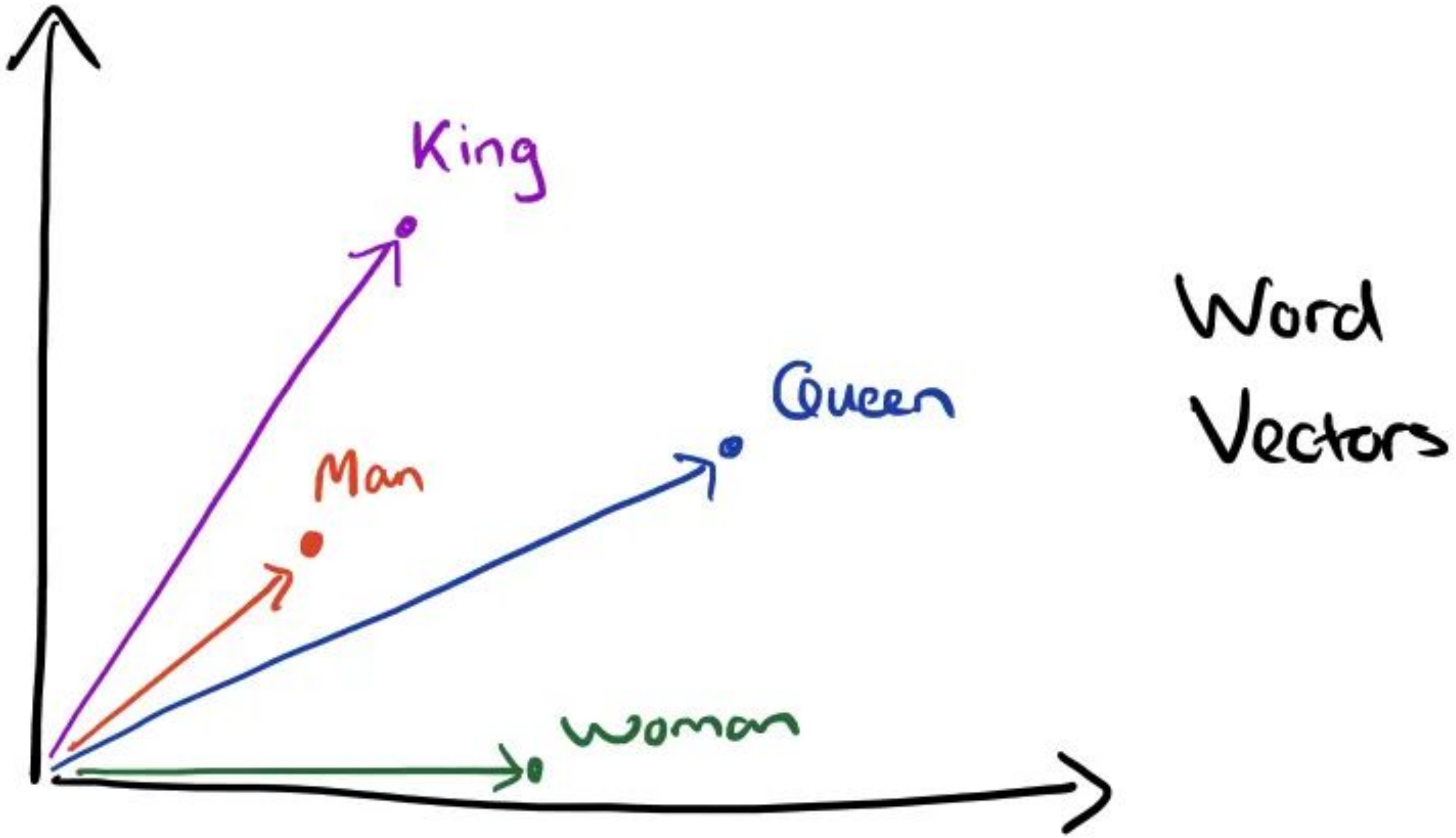
Text vector with distributed weights

Learned

elements of a word

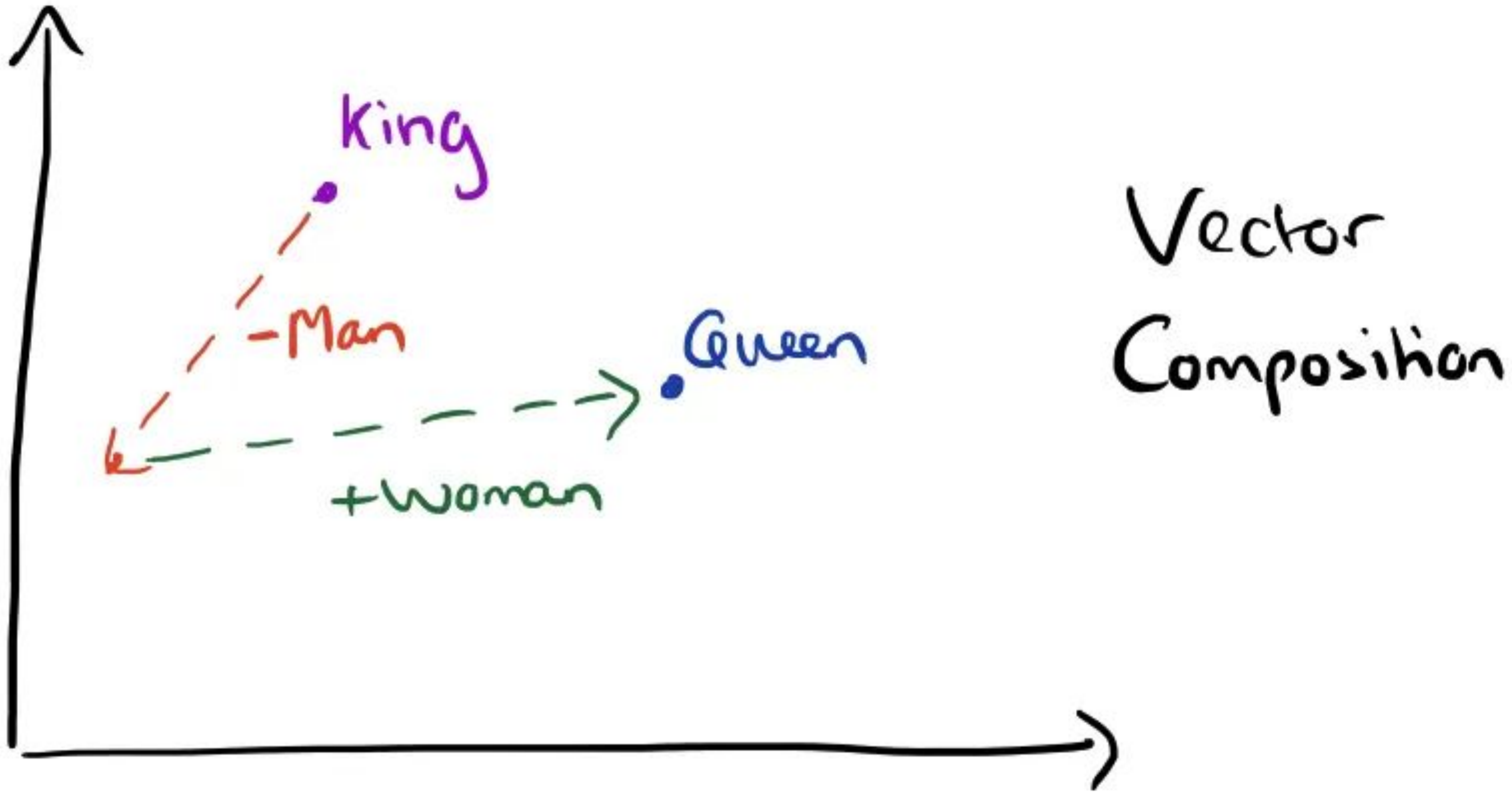
https://blog.acolyer.org/2016/04/21/the-amazing-power-of-word-vectors/

Word Vectors



https://blog.acolyer.org/2016/04/21/the-amazing-power-of-word-vectors/

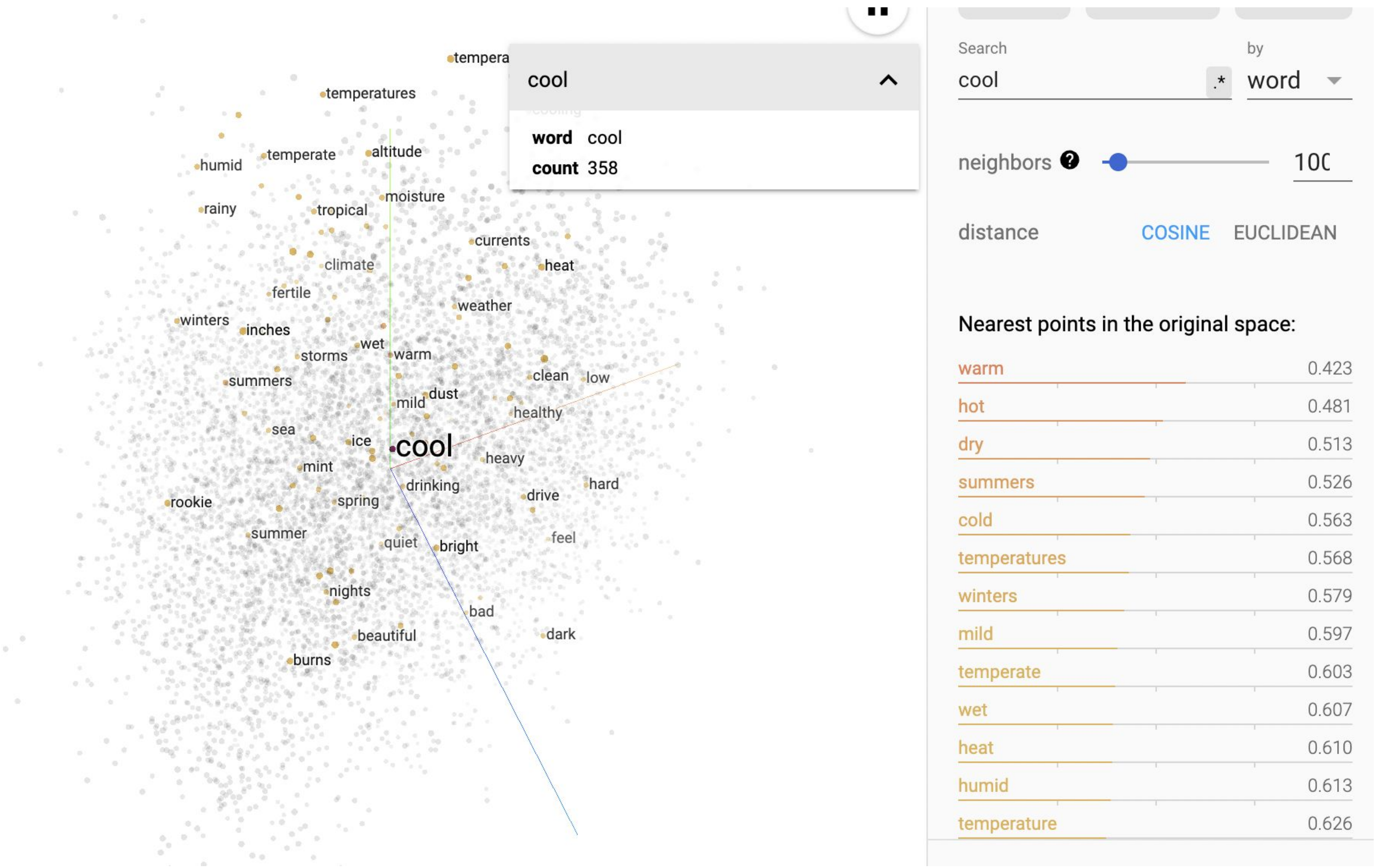
Word Vectors



king - man + woman = queen

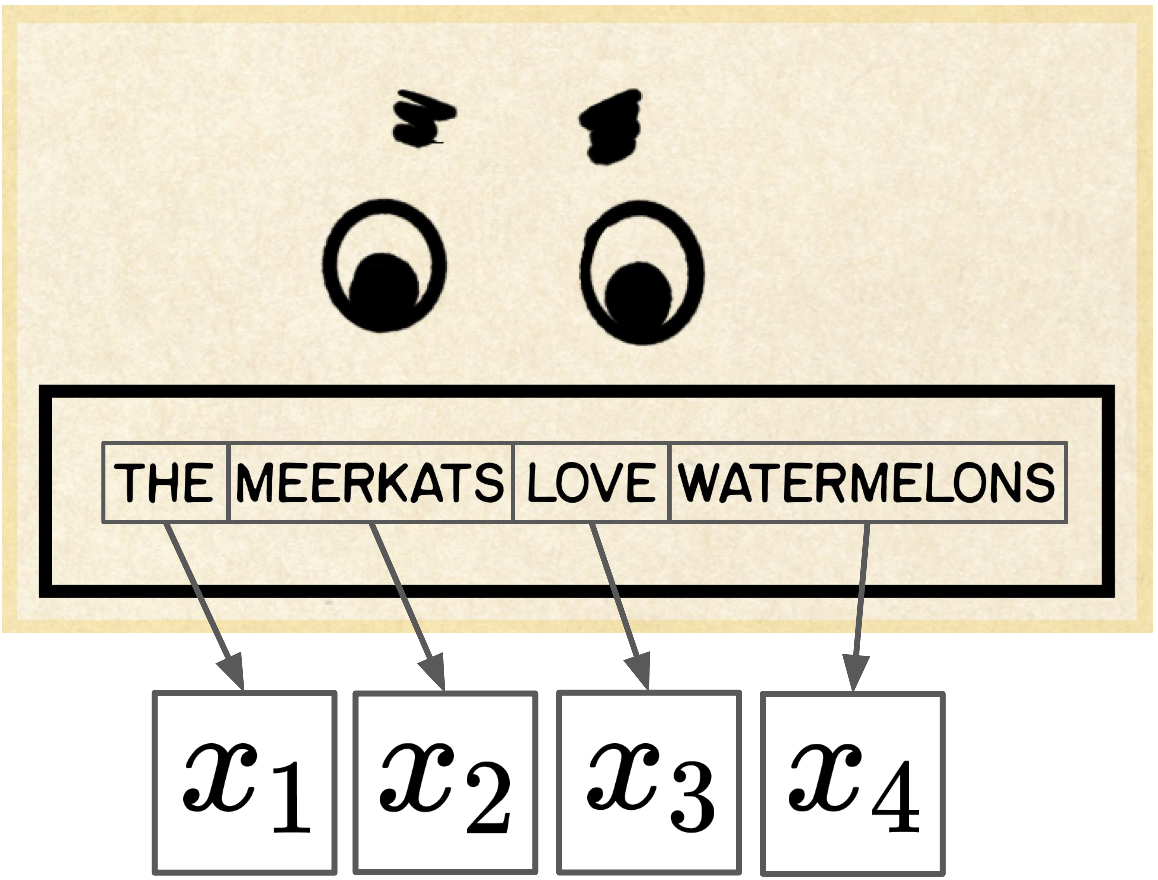
https://blog.acolyer.org/2016/04/21/the-amazing-power-of-word-vectors/

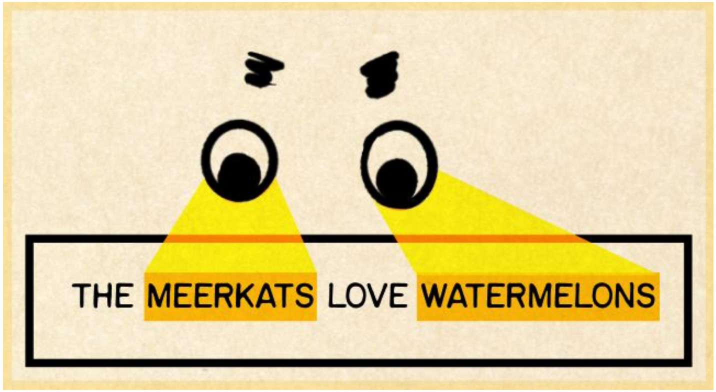
Text as a point in space



https://projector.tensorflow.org/

Paying Attention

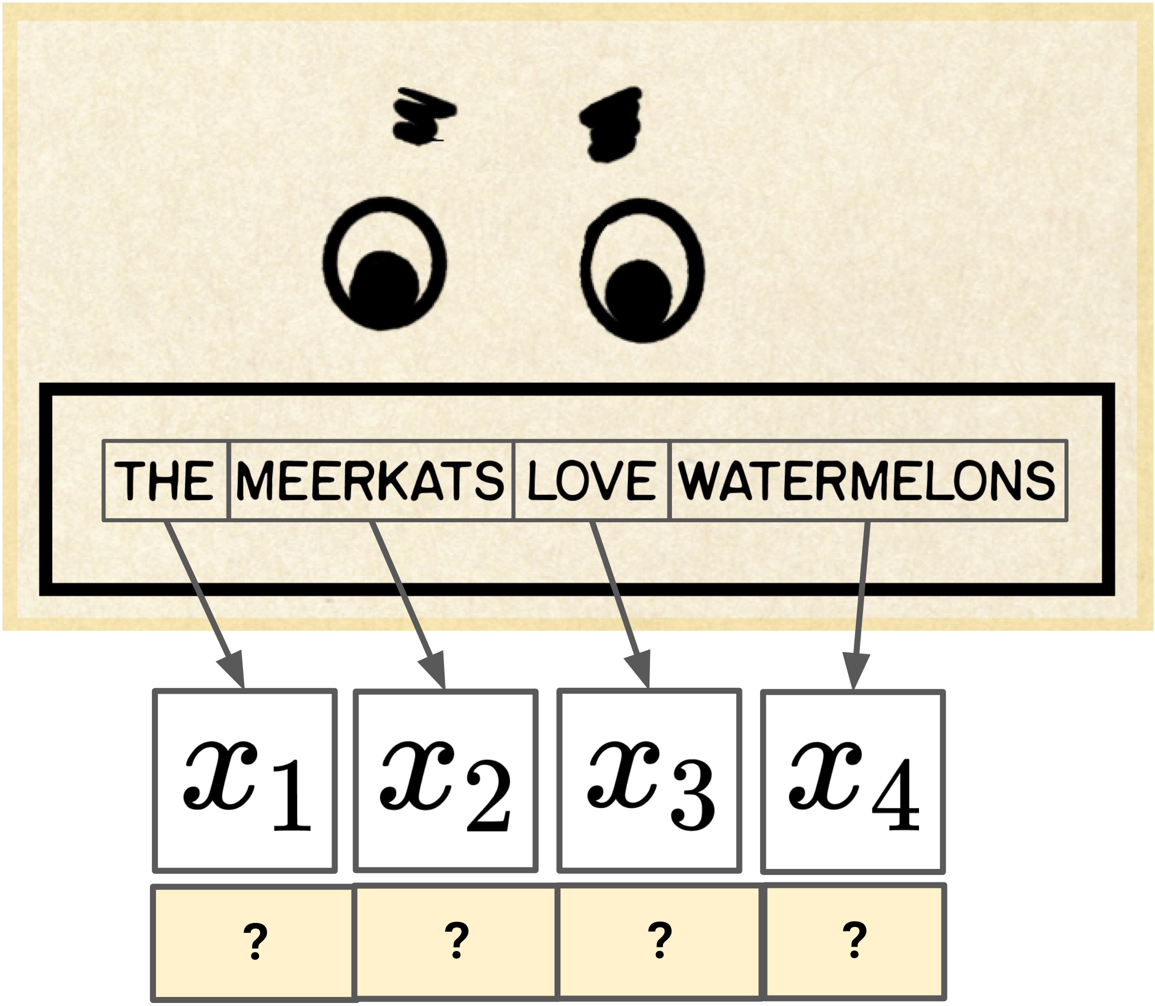


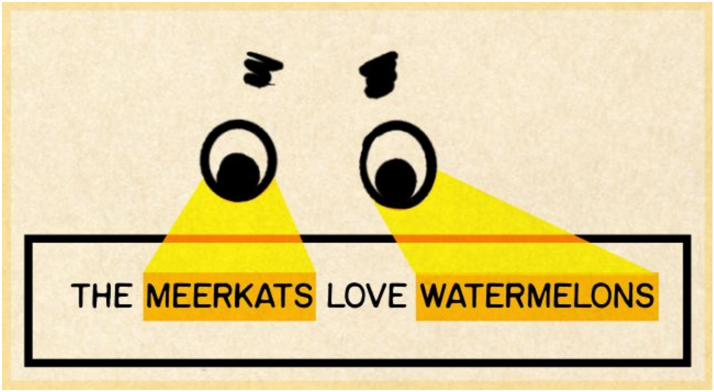


Our Goal

x_i Text Embedding Vector

Paying Attention

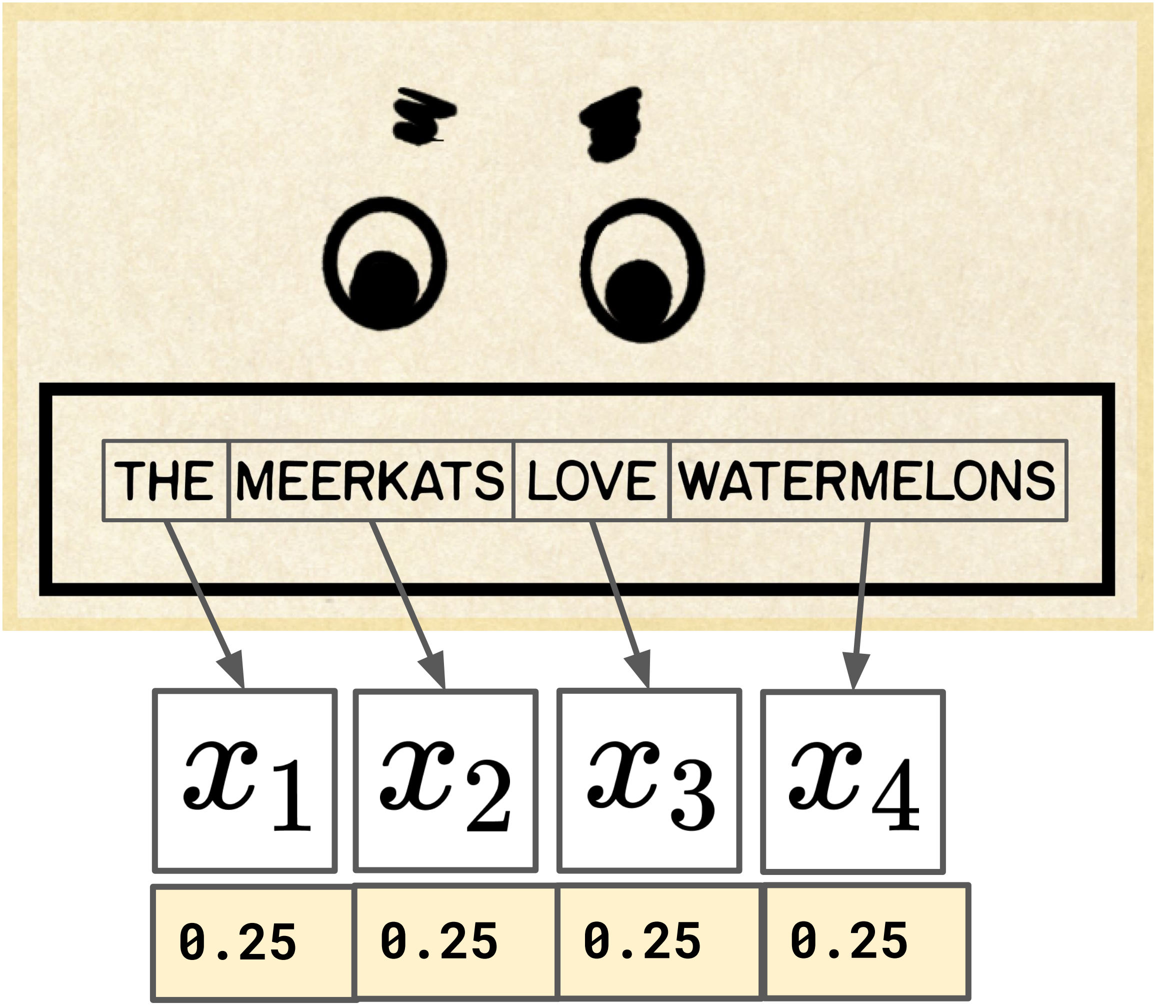




Our Goal

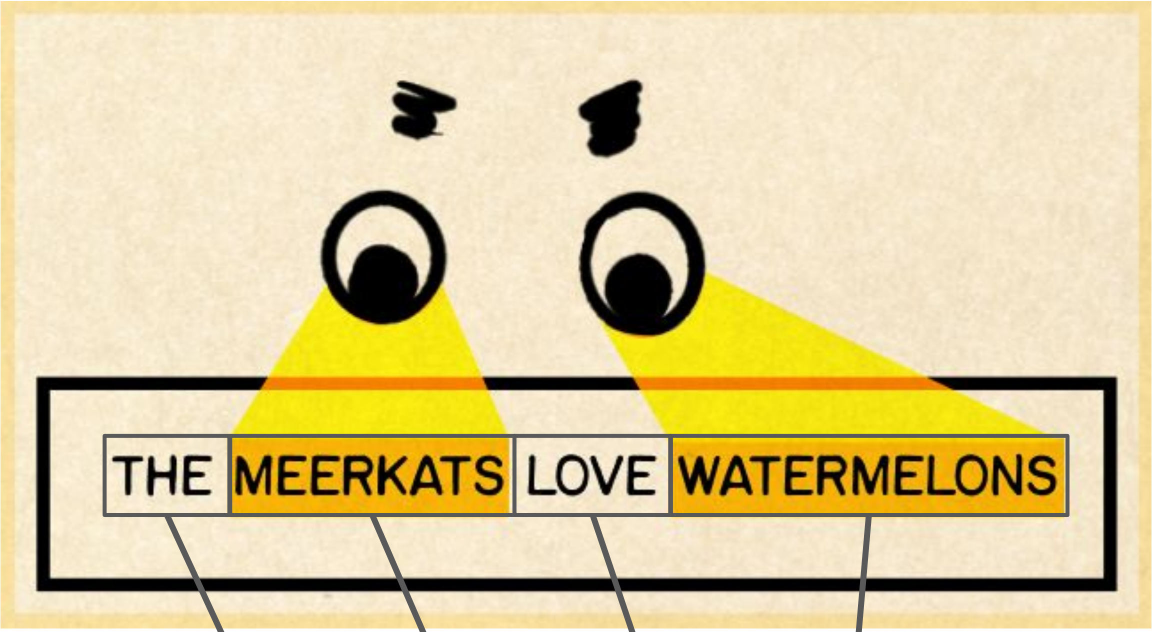
Attention Weights

Paying Attention



Uniform Attention

Paying Attention

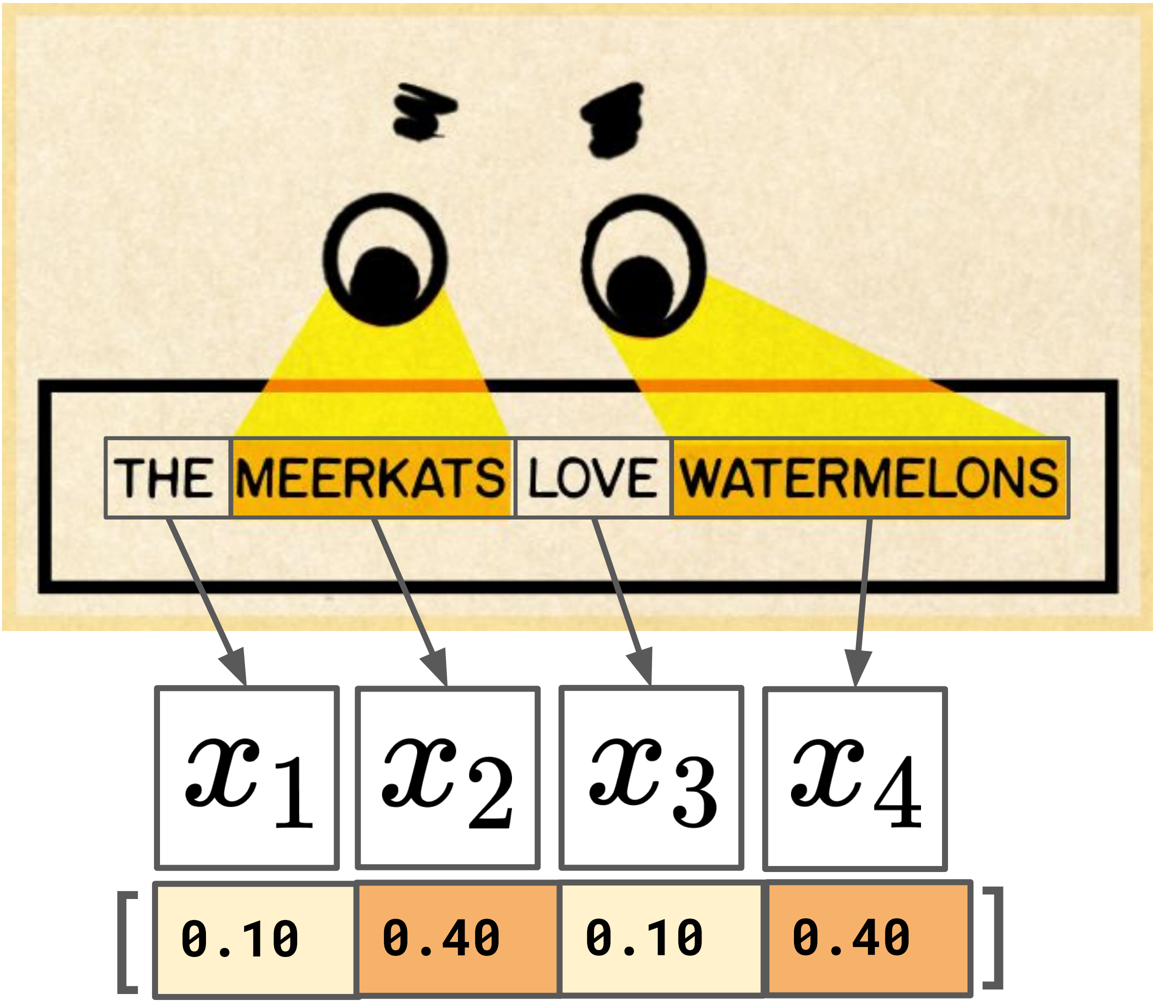


x_1 x_2 x_3 x_4

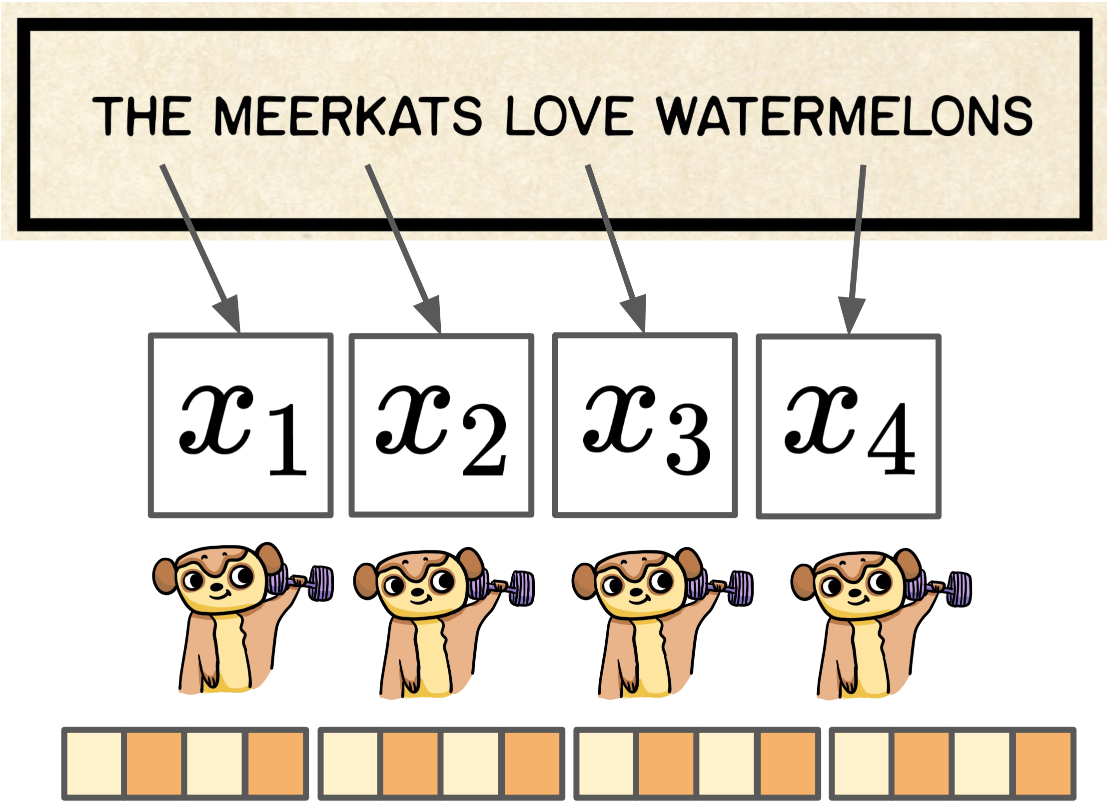
[0.10 0.40 0.10 0.40]

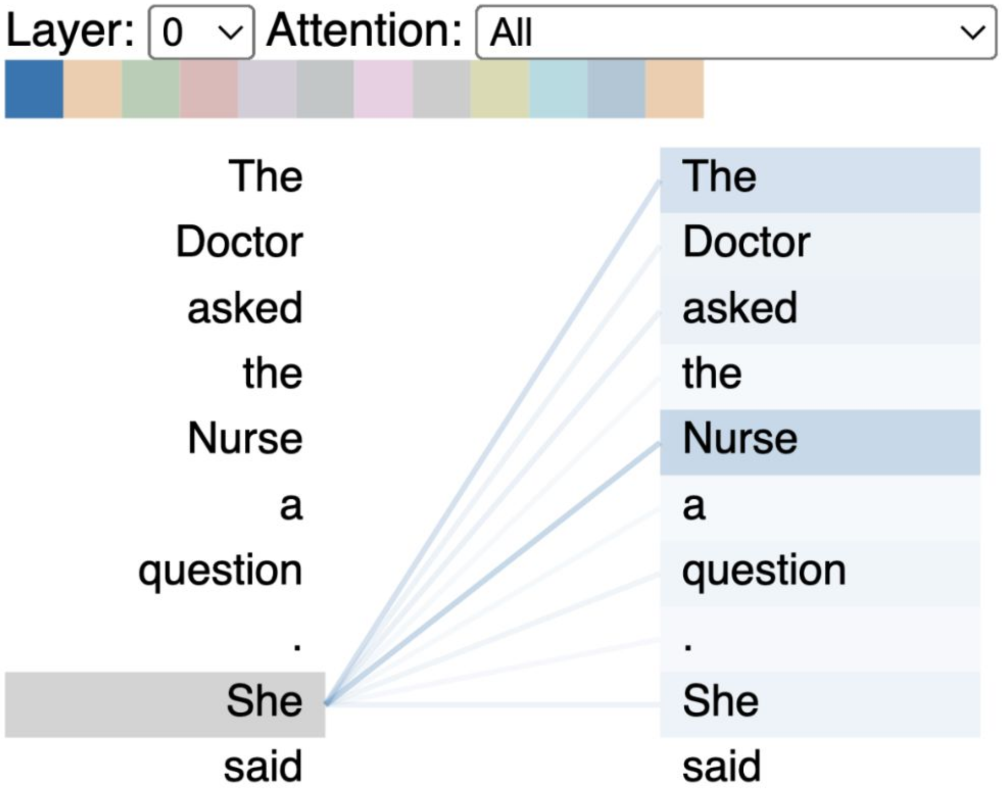
Scaled Attention

Paying Attention

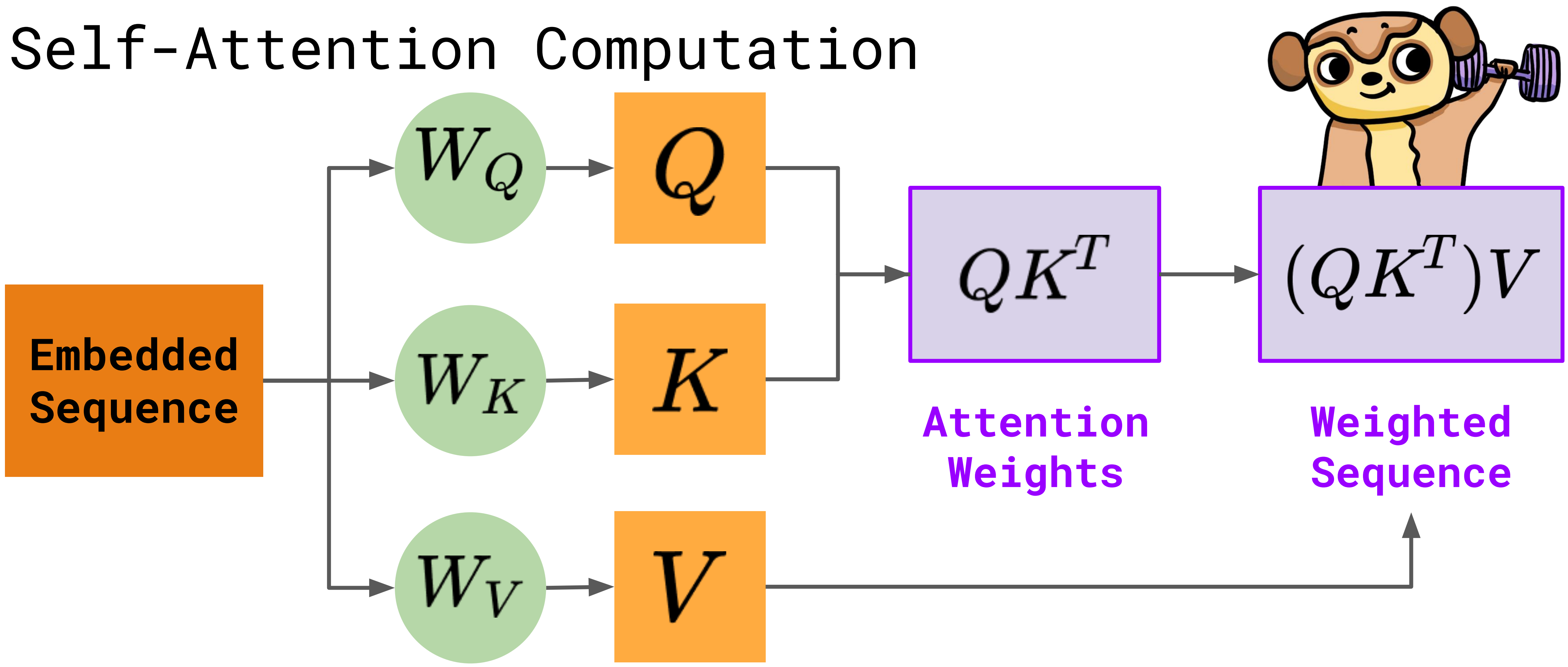


Paying Attention to Yourself - Self-attention



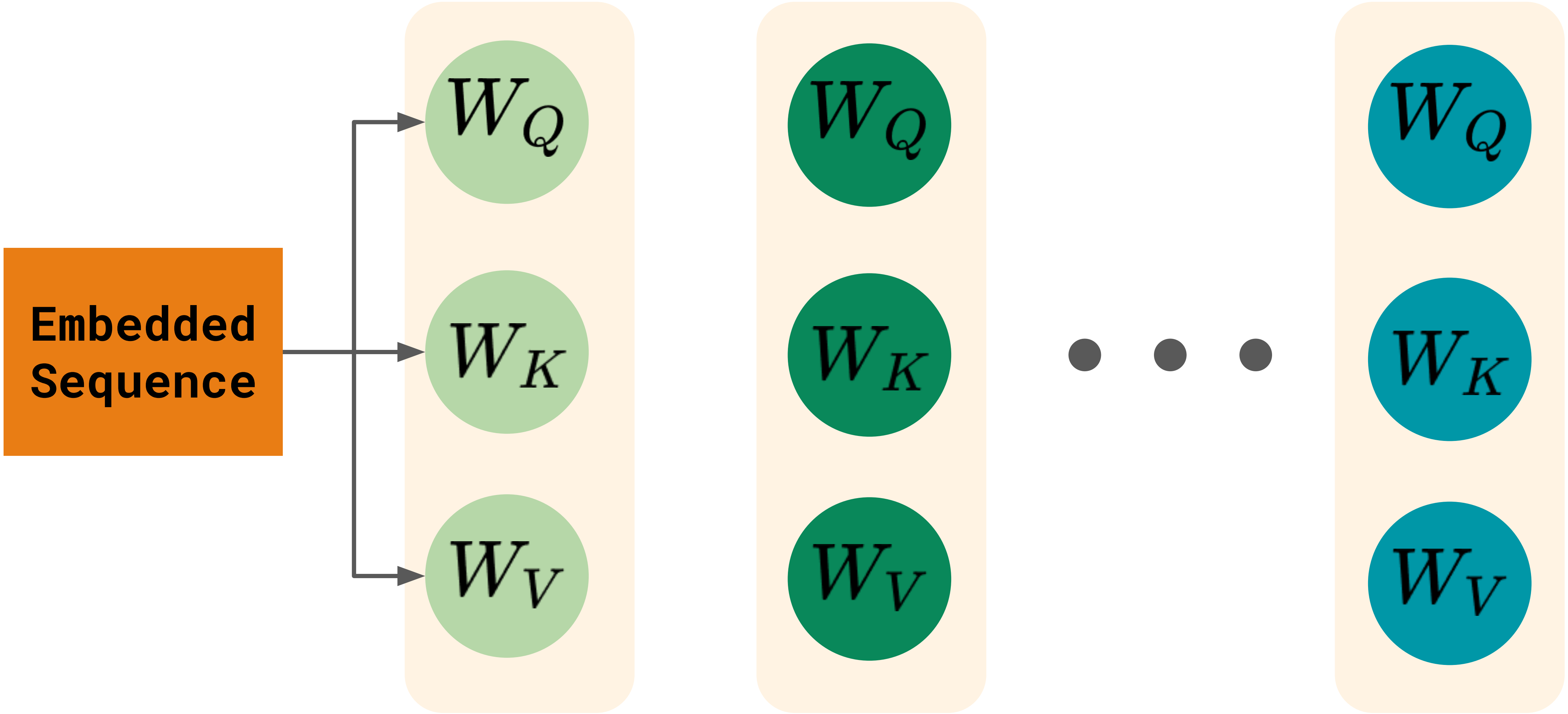


Self-Attention Computation




```
W_q = torch.nn.Parameter(torch.rand(...))
W_k = torch.nn.Parameter(torch.rand(...))
W_v = torch.nn.Parameter(torch.rand(...))
```

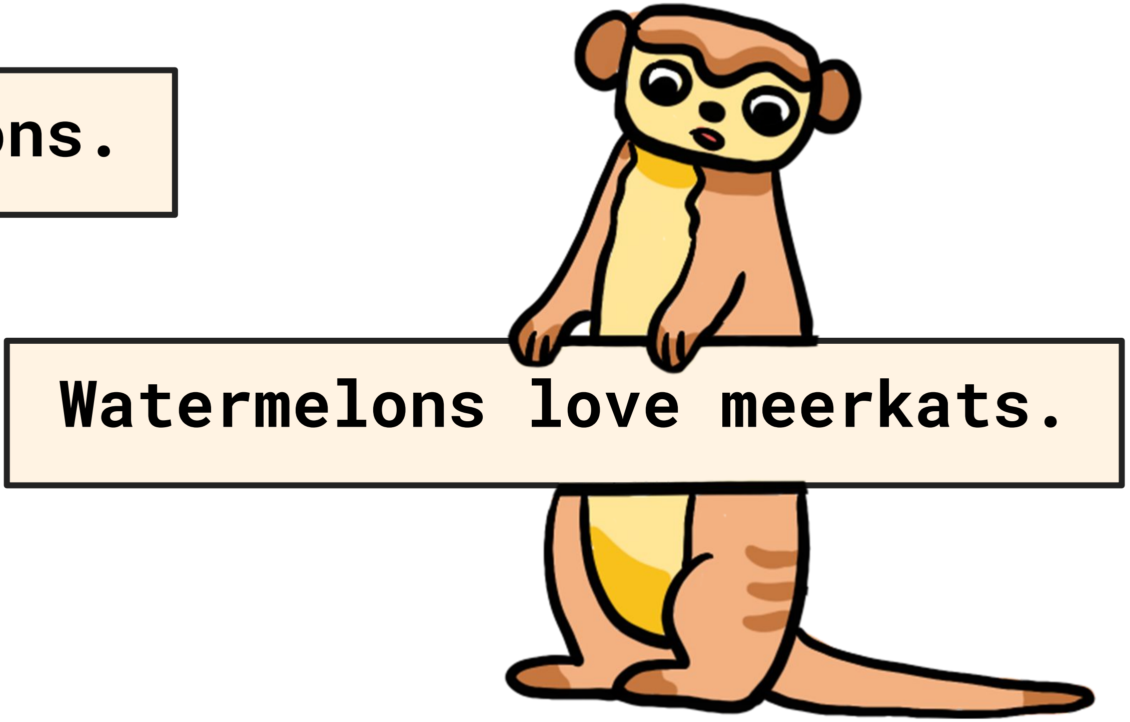
Multi-head Self-attention

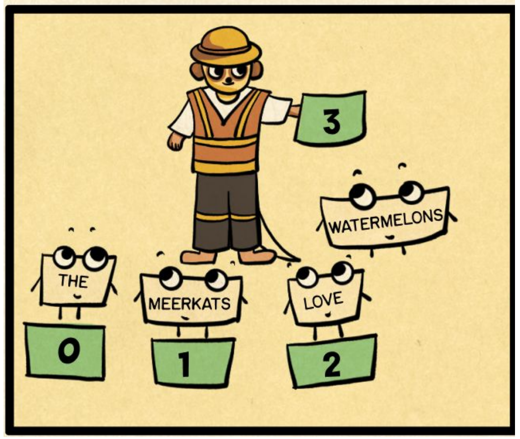


Takeaway

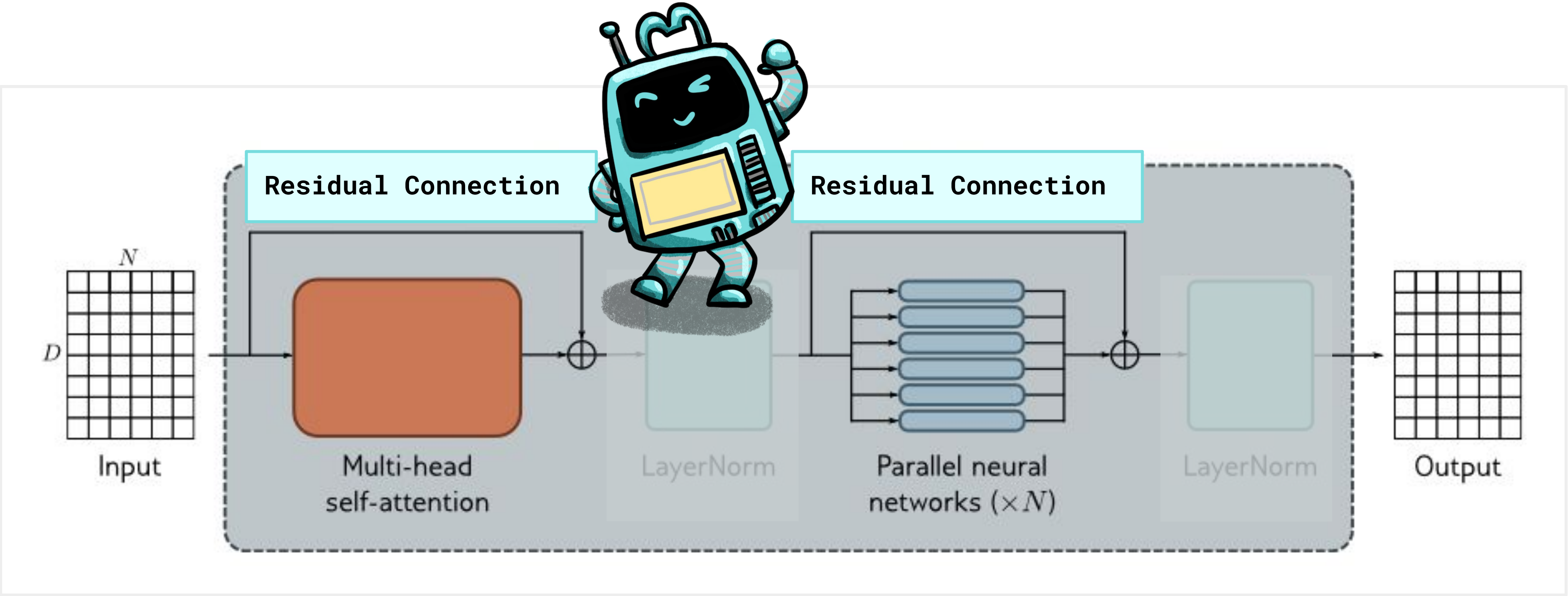
- Tokenization: text → tokens
- Embedding: n-dim word vector
- Self-attention computation

Positional Encoding

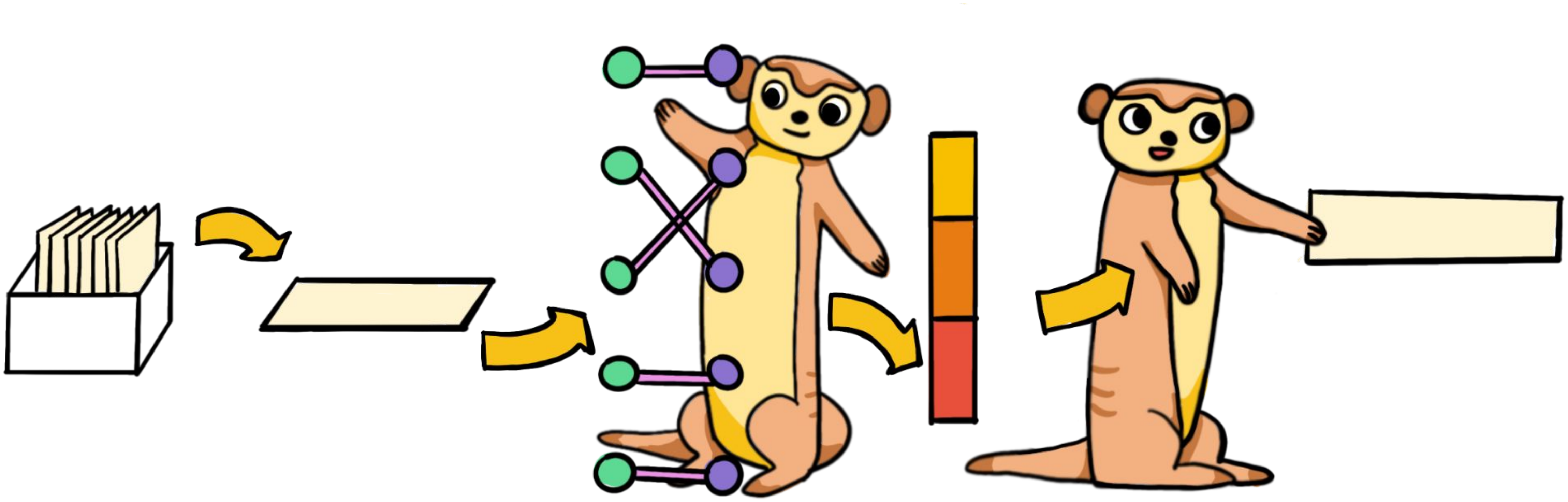




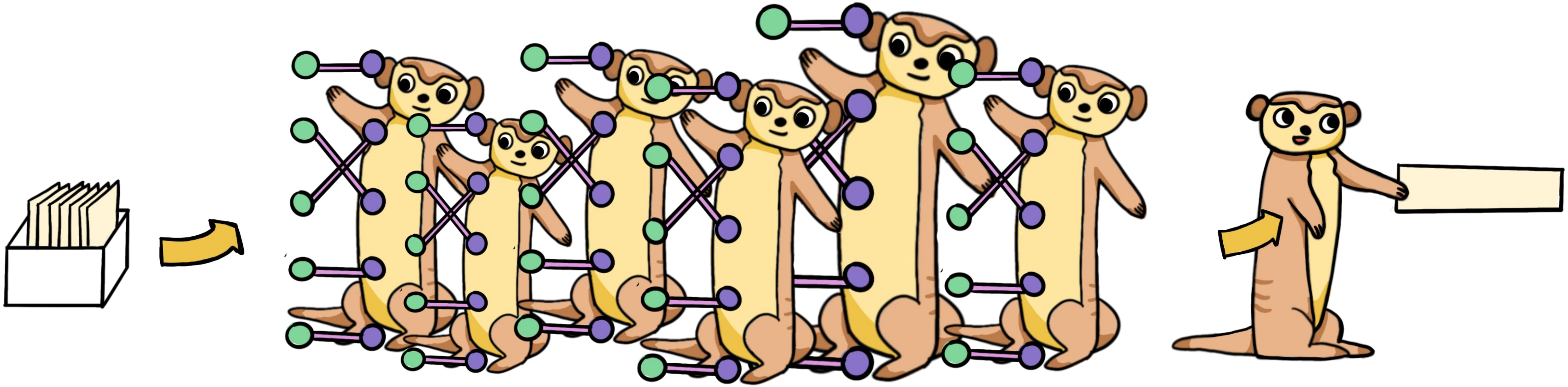
Transformer - Residual Connection



Simon J.D. Prince (2023). Understanding Deep Learning



Input → Some Layers → Output

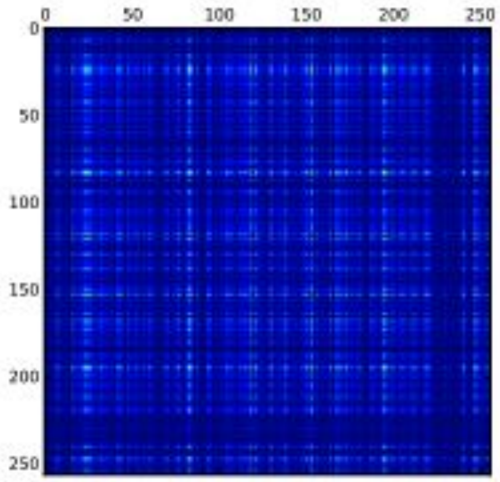


Input → A LOT MORE Layers → Output

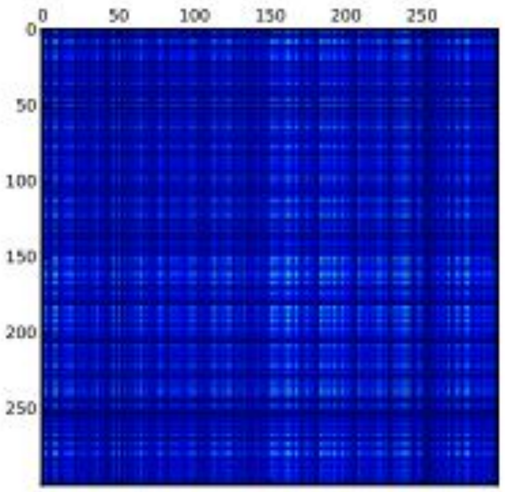
Shattered Gradient Problem

As depth increases, gradients in standard feedforward networks increasingly resemble white noise.

Shattered Gradient Problem



(b) 24-layer feedforward.



(e) White noise.

Balduzzi, D., Frean, M., Leary, L., et al. (2018). The shattered gradients problem

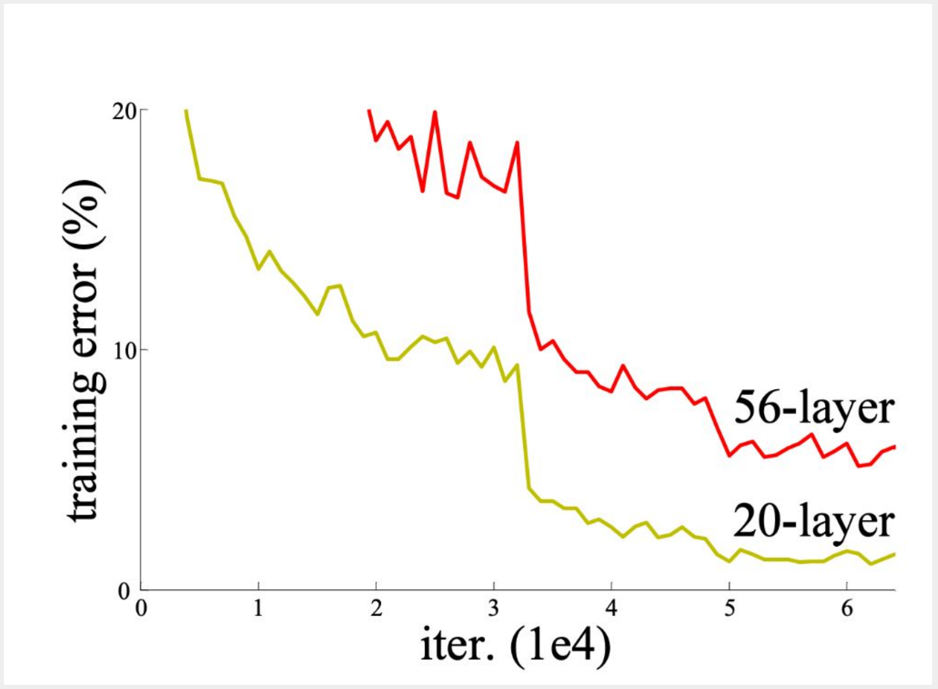

We are lost!

More layers ≠ better results

Deeper

Shallower

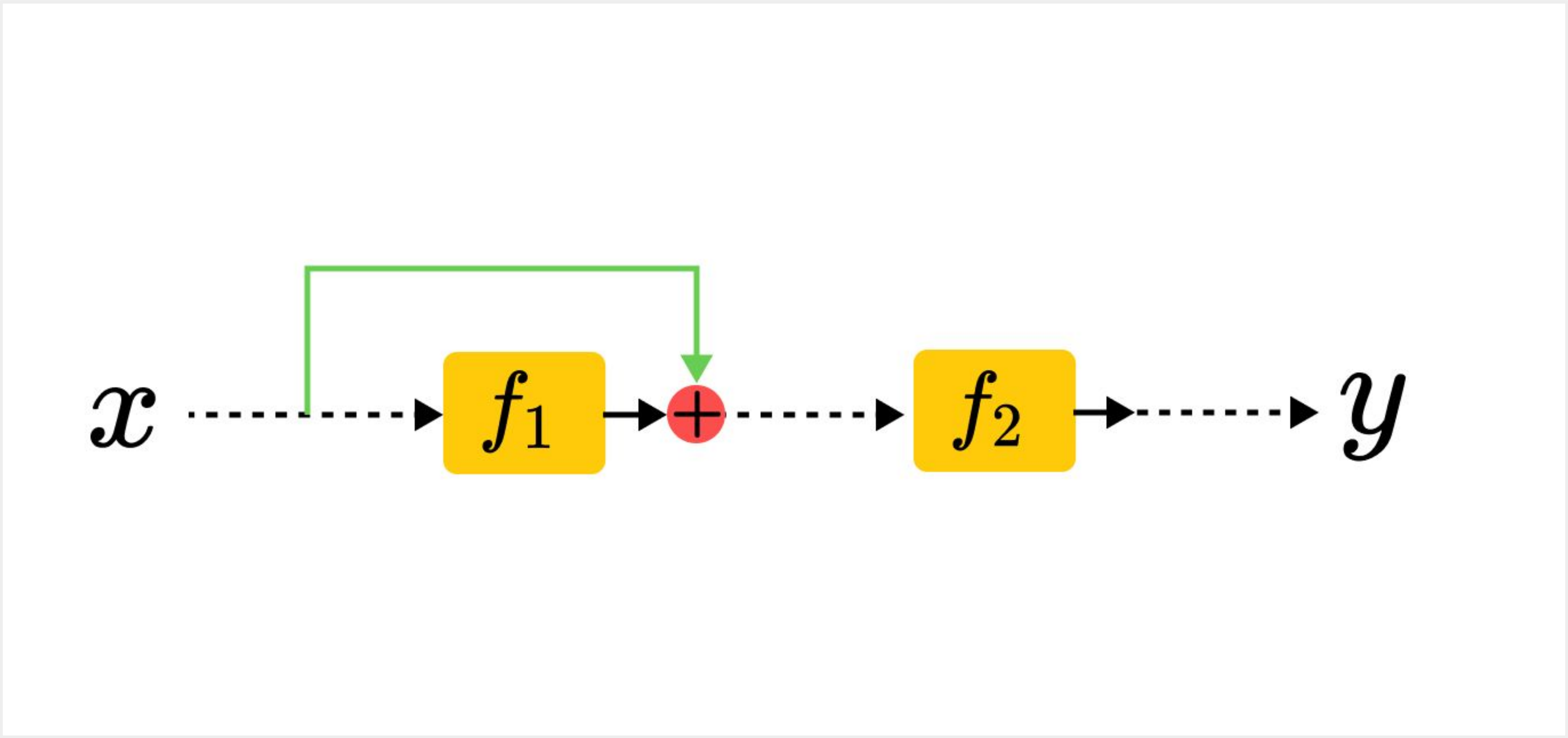
# of Errors



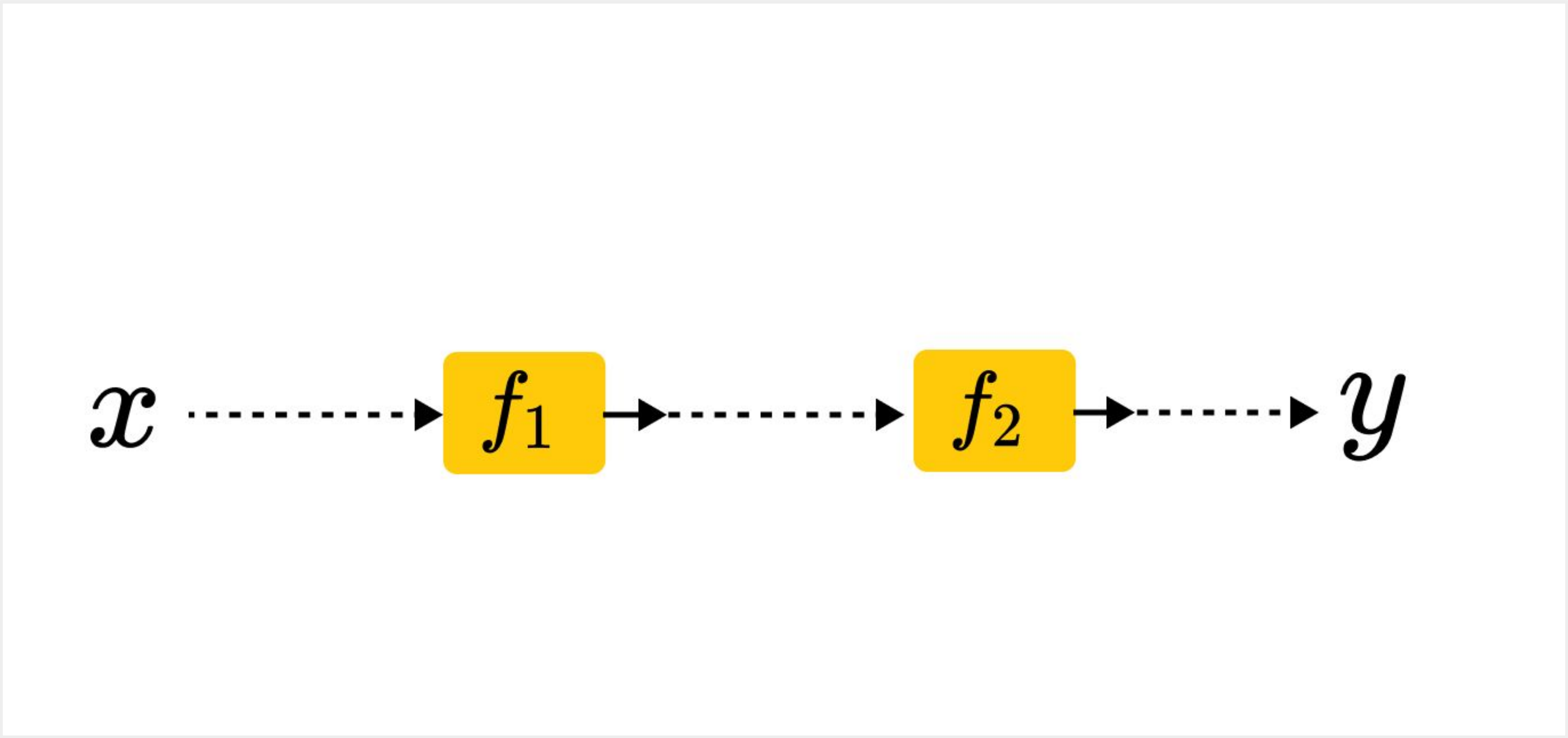
# of Iterations

He et al. (2015) Deep Residual Learning for Image Recognition

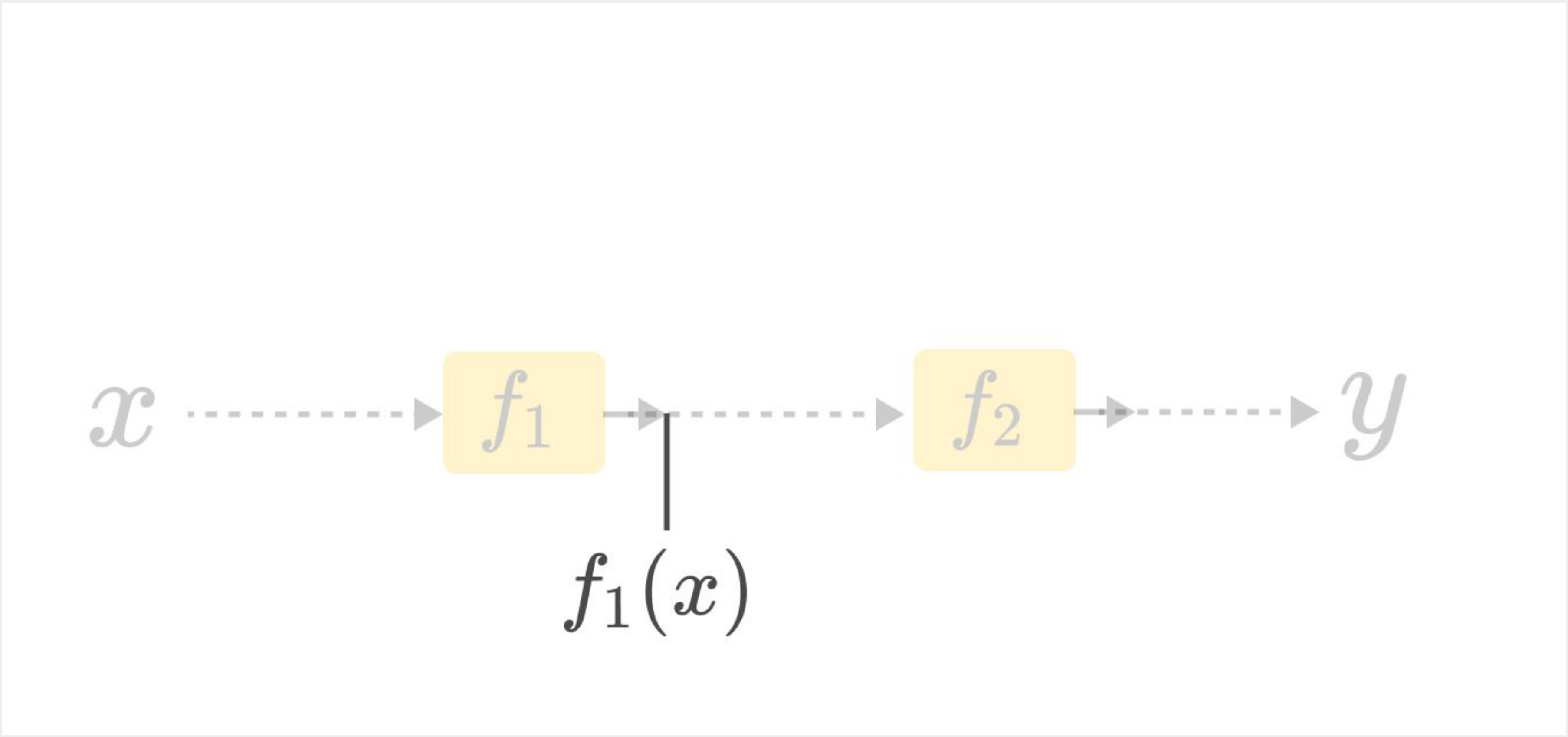
Residual Connection



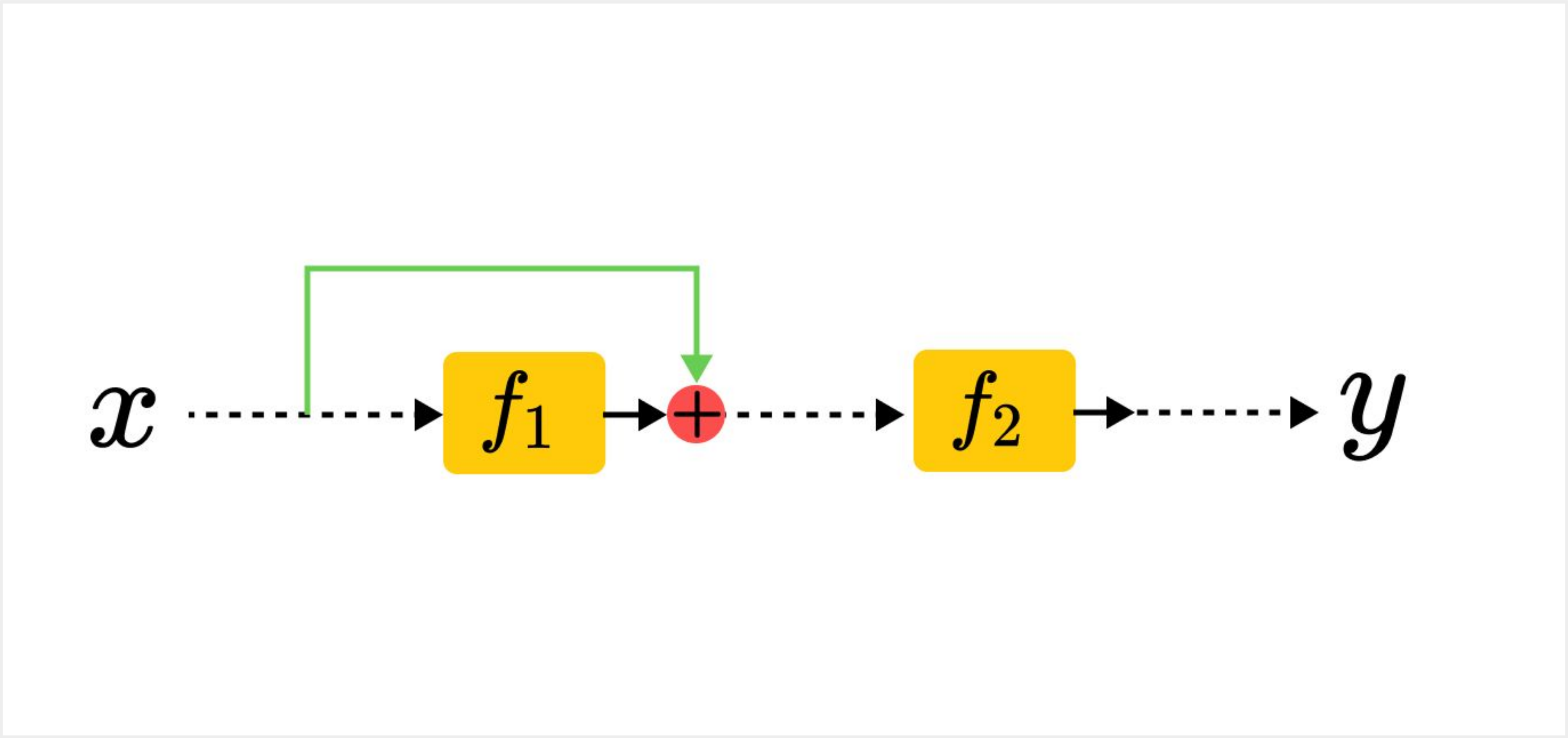
Sequential Connection



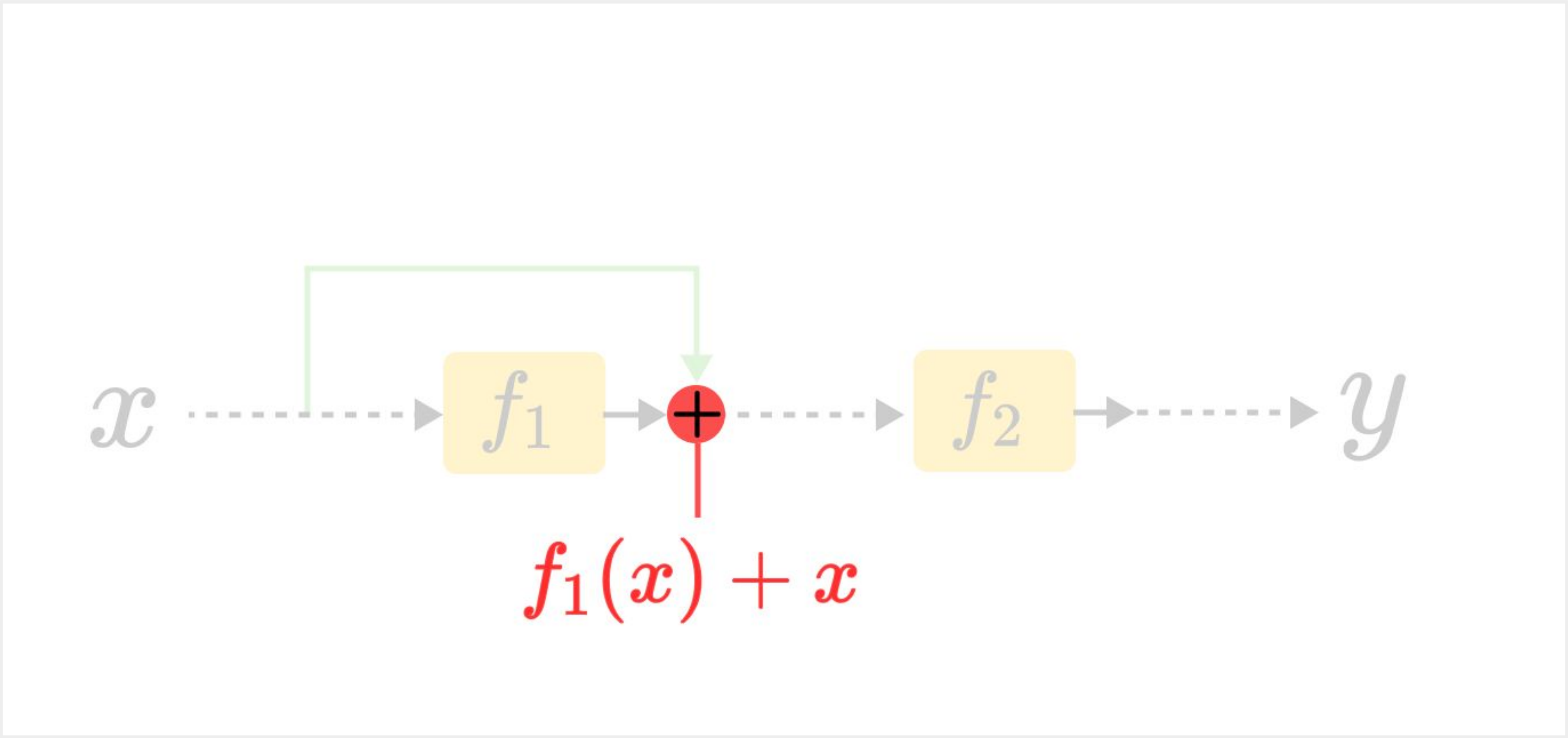
Sequential Connection



Residual Connection



Residual Connection



Residual Connections

$$\begin{aligned}
\mathbf{h}_1 &= \mathbf{x} + \mathbf{f}_1[\mathbf{x}, \phi_1] \\
\mathbf{h}_2 &= \mathbf{h}_1 + \mathbf{f}_2[\mathbf{h}_1, \phi_2] \\
\mathbf{h}_3 &= \mathbf{h}_2 + \mathbf{f}_3[\mathbf{h}_2, \phi_3] \\
\mathbf{y} &= \mathbf{h}_3 + \mathbf{f}_4[\mathbf{h}_3, \phi_4]
\end{aligned}$$

ϕ_k = weights & biases

Simon J.D. Prince (2023). Understanding Deep Learning

Residual Connections

$$\mathbf{h}_1 = \mathbf{x} + \mathbf{f}_1[\mathbf{x}, \boldsymbol{\phi}_1]$$

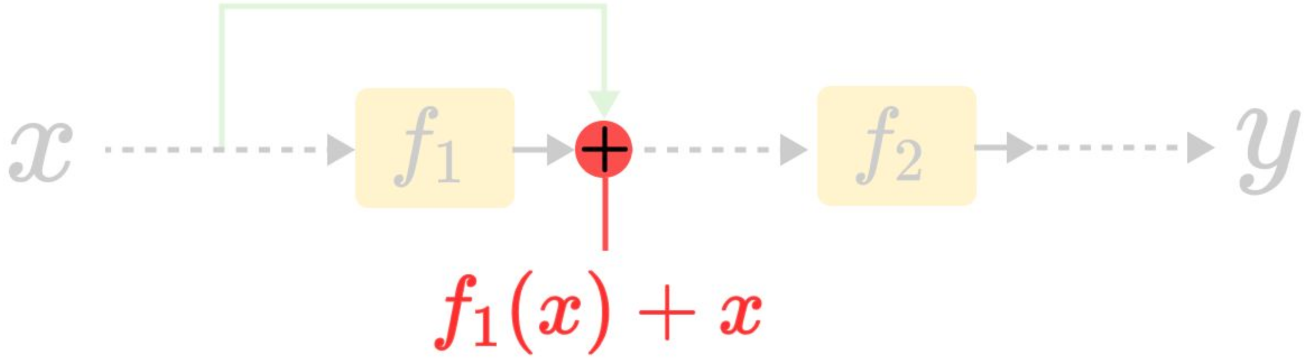
ϕ_k = weights & biases

Simon J.D. Prince (2023). Understanding Deep Learning

Residual Connections

Intermediate Input Transf. Weights

$$\mathbf{h}_1 = \mathbf{x} + \mathbf{f}_1[\mathbf{x}, \phi_1]$$



ϕ_k = weights & biases

Simon J.D. Prince (2023). Understanding Deep Learning

Residual Connections

$$
\begin{aligned}
\mathbf{h}_1 &= \mathbf{x} + \mathbf{f}_1[\mathbf{x}, \phi_1] \\
\mathbf{h}_2 &= \mathbf{h}_1 + \mathbf{f}_2[\mathbf{h}_1, \phi_2] \\
\mathbf{h}_3 &= \mathbf{h}_2 + \mathbf{f}_3[\mathbf{h}_2, \phi_3] \\
\mathbf{y} &= \mathbf{h}_3 + \mathbf{f}_4[\mathbf{h}_3, \phi_4]
\end{aligned}
$$

ϕ_k = weights & biases

Simon J.D. Prince (2023). Understanding Deep Learning

Residual Connections

$$
\begin{aligned}
\mathbf{h}_1 &= \mathbf{x} + \mathbf{f}_1[\mathbf{x}, \phi_1] \\
\mathbf{h}_2 &= \mathbf{h}_1 + \mathbf{f}_2[\mathbf{h}_1, \phi_2] \\
\boxed{\mathbf{h}_3} &= \mathbf{h}_2 + \mathbf{f}_3[\mathbf{h}_2, \phi_3] \\
\mathbf{y} &= \boxed{\mathbf{h}_3} + \mathbf{f}_4[\boxed{\mathbf{h}_3,} \phi_4]
\end{aligned}
$$

ϕ_k = weights & biases

Simon J.D. Prince (2023). Understanding Deep Learning

Residual Connections

$$
\begin{aligned}
\mathbf{h}_1 &= \mathbf{x} + \mathbf{f}_1[\mathbf{x}, \phi_1] \\
\mathbf{h}_2 &= \mathbf{h}_1 + \mathbf{f}_2[\mathbf{h}_1, \phi_2] \\
\mathbf{h}_3 &= \mathbf{h}_2 + \mathbf{f}_3[\mathbf{h}_2, \phi_3] \\
\mathbf{y} &= \mathbf{h}_3 + \mathbf{f}_4[\mathbf{h}_3, \phi_4]
\end{aligned}
$$

ϕ_k = weights & biases

Simon J.D. Prince (2023). Understanding Deep Learning

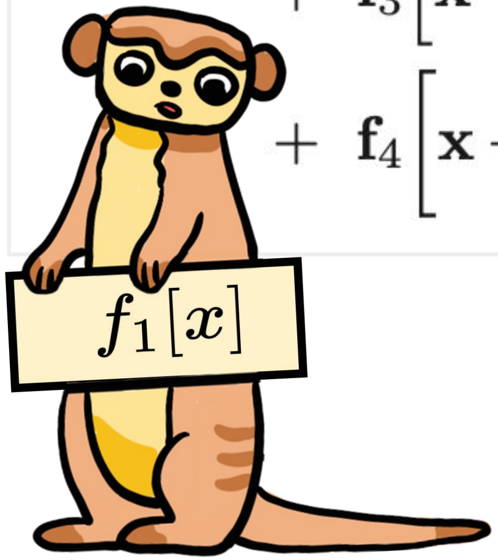
Residual Connections

$$
\begin{aligned}
\mathbf{y} = \mathbf{x} \; + \;& \mathbf{f}_1[\mathbf{x}] \\
+ \;& \mathbf{f}_2\big[\mathbf{x} + \mathbf{f}_1[\mathbf{x}]\big] \\
+ \;& \mathbf{f}_3\Big[\mathbf{x} + \mathbf{f}_1[\mathbf{x}] + \mathbf{f}_2\big[\mathbf{x} + \mathbf{f}_1[\mathbf{x}]\big]\Big] \\
+ \;& \mathbf{f}_4\bigg[\mathbf{x} + \mathbf{f}_1[\mathbf{x}] + \mathbf{f}_2\big[\mathbf{x} + \mathbf{f}_1[\mathbf{x}]\big] + \mathbf{f}_3\Big[\mathbf{x} + \mathbf{f}_1[\mathbf{x}] + \mathbf{f}_2\big[\mathbf{x} + \mathbf{f}_1[\mathbf{x}]\big]\Big]\bigg]
\end{aligned}
$$

Simon J.D. Prince (2023). Understanding Deep Learning

Residual Connections

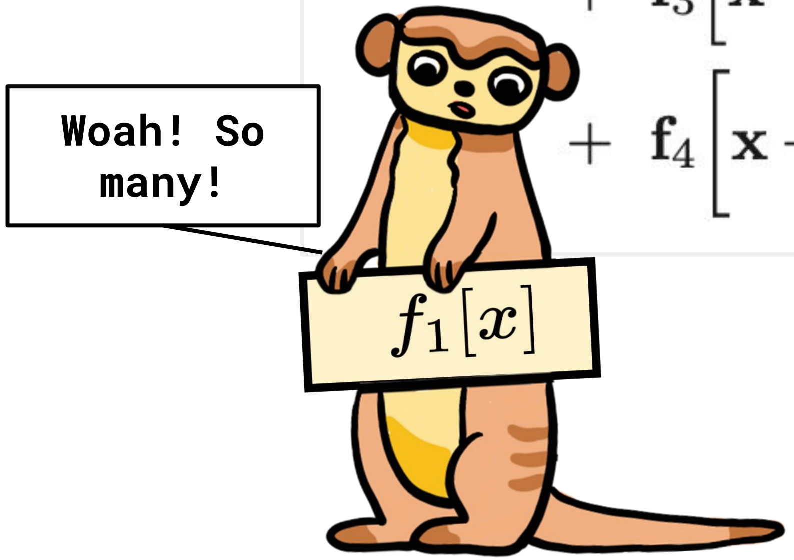
$$
\begin{aligned}
\mathbf{y} = \mathbf{x} \; + \; & \mathbf{f}_1[\mathbf{x}] \\
+ \; & \mathbf{f}_2\big[\mathbf{x} + \mathbf{f}_1[\mathbf{x}]\big] \\
+ \; & \mathbf{f}_3\Big[\mathbf{x} + \mathbf{f}_1[\mathbf{x}] + \mathbf{f}_2\big[\mathbf{x} + \mathbf{f}_1[\mathbf{x}]\big]\Big] \\
+ \; & \mathbf{f}_4\Big[\mathbf{x} + \mathbf{f}_1[\mathbf{x}] + \mathbf{f}_2\big[\mathbf{x} + \mathbf{f}_1[\mathbf{x}]\big] + \mathbf{f}_3\Big[\mathbf{x} + \mathbf{f}_1[\mathbf{x}] + \mathbf{f}_2\big[\mathbf{x} + \mathbf{f}_1[\mathbf{x}]\big]\Big]\Big]
\end{aligned}
$$



Simon J.D. Prince (2023). Understanding Deep Learning

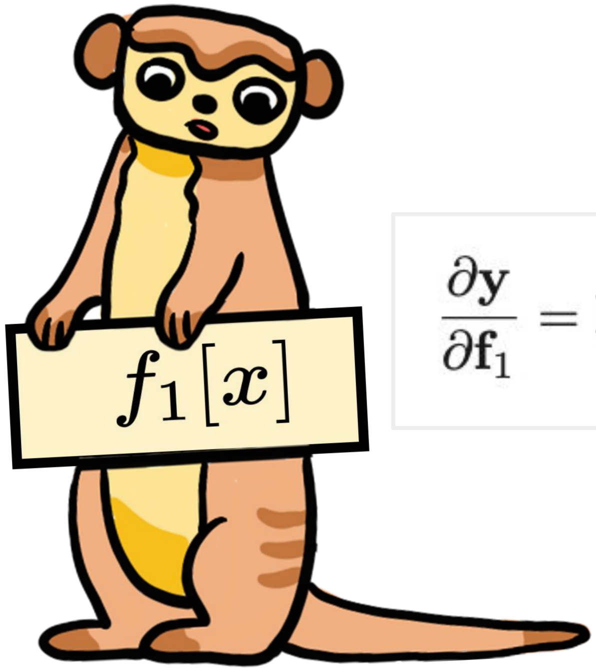
Residual Connections

$$
\begin{aligned}
\mathbf{y} = \mathbf{x} &+ \boxed{\mathbf{f}_1[\mathbf{x}]} \\
&+ \mathbf{f}_2\left[\mathbf{x} + \boxed{\mathbf{f}_1[\mathbf{x}]}\right] \\
&+ \mathbf{f}_3\left[\mathbf{x} + \boxed{\mathbf{f}_1[\mathbf{x}]} + \mathbf{f}_2\left[\mathbf{x} + \boxed{\mathbf{f}_1[\mathbf{x}]}\right]\right] \\
&+ \mathbf{f}_4\left[\mathbf{x} + \boxed{\mathbf{f}_1[\mathbf{x}]} + \mathbf{f}_2\left[\mathbf{x} + \boxed{\mathbf{f}_1[\mathbf{x}]}\right] + \mathbf{f}_3\left[\mathbf{x} + \boxed{\mathbf{f}_1[\mathbf{x}]} + \mathbf{f}_2\left[\mathbf{x} + \boxed{\mathbf{f}_1[\mathbf{x}]}\right]\right]\right]
\end{aligned}
$$



Simon J.D. Prince (2023). Understanding Deep Learning

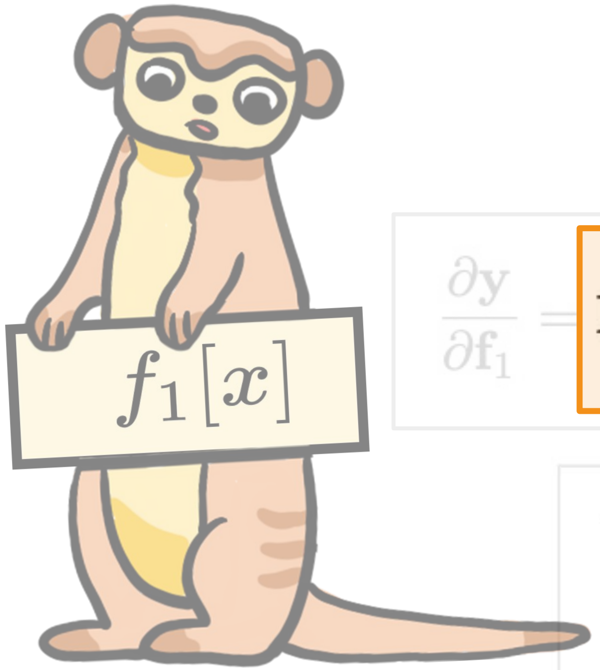
Residual Connections

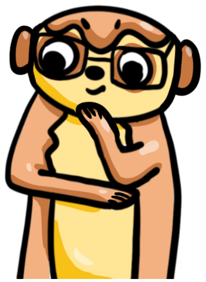


$$\frac{\partial \mathbf{y}}{\partial \mathbf{f}_1} = \mathbf{I} + \frac{\partial \mathbf{f}_2}{\partial \mathbf{f}_1} + \left(\frac{\partial \mathbf{f}_3}{\partial \mathbf{f}_1} + \frac{\partial \mathbf{f}_3}{\partial \mathbf{f}_2}\frac{\partial \mathbf{f}_2}{\partial \mathbf{f}_1} \right) + \left(\frac{\partial \mathbf{f}_4}{\partial \mathbf{f}_1} + \frac{\partial \mathbf{f}_4}{\partial \mathbf{f}_2}\frac{\partial \mathbf{f}_2}{\partial \mathbf{f}_1} + \frac{\partial \mathbf{f}_4}{\partial \mathbf{f}_3}\frac{\partial \mathbf{f}_3}{\partial \mathbf{f}_1} + \frac{\partial \mathbf{f}_4}{\partial \mathbf{f}_3}\frac{\partial \mathbf{f}_3}{\partial \mathbf{f}_2}\frac{\partial \mathbf{f}_2}{\partial \mathbf{f}_1} \right)$$

Simon J.D. Prince (2023). Understanding Deep Learning

Residual Connections





$$\frac{\partial \mathbf{y}}{\partial \mathbf{f}_1} = \boxed{\mathbf{I} + \frac{\partial \mathbf{f}_2}{\partial \mathbf{f}_1} + \left(\frac{\partial \mathbf{f}_3}{\partial \mathbf{f}_1} + \frac{\partial \mathbf{f}_3}{\partial \mathbf{f}_2}\frac{\partial \mathbf{f}_2}{\partial \mathbf{f}_1}\right) + \left(\frac{\partial \mathbf{f}_4}{\partial \mathbf{f}_1} + \frac{\partial \mathbf{f}_4}{\partial \mathbf{f}_2}\frac{\partial \mathbf{f}_2}{\partial \mathbf{f}_1} + \frac{\partial \mathbf{f}_4}{\partial \mathbf{f}_3}\frac{\partial \mathbf{f}_3}{\partial \mathbf{f}_1} + \frac{\partial \mathbf{f}_4}{\partial \mathbf{f}_3}\frac{\partial \mathbf{f}_3}{\partial \mathbf{f}_2}\frac{\partial \mathbf{f}_2}{\partial \mathbf{f}_1}\right)}$$

$$\begin{aligned}
\mathbf{y} = \mathbf{x} &+ \boxed{\mathbf{f}_1[\mathbf{x}]} \\
&+ \mathbf{f}_2\big[\mathbf{x} + \boxed{\mathbf{f}_1[\mathbf{x}]}\big] \\
&+ \mathbf{f}_3\Big[\mathbf{x} + \boxed{\mathbf{f}_1[\mathbf{x}]} + \mathbf{f}_2\big[\mathbf{x} + \boxed{\mathbf{f}_1[\mathbf{x}]}\big]\Big] \\
&+ \mathbf{f}_4\Big[\mathbf{x} + \boxed{\mathbf{f}_1[\mathbf{x}]} + \mathbf{f}_2\big[\mathbf{x} + \boxed{\mathbf{f}_1[\mathbf{x}]}\big] + \mathbf{f}_3\big[\mathbf{x} + \boxed{\mathbf{f}_1[\mathbf{x}]} + \mathbf{f}_2\big[\mathbf{x} + \boxed{\mathbf{f}_1[\mathbf{x}]}\big]\big]\Big]
\end{aligned}$$

Simon J.D. Prince (2023). Understanding Deep Learning


Thank you!
Now we can
find our way.

Takeaway

Residual connections suffer less from gradient problems.

What else can we do for bad gradients?

Batch Normalization

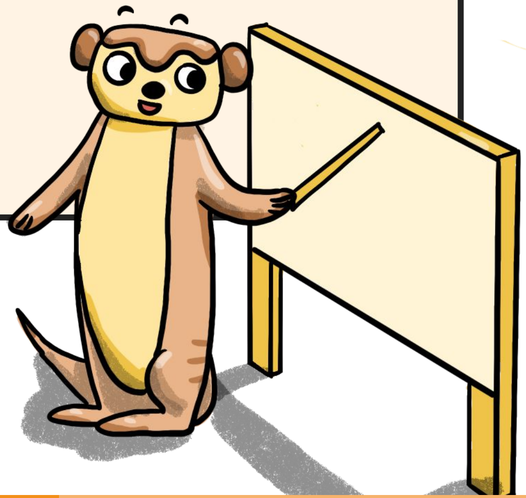
```
torch.nn.BatchNorm2d(num_features,…)
```

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

Batch Normalization

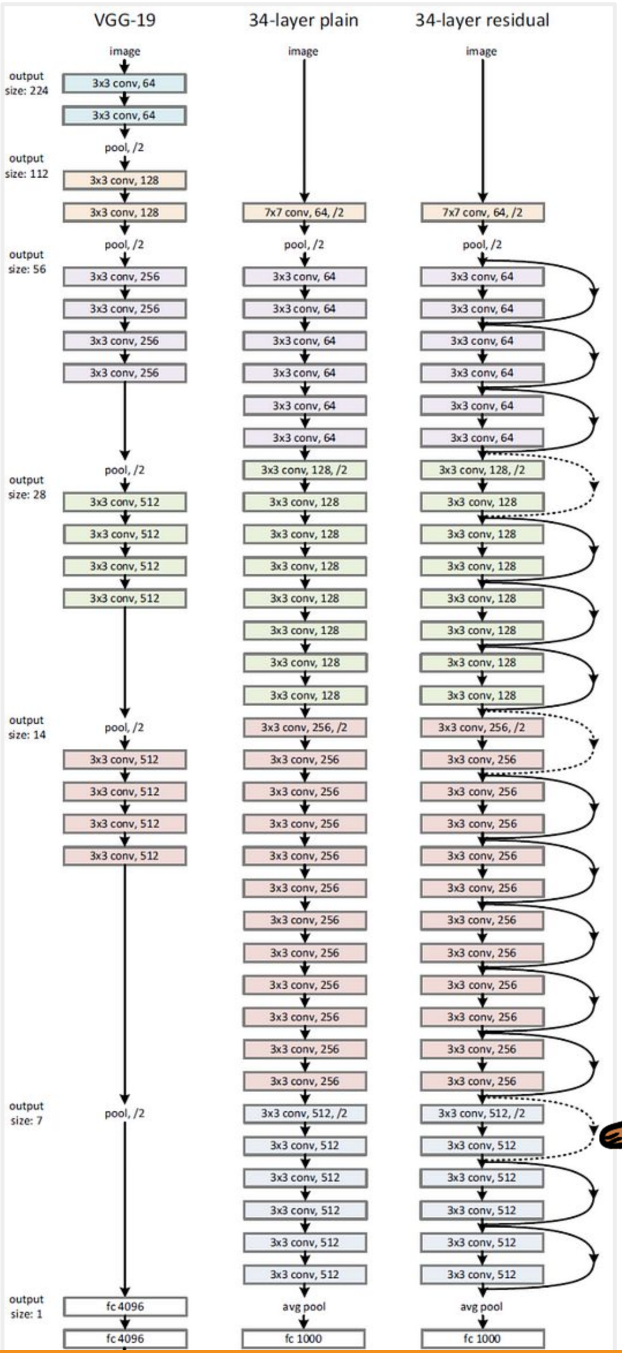
`torch.nn.BatchNorm2d(num_features,…)`

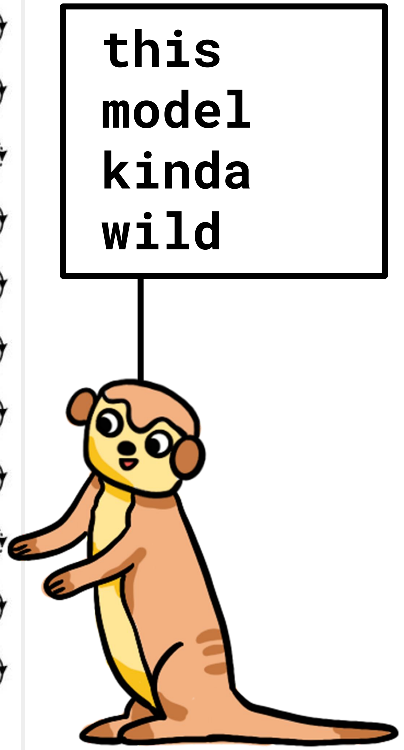
- Reduce shattered gradients
- Stable forward propagation
- Regularization



ResNet

- Batch Normalization
- Residual Connection



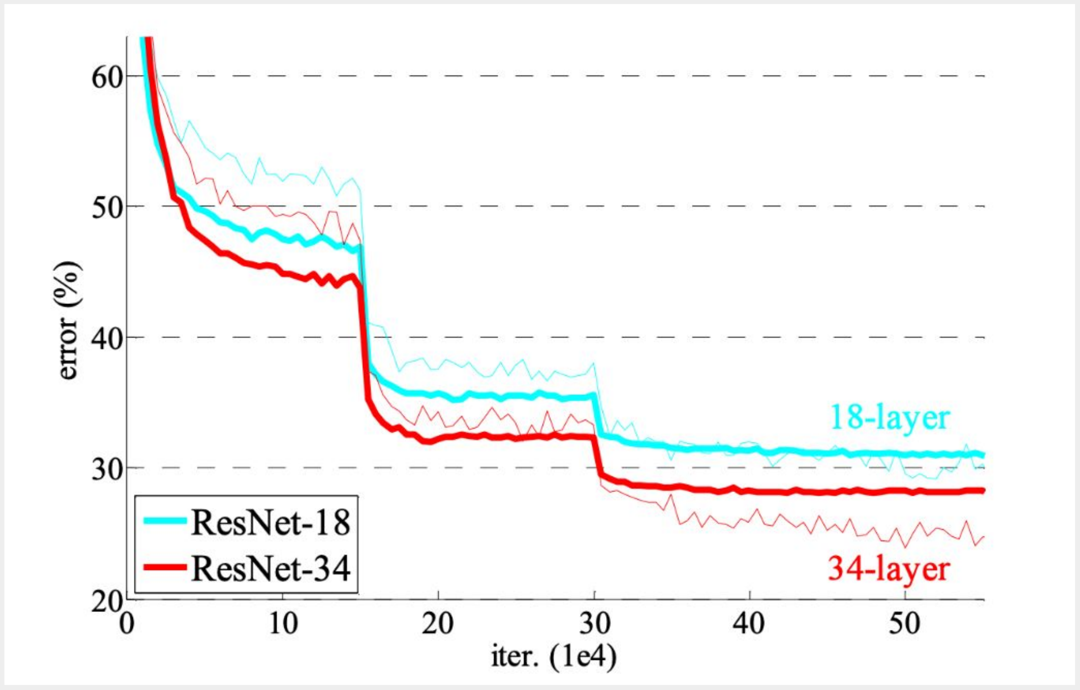


ResNet

Deeper

Shallower

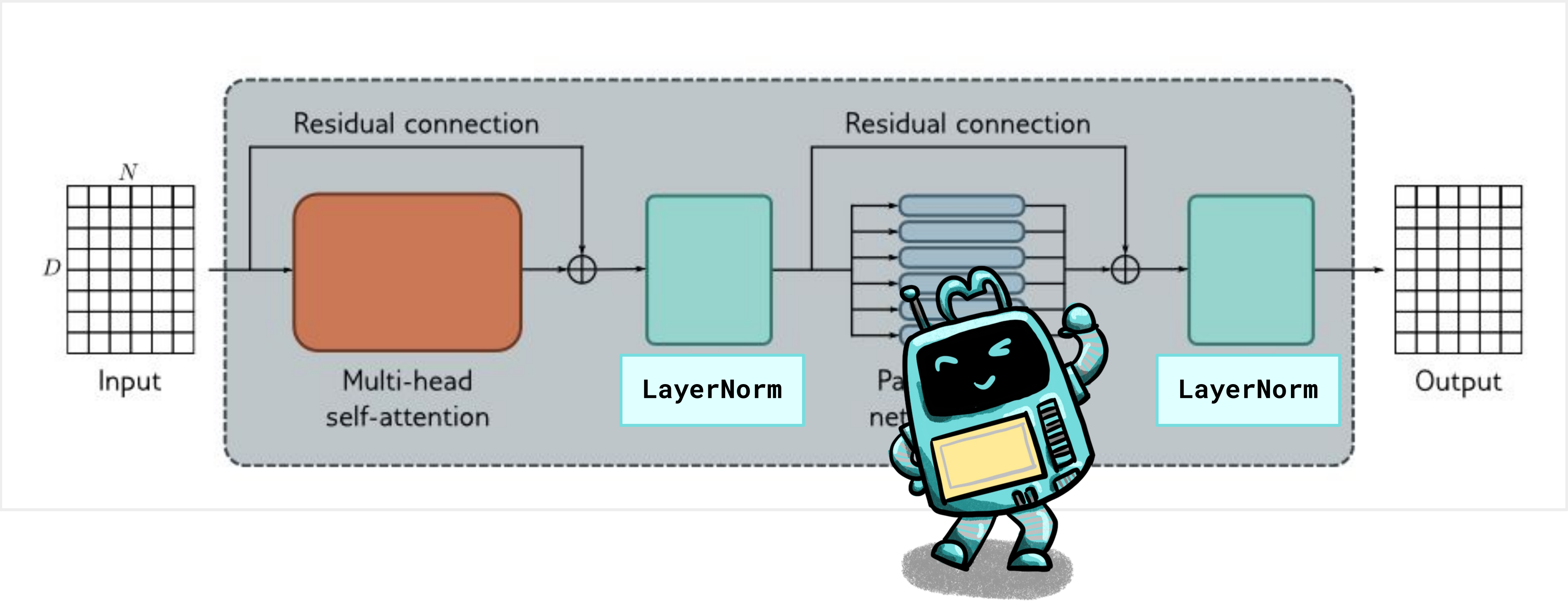
# of Errors



# of Iterations

He et al. (2015) Deep Residual Learning for Image Recognition

Transformer - LayerNorm

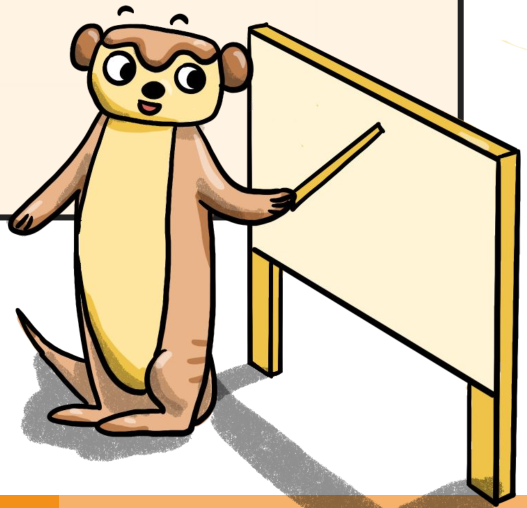


Simon J.D. Prince (2023). *Understanding Deep Learning*

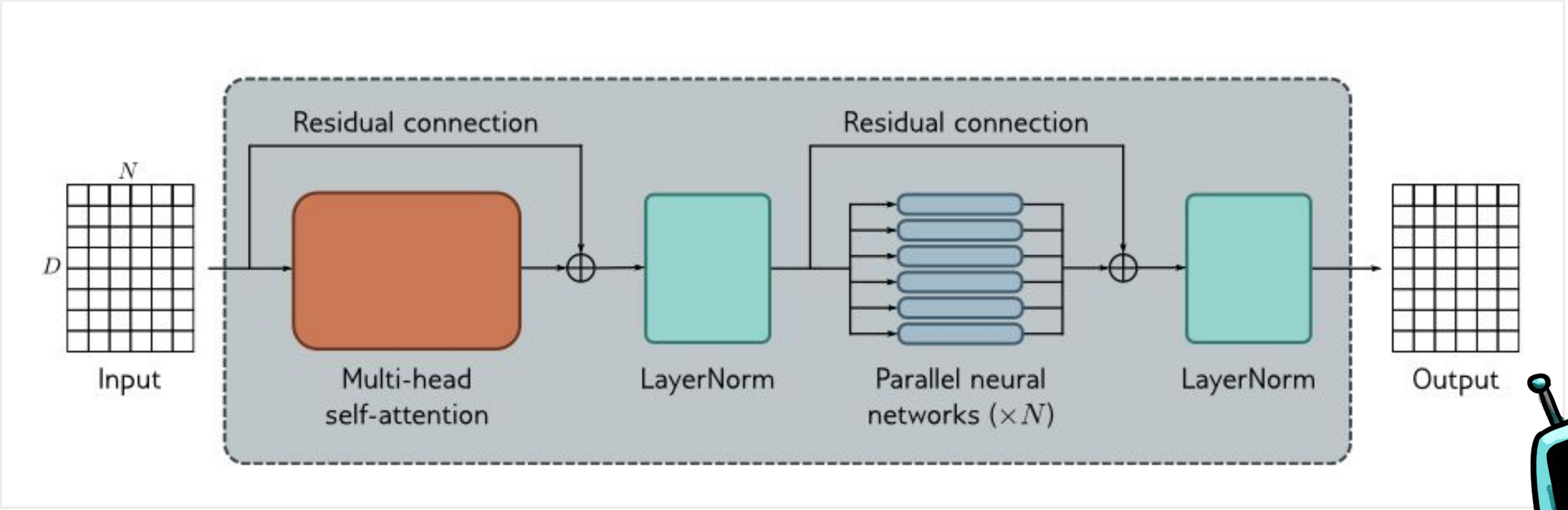
Layer Normalization

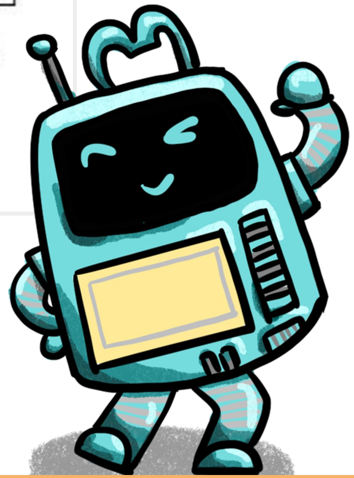
`torch.nn.LayerNorm(normalized_shape,…)`

- Reduce shattered gradients
- Stable forward propagation



Transformer Architecture

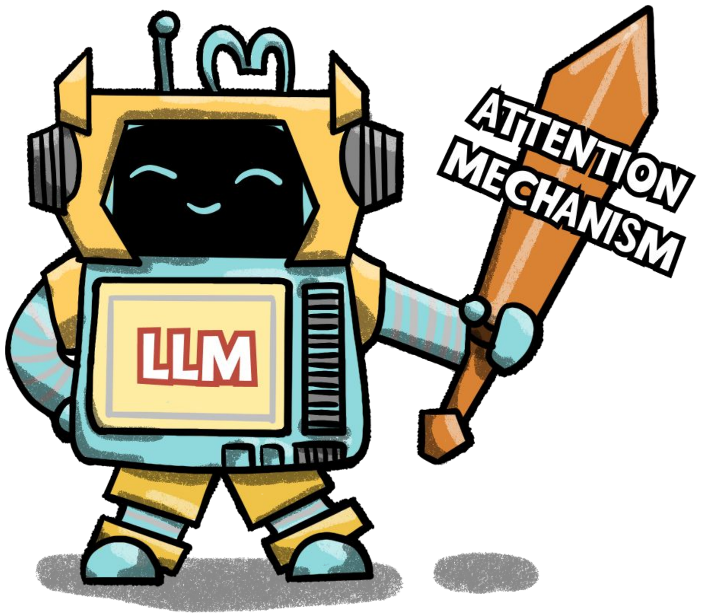




Simon J.D. Prince (2023). Understanding Deep Learning

LLM

Model Architecture Advancements



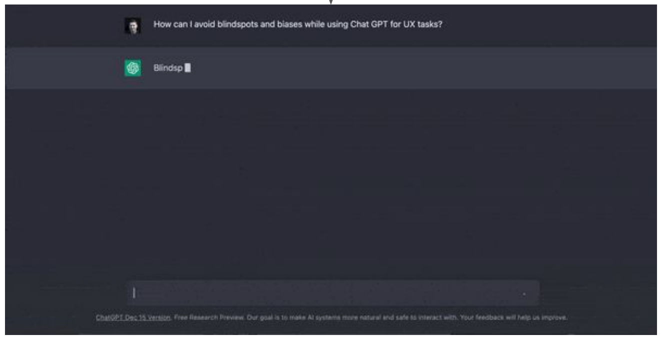
- ~~**Transformer**~~
  - ~~Word Vectors~~
  - ~~Attention Mechanism~~
  - ~~Residual Connections~~
- **Text Generation**
  - Predict next token
- **RLHF**

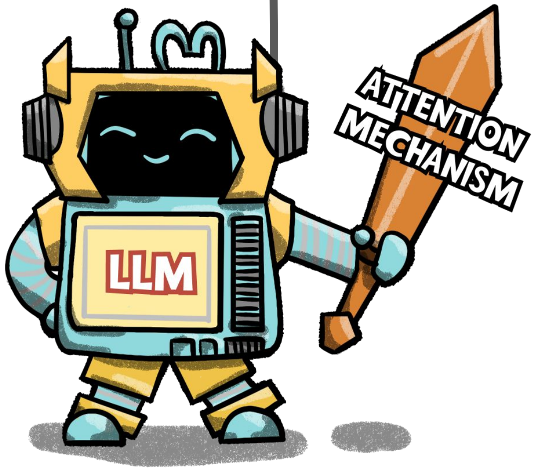
LLM - Large Language Models

Meerkats love to eat watermelons when they ...

Output

What do meerkats like to eat?

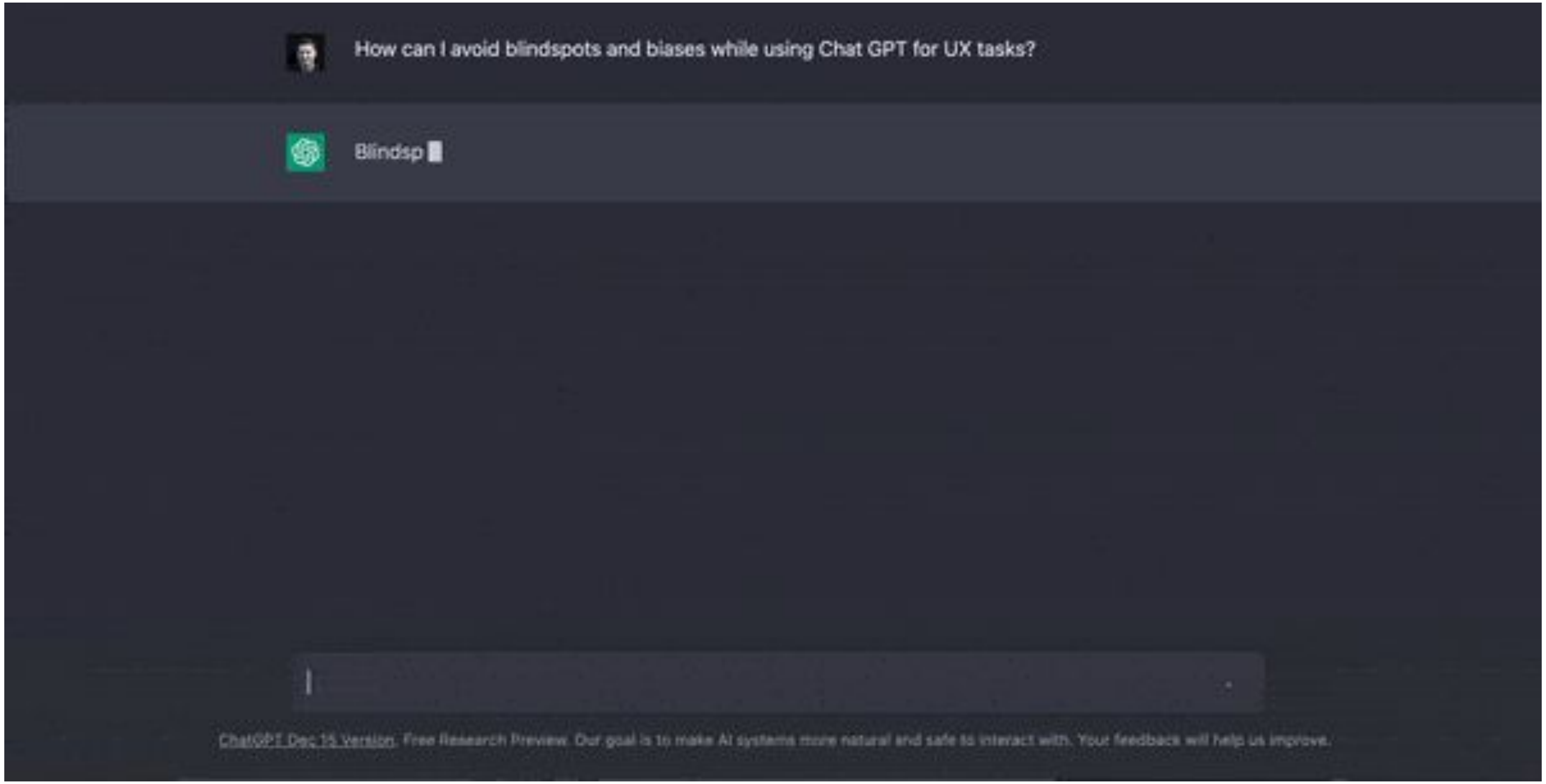


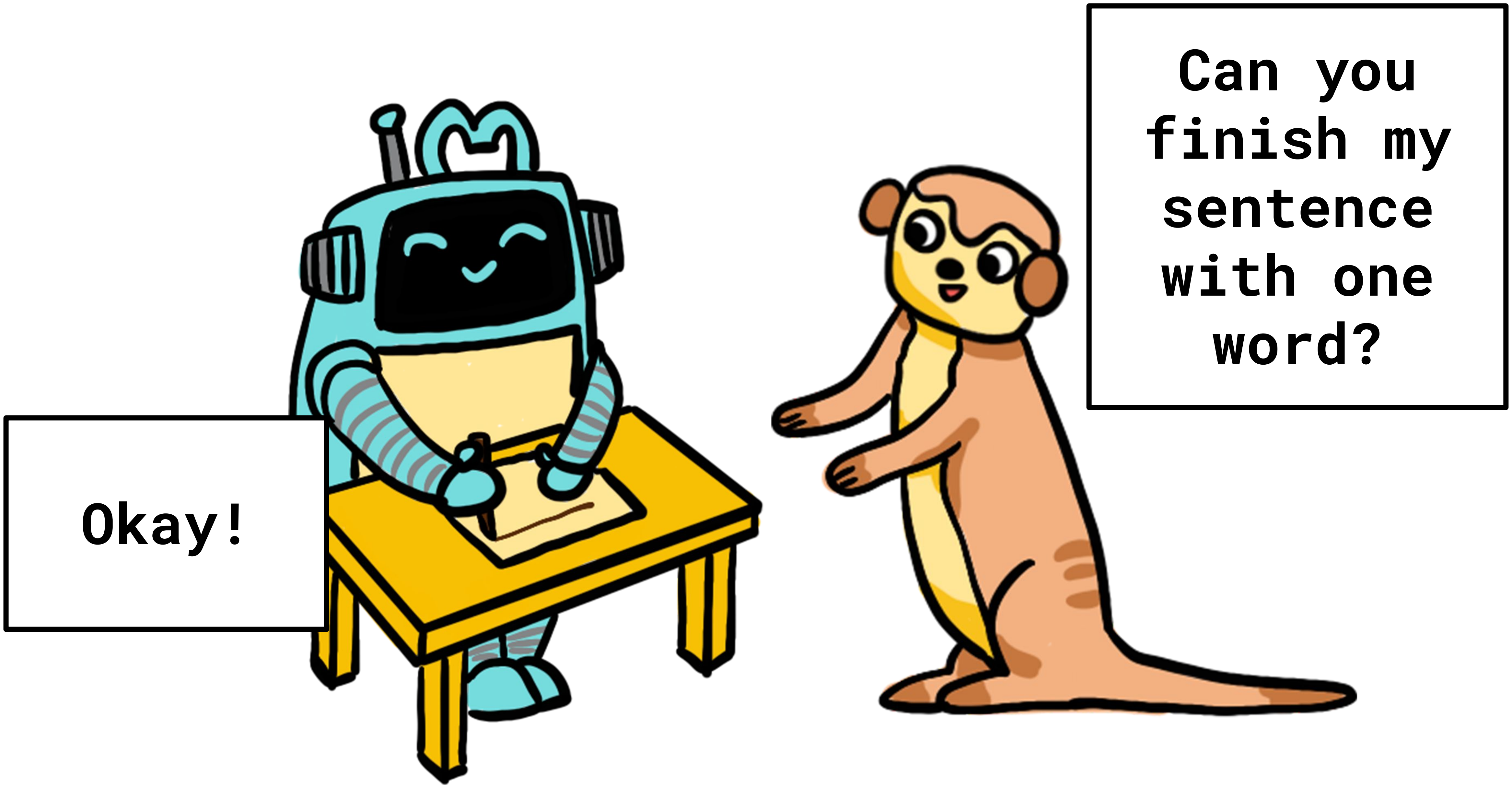


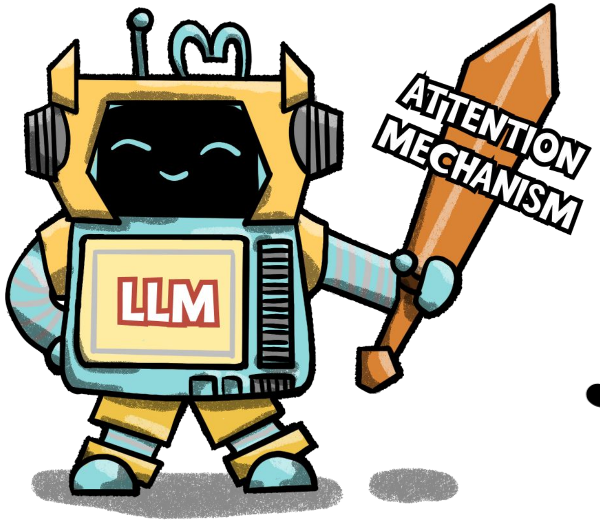
User

Input

ML Model


How can I avoid blindspots and biases while using Chat GPT for UX tasks?
Blindsp
ChatGPT Dec 15 Version. Free Research Preview. Our goal is to make AI systems more natural and safe to interact with. Your feedback will help us improve.

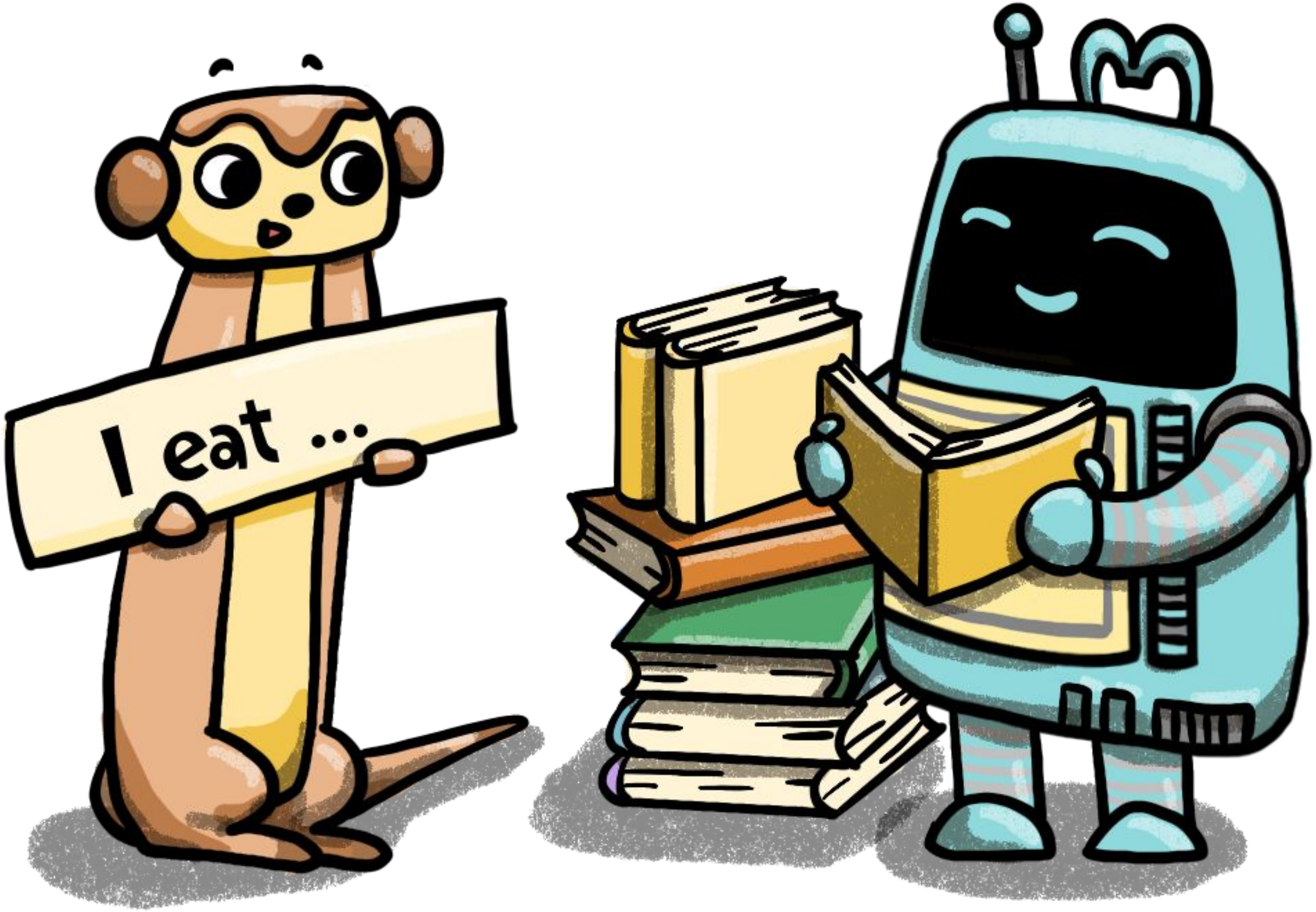

Okay!
Can you finish my sentence with one word?

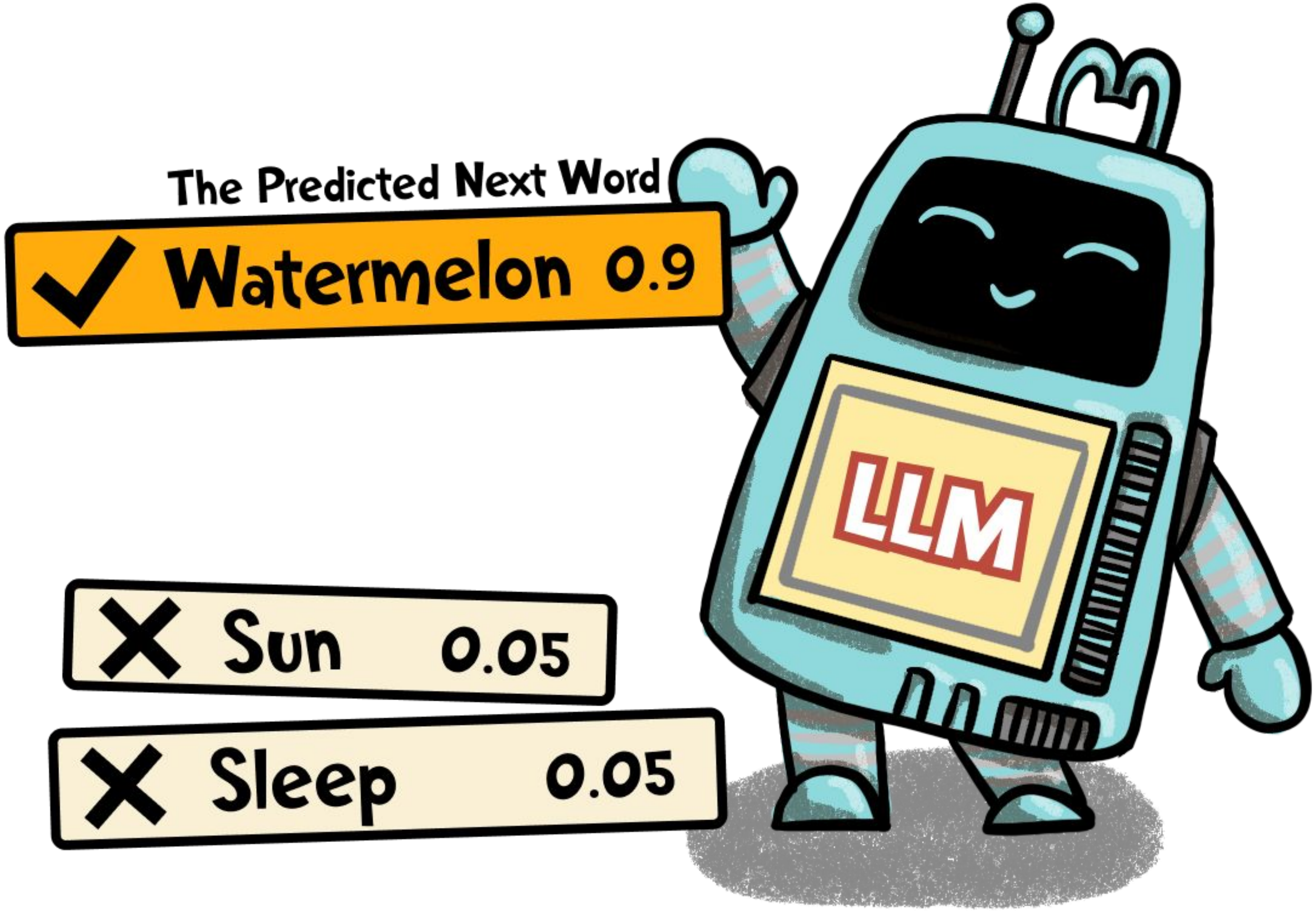


$$f(x) =$$

Entire Vocab	P
Watermelon	0.9
Sun	0.05
Sleep	0.05

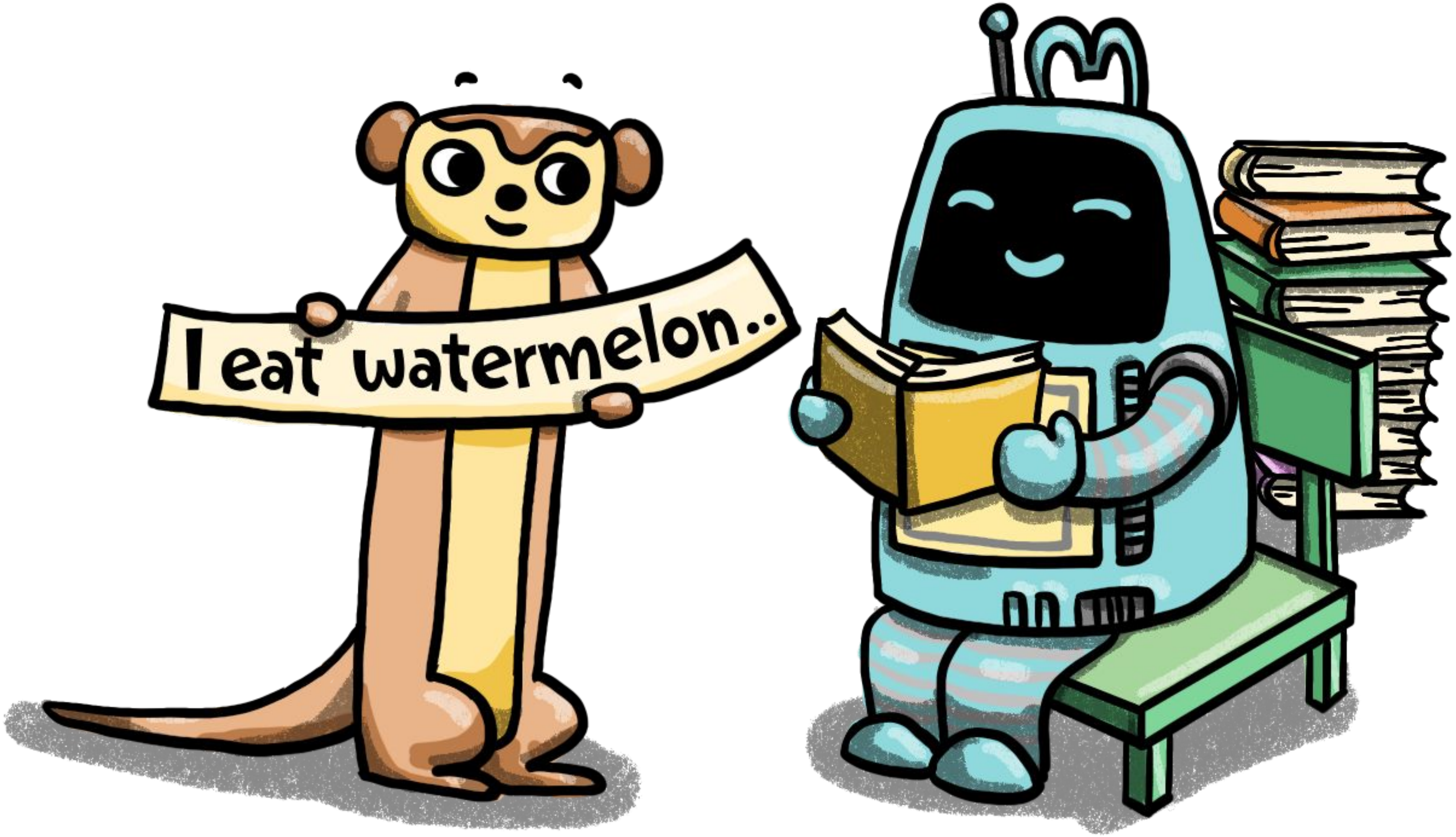
What do meerkats like to eat?


I eat ...


The Predicted Next Word
Watermelon 0.9
Sun 0.05
Sleep 0.05
LLM

Word Selection

- Greedy Search
- Beam Search
- Top-K Sampling
- …


I eat watermelon..

sigasia_llm_gen.py

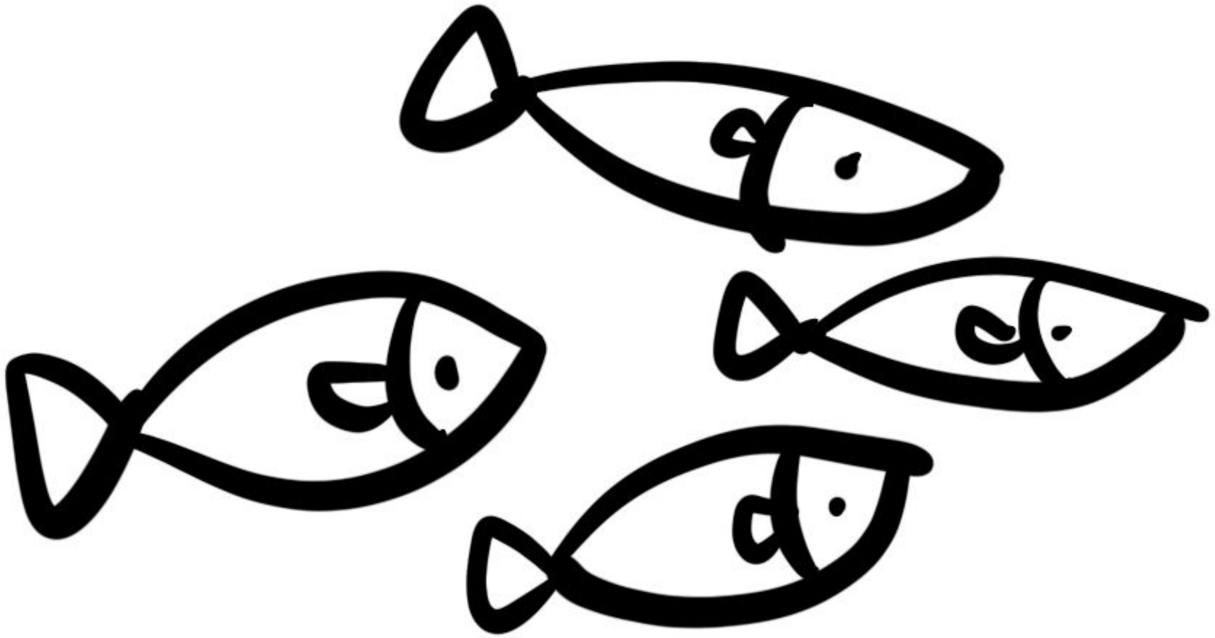
```python
from transformers import TFGPT2LMHeadModel, GPT2Tokenizer
from transformers.trainer_utils import set_seed

tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
model = TFGPT2LMHeadModel.from_pretrained("gpt2")
input_ids = tokenizer.encode('I enjoy walking with my cute dog', return_tensors="pt")
```

Can we capture the entire distribution?

Capturing Distribution

Our goal:
Generate fish that looks and behaves like it belongs to this river

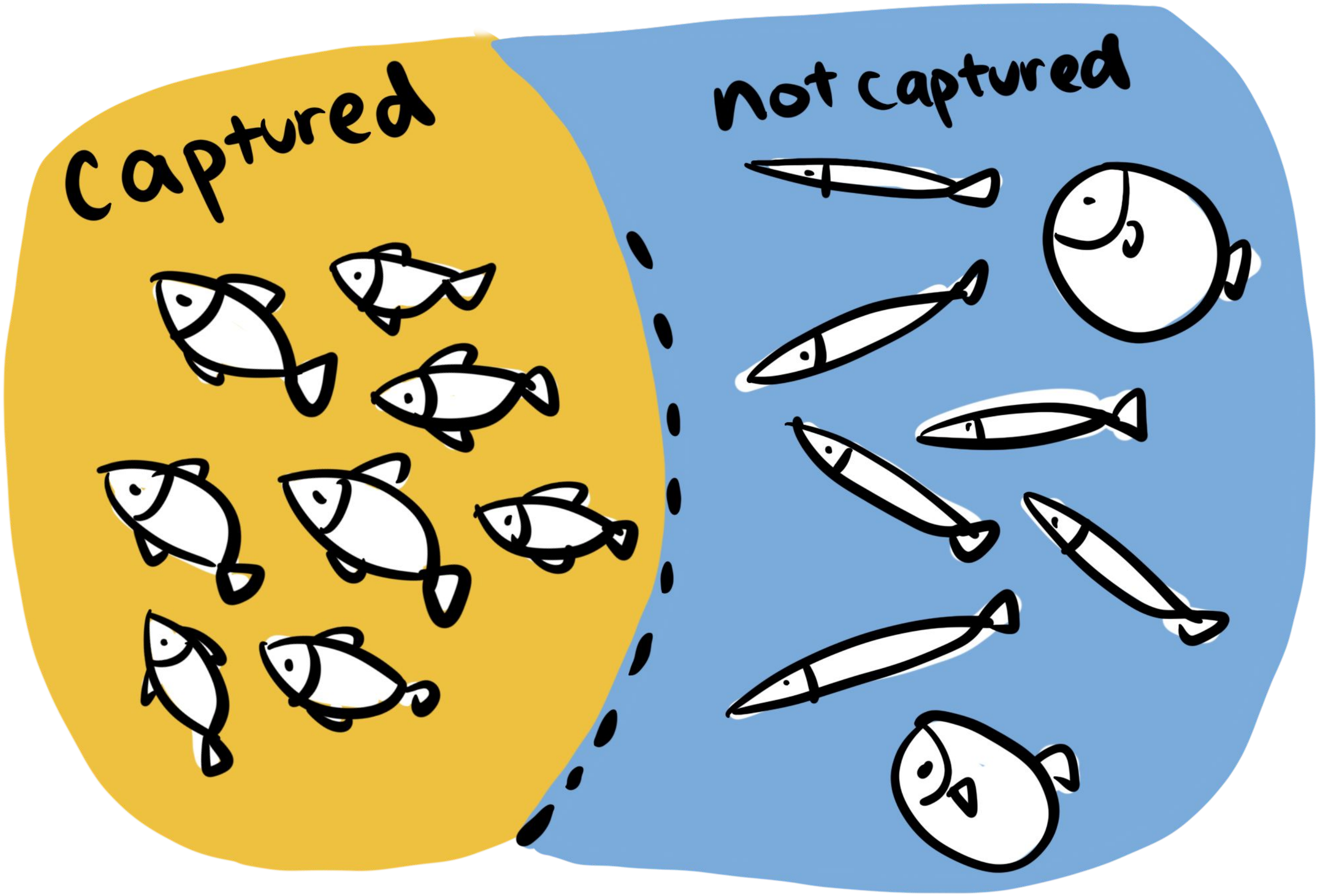
Capturing Distribution

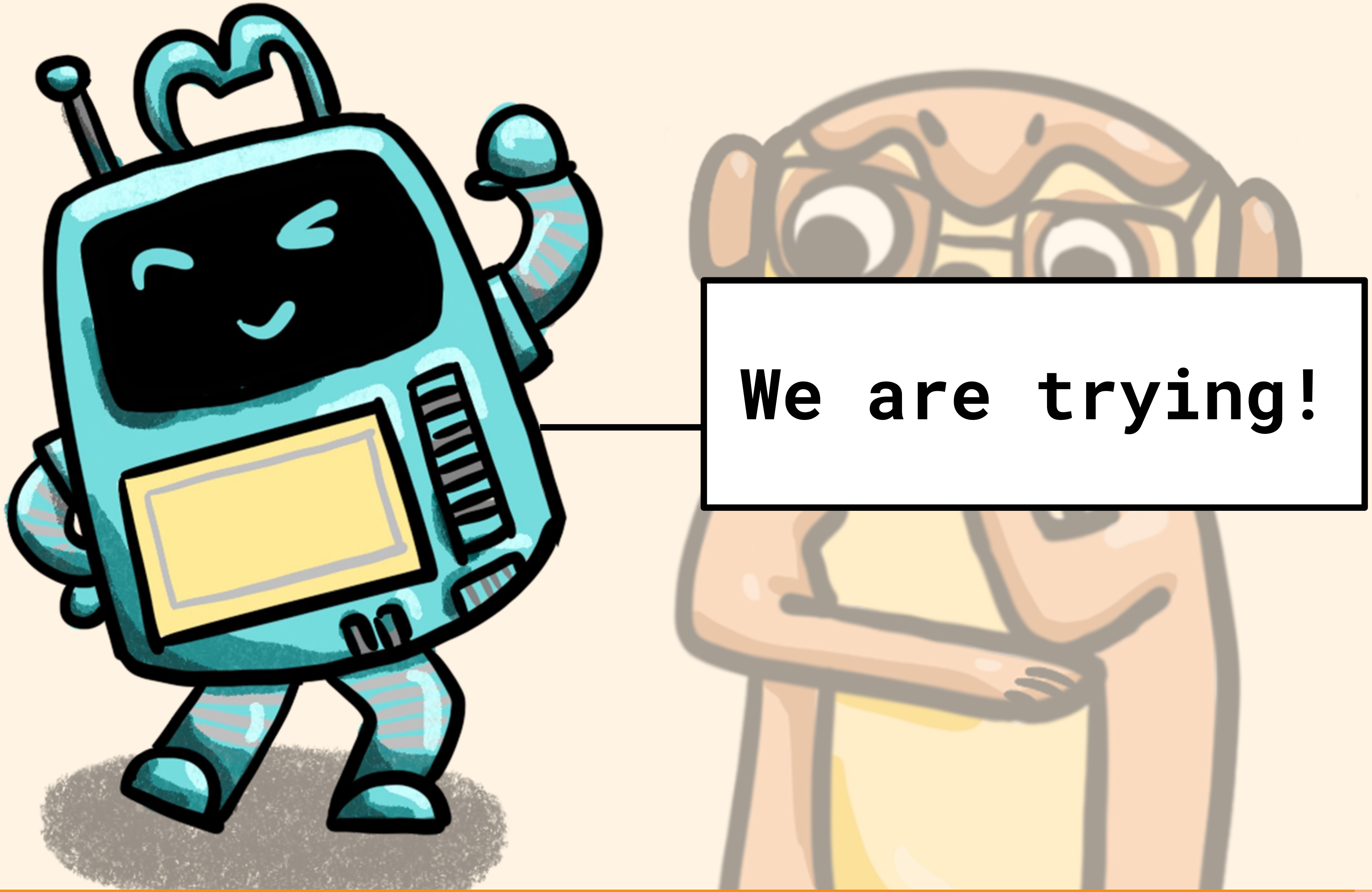


Capture fish from the river

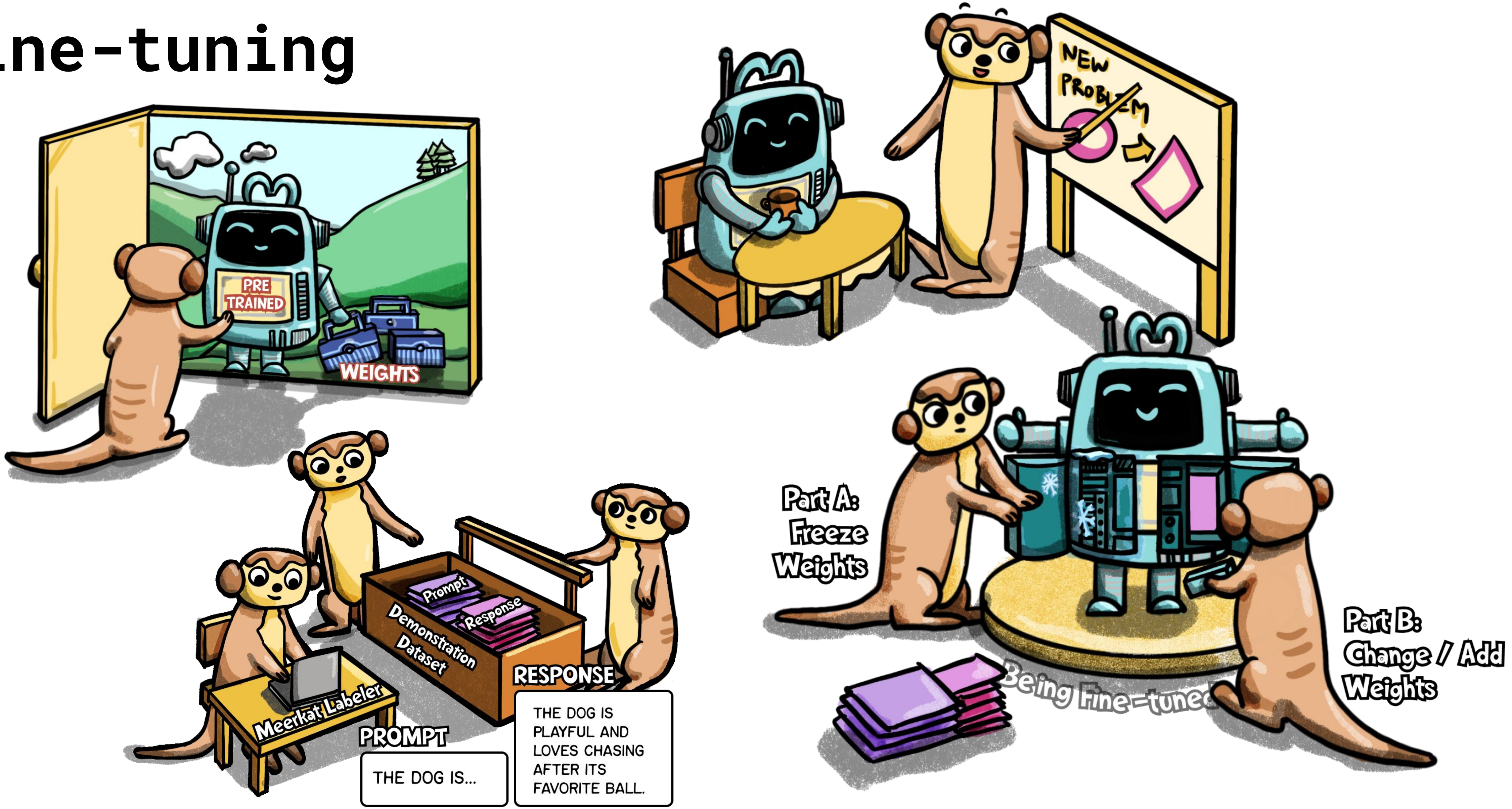
a lot of

Capturing Distribution

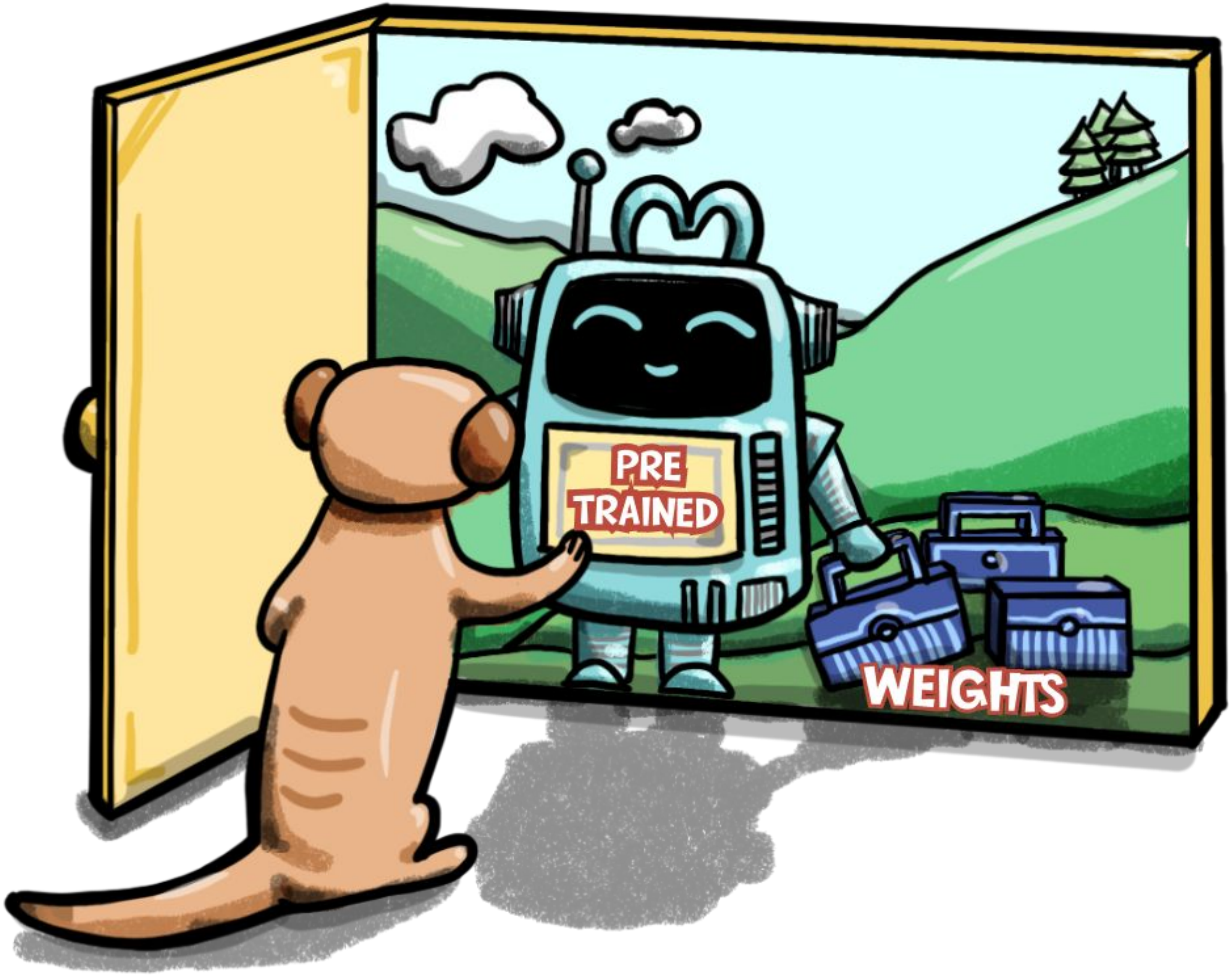


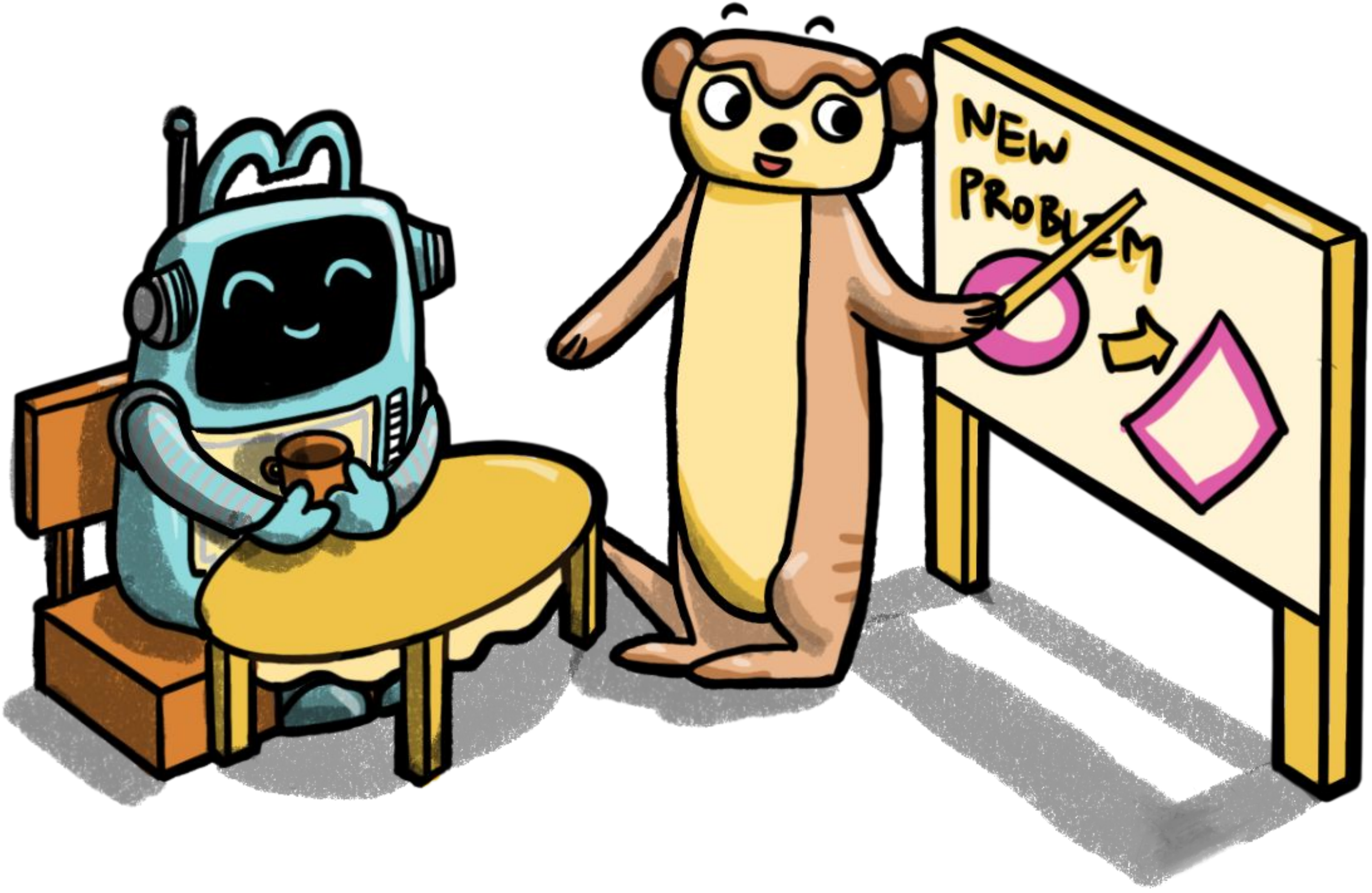

We are trying!

Fine-tuning

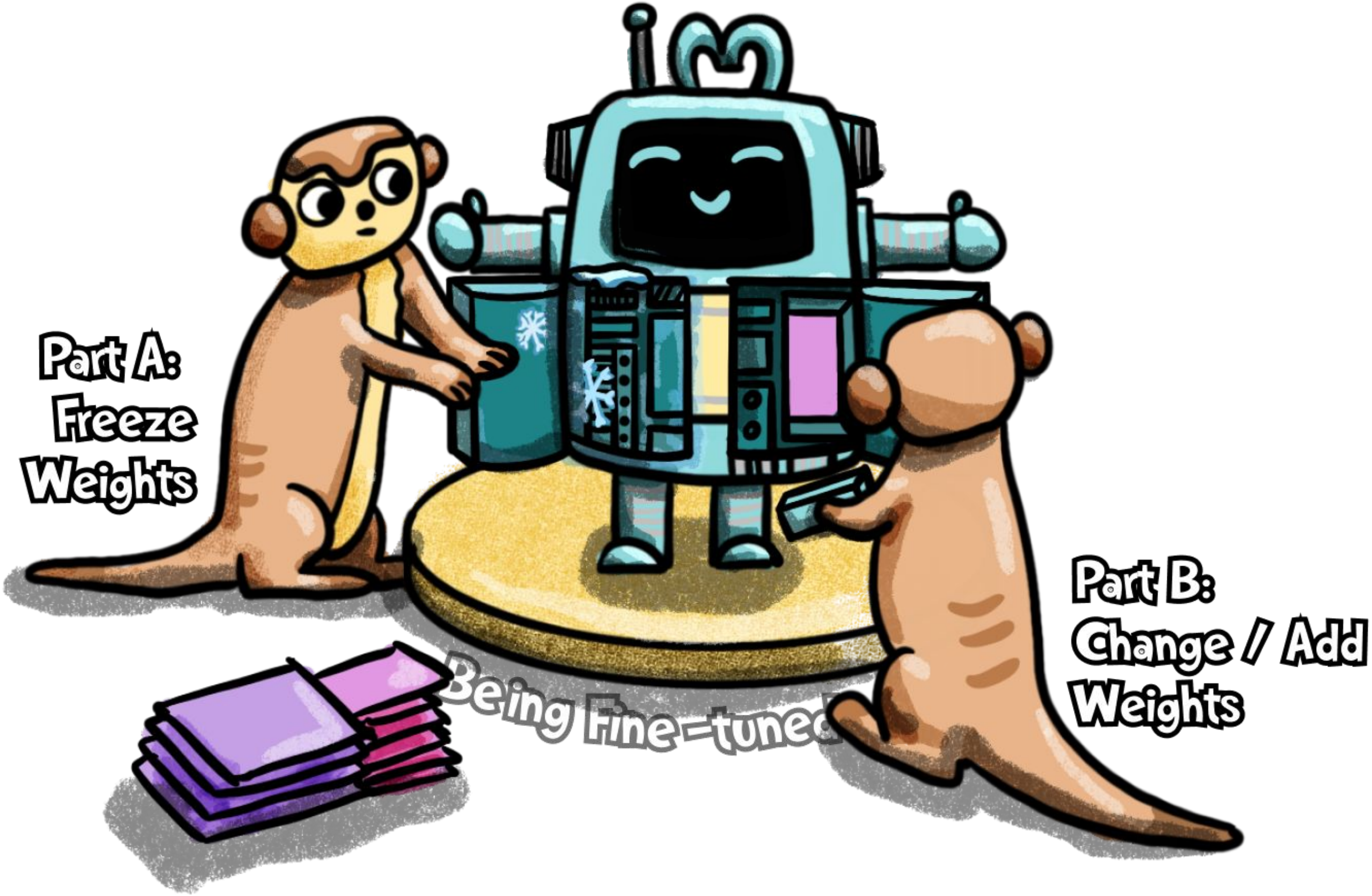


PRE TRAINED
WEIGHTS
NEW PROBLEM
Prompt
Response
Demonstration Dataset
Meerkat Labeler
PROMPT
THE DOG IS...
RESPONSE
THE DOG IS PLAYFUL AND LOVES CHASING AFTER ITS FAVORITE BALL.
Part A: Freeze Weights
Part B: Change / Add Weights
Being Fine-tuned


PRE
TRAINED
WEIGHTS


NEW
PROBLEM


Prompt
Response
Demonstration Dataset
Meerkat Labeler
RESPONSE
THE DOG IS PLAYFUL AND LOVES CHASING AFTER ITS FAVORITE BALL.
PROMPT
THE DOG IS...

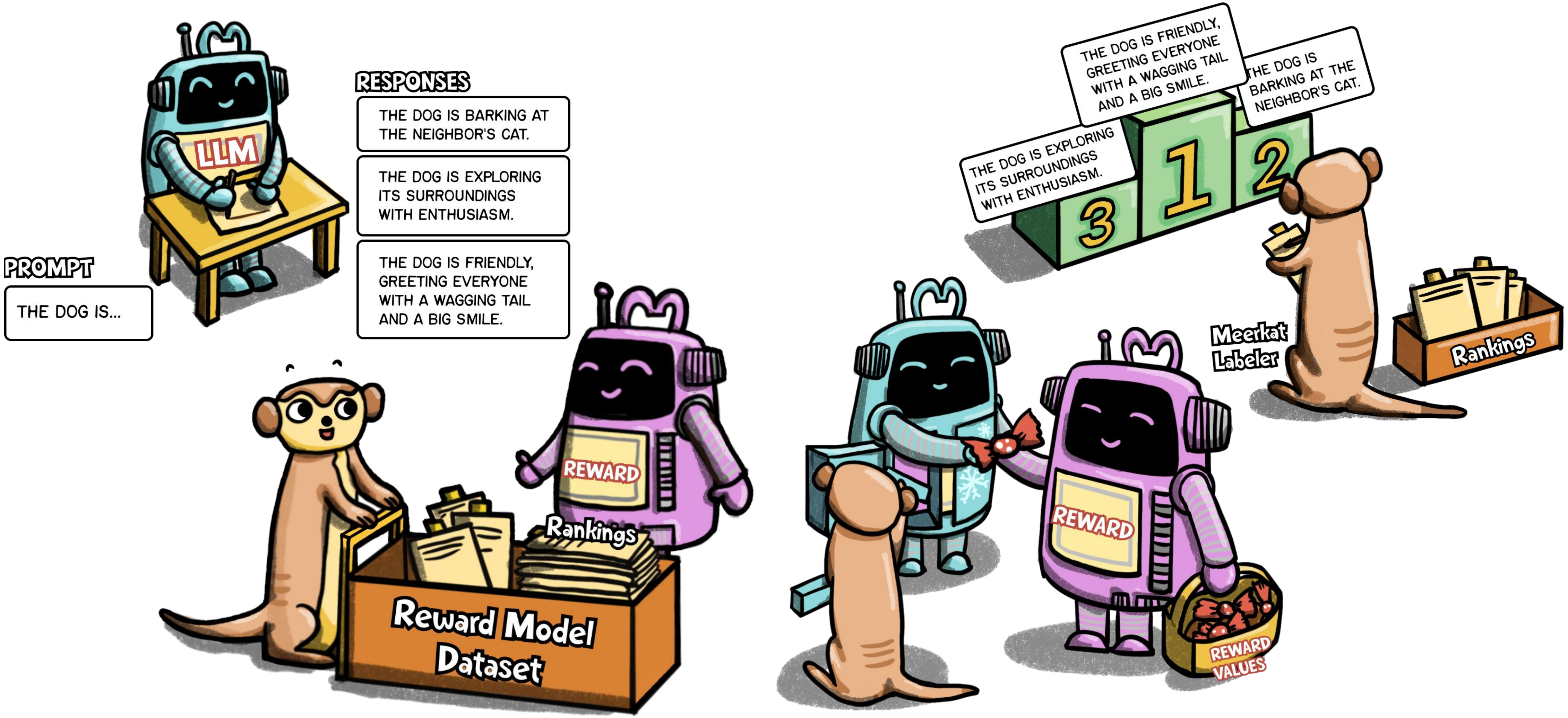

Part A:
Freeze
Weights
Being Fine-tuned
Part B:
Change / Add
Weights

sigasia_llm_ft.py

```python
from transformers import AutoModelForSeq2SeqLM, DataCollatorForSeq2Seq, Seq2SeqTrainingArguments, Seq2Se

# Set up data
# Clean up, tokenize ...

# Set up pretrained model, and how we want to finetune
model = AutoModelForSeq2SeqLM.from_pretrained(model_checkpoint)
batch_size = 16
args = Seq2SeqTrainingArguments(
    f"{model_name}-finetuned-xsum",
    evaluation_strategy = "epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    weight_decay=0.01,
    num_train_epochs=1,
    predict_with_generate=True
)

```

```python
def compute_metrics(eval_pred):
  # decode prediction into text
  # use some metric
  # ...

# Set up trainer
trainer = Seq2SeqTrainer(
    model,
    args,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["validation"],
    tokenizer=tokenizer,
    compute_metrics=compute_metrics
)
trainer.train()
```

Reinforcement Learning w/ Human Feedback

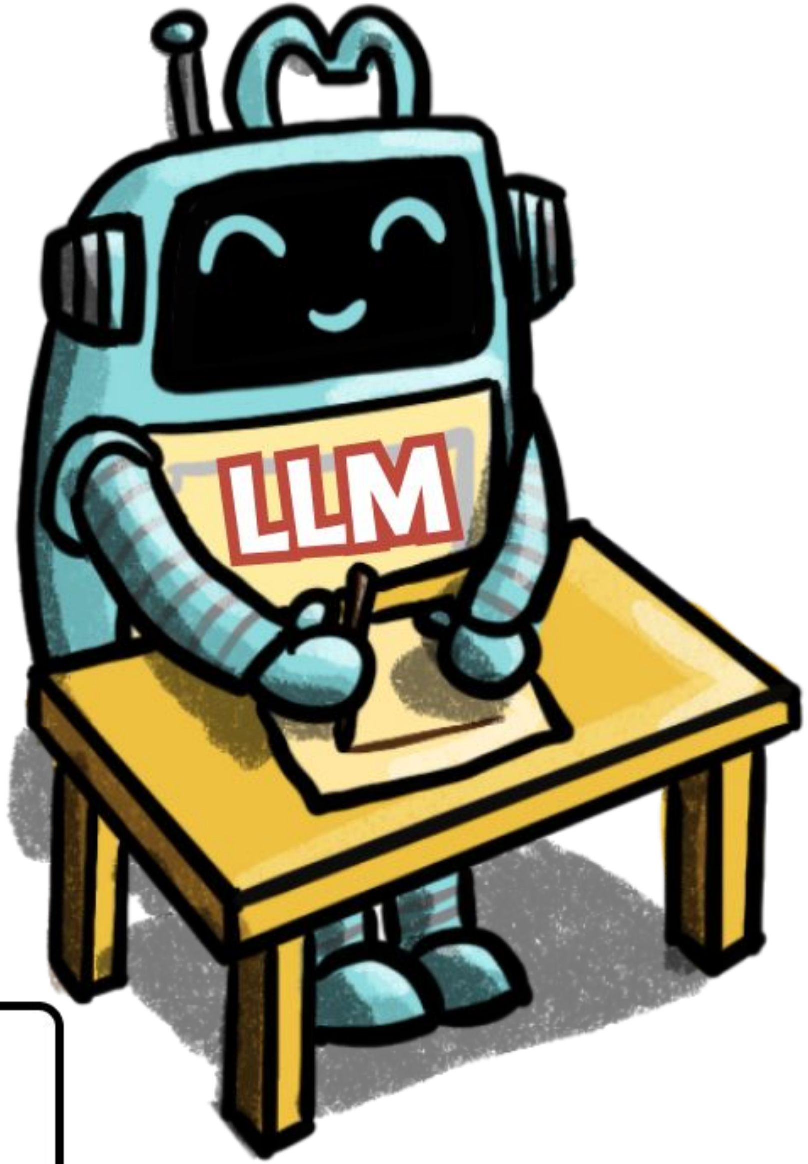


RESPONSES

THE DOG IS BARKING AT THE NEIGHBOR'S CAT.

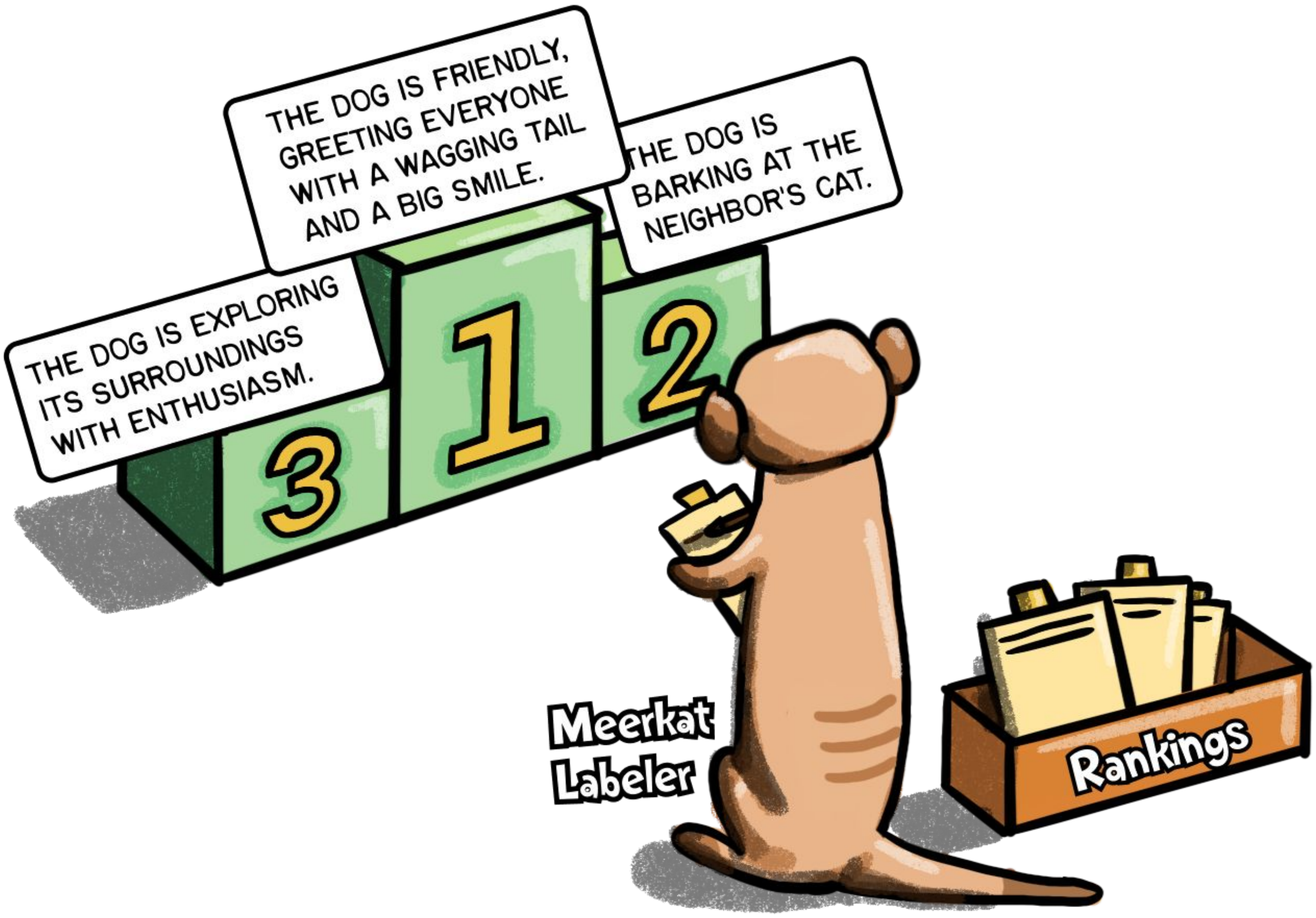
THE DOG IS EXPLORING ITS SURROUNDINGS WITH ENTHUSIASM.

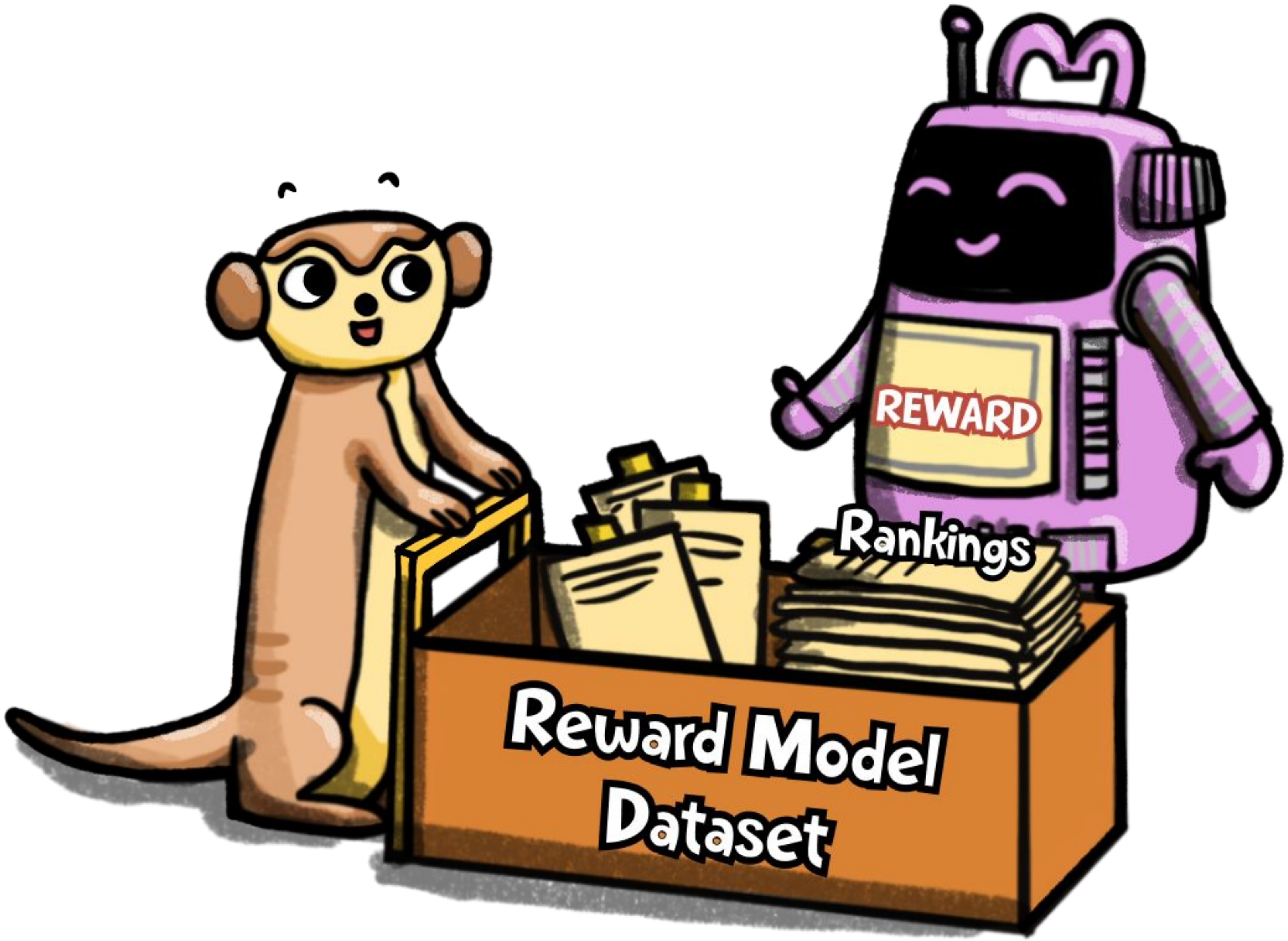
THE DOG IS FRIENDLY, GREETING EVERYONE WITH A WAGGING TAIL AND A BIG SMILE.



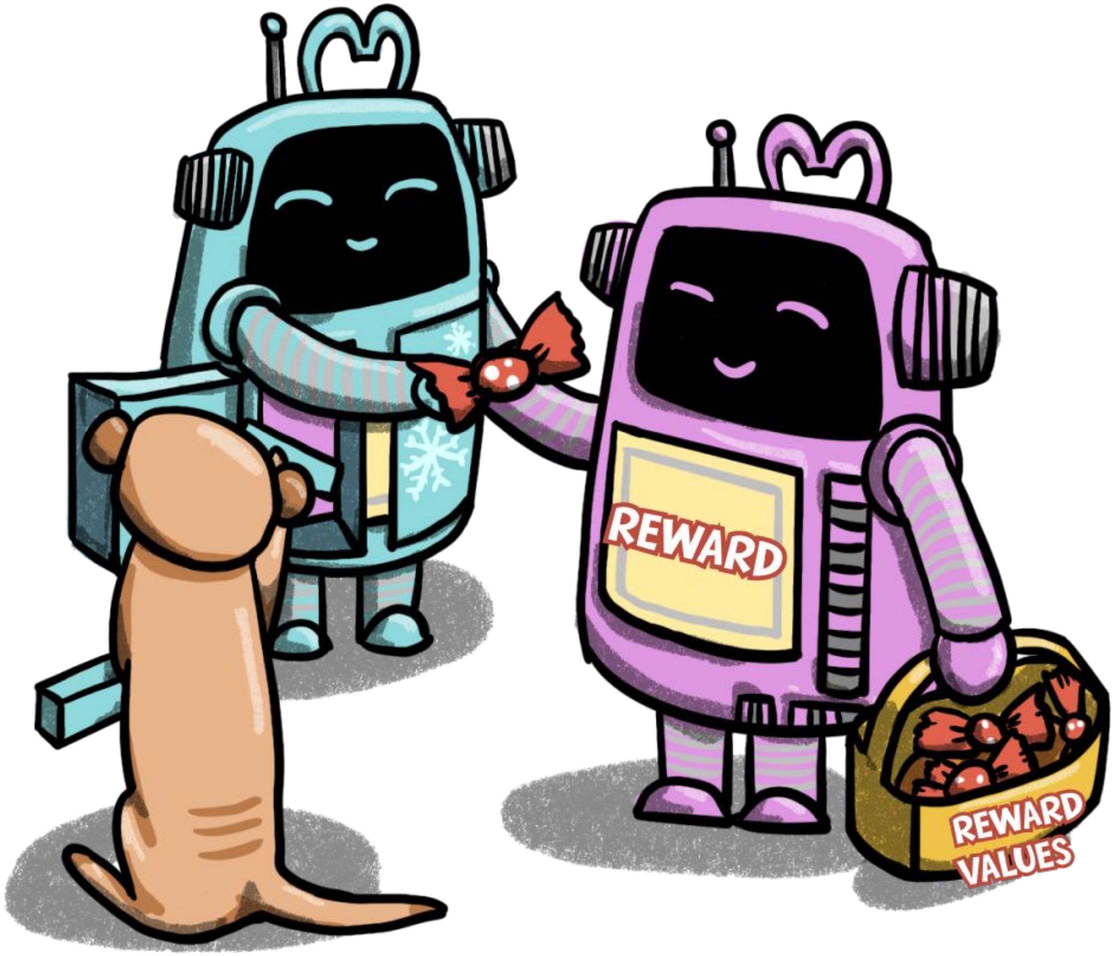
PROMPT

THE DOG IS...


THE DOG IS FRIENDLY, GREETING EVERYONE WITH A WAGGING TAIL AND A BIG SMILE.
THE DOG IS BARKING AT THE NEIGHBOR'S CAT.
THE DOG IS EXPLORING ITS SURROUNDINGS WITH ENTHUSIASM.
3
1
2
Meerkat Labeler
Rankings


REWARD
Rankings
Reward Model Dataset

Reward Model Fine-tuning

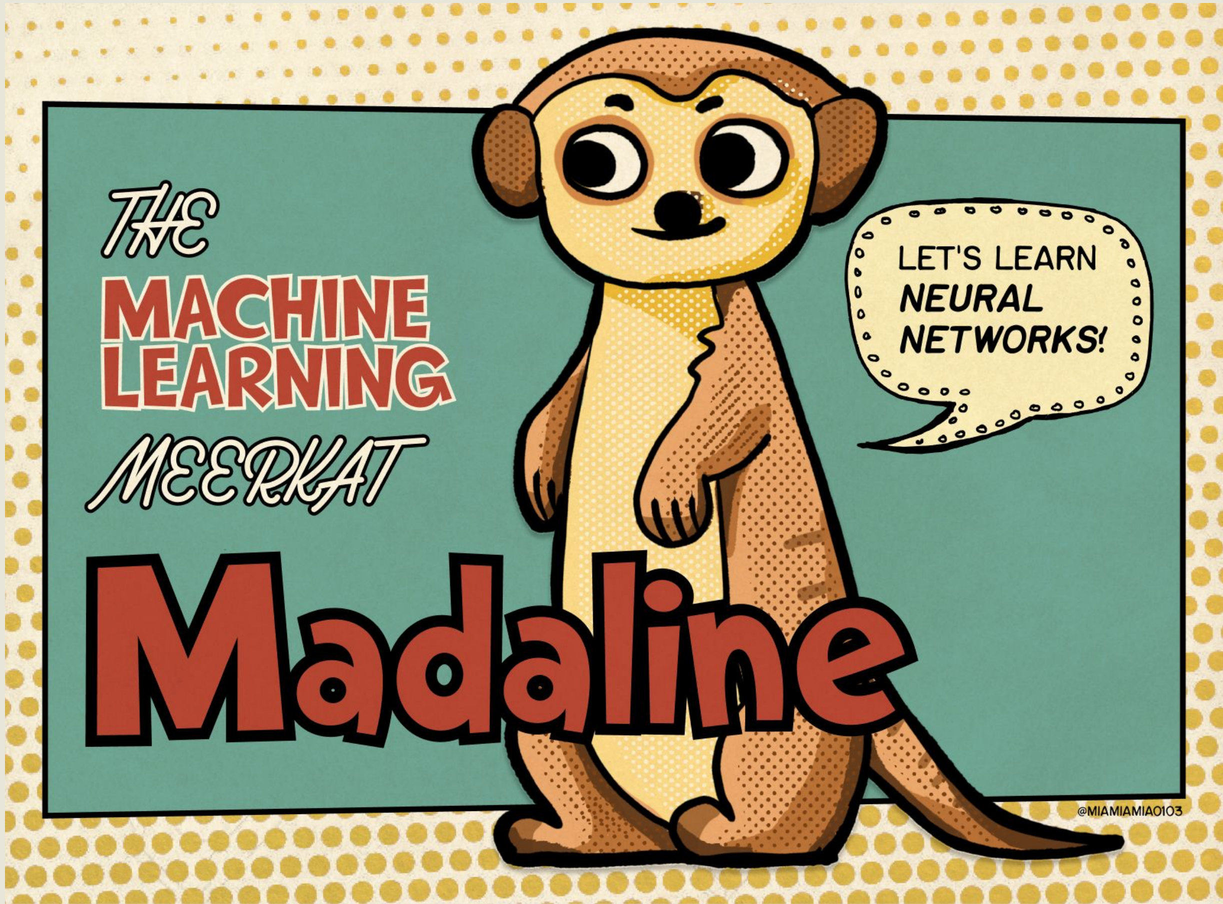


10 Minute Break

Machine Learning & Neural Networks

4

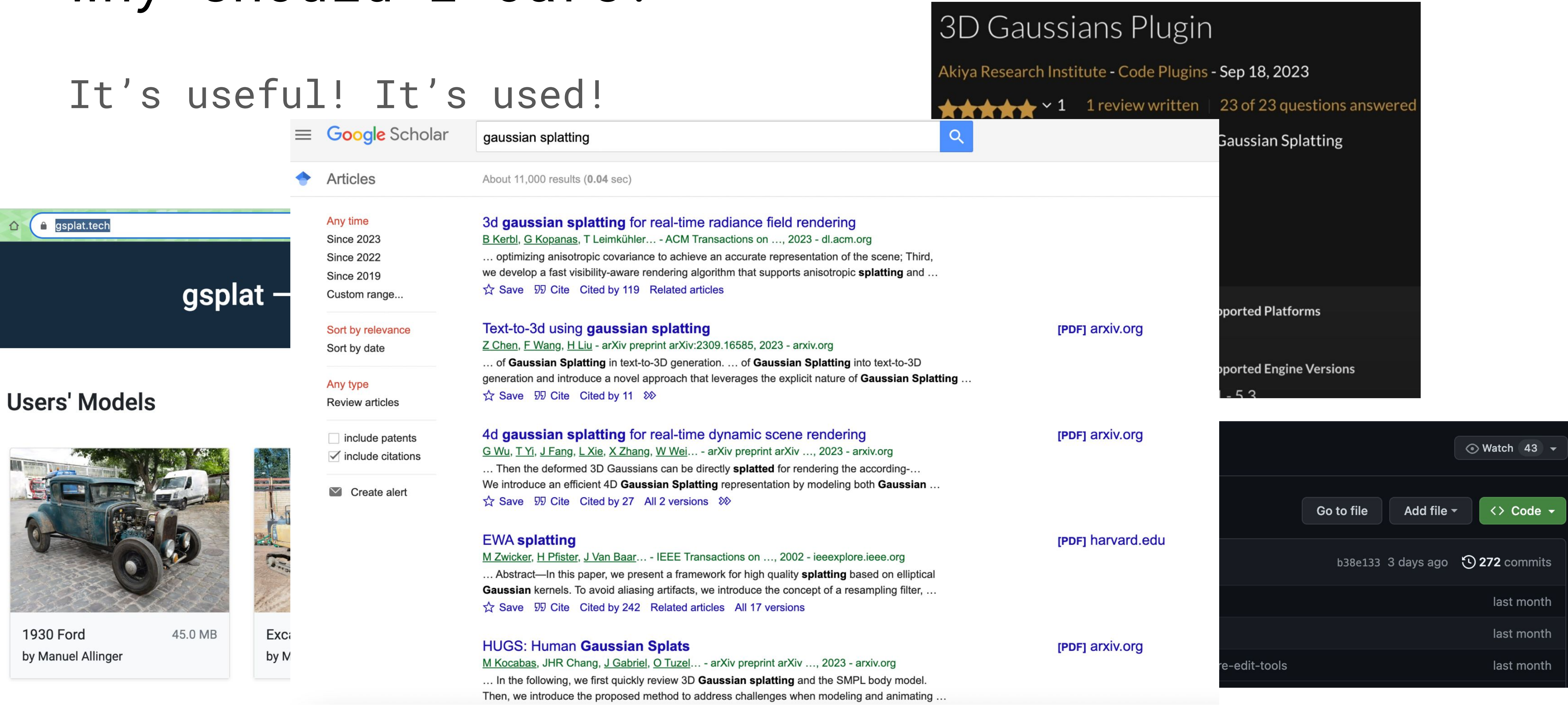
Rajesh Sharma @xarmalarma
Mia Tang @miamiamia0103



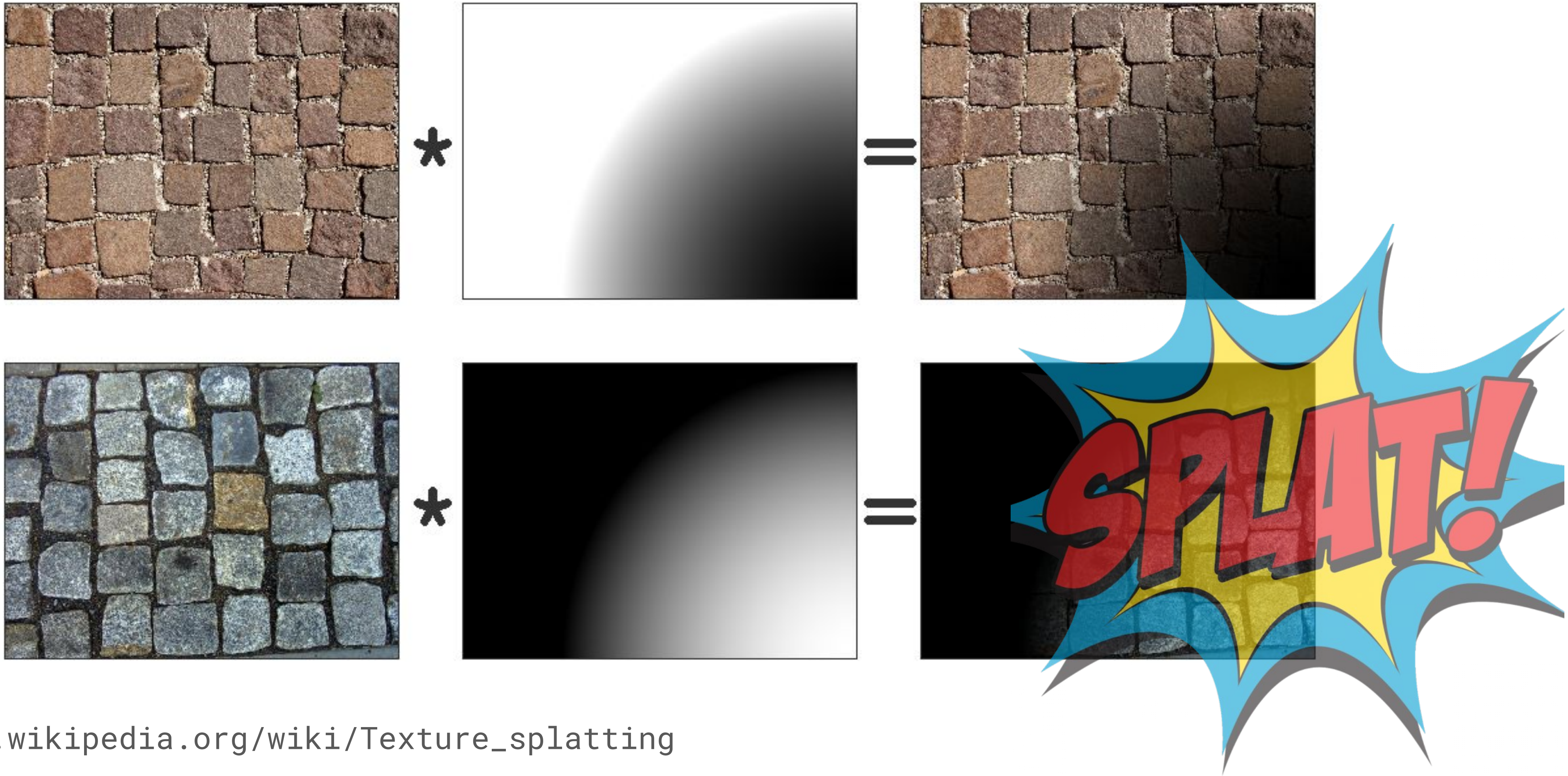
3D Gaussian Splatting

Why should I care?

It's useful! It's used!

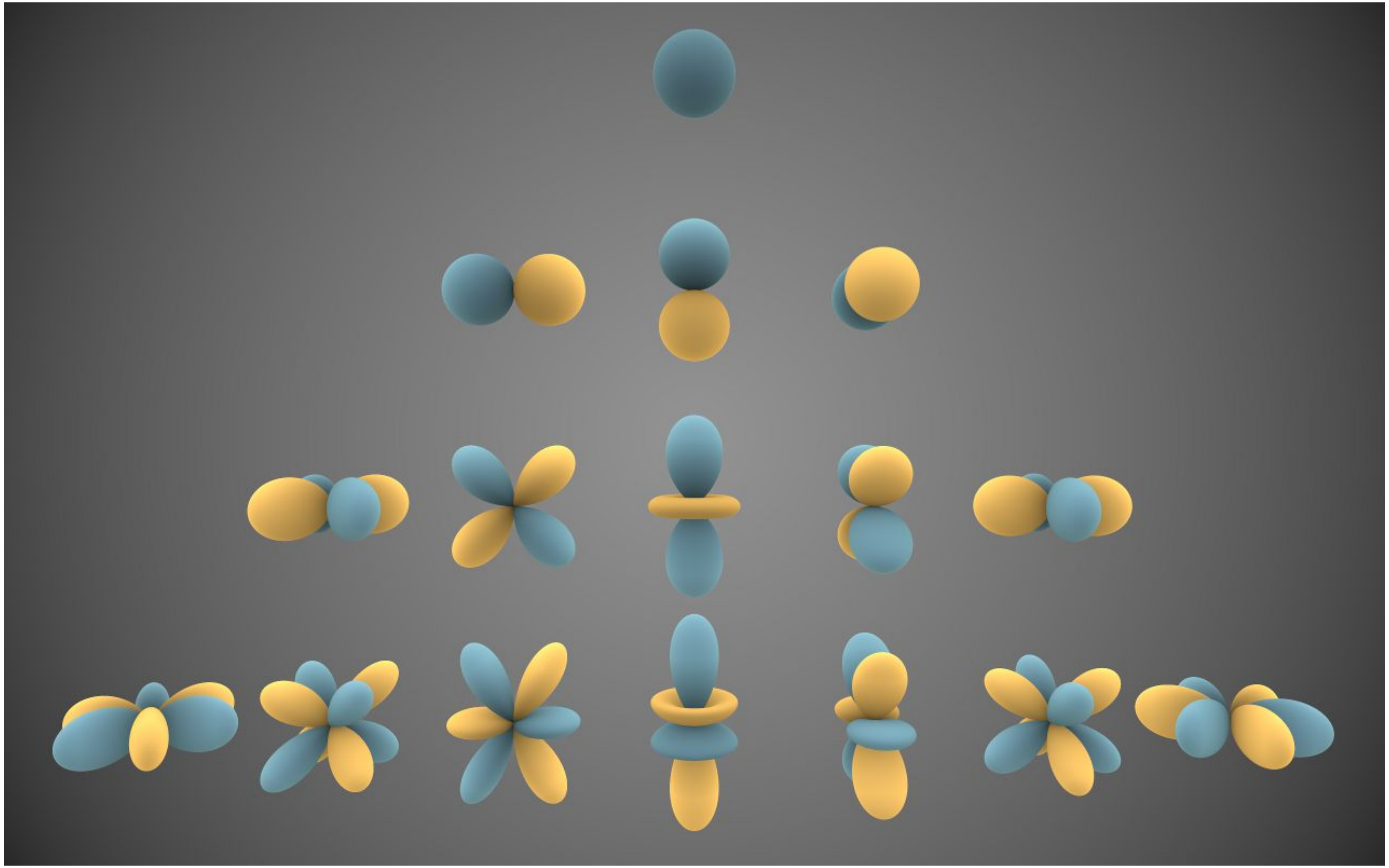


What's Splatting?



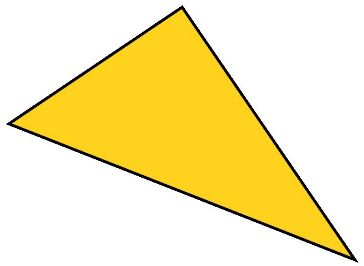
https://en.wikipedia.org/wiki/Texture_splatting

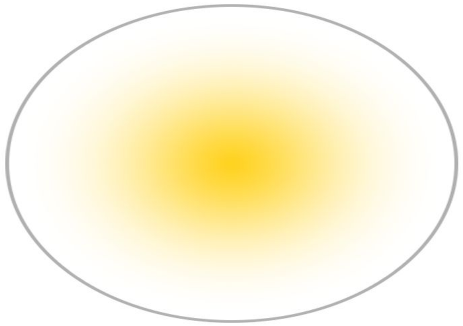
What’s 3D Gaussian?



Gaussian Splatting

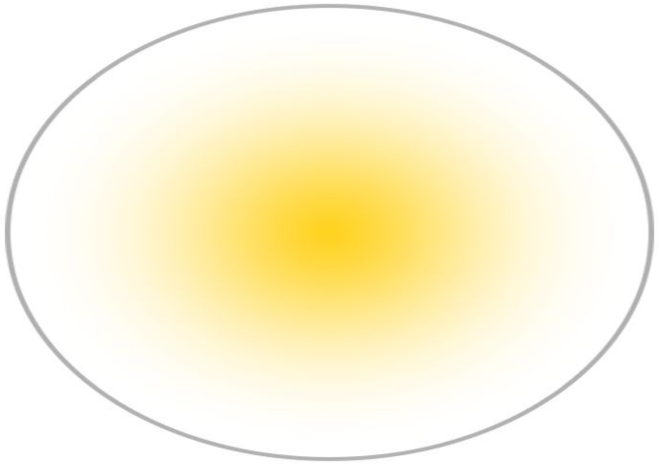
- A rasterization technique
- analogous to triangle rasterization in computer graphics
- Triangles → Gaussians





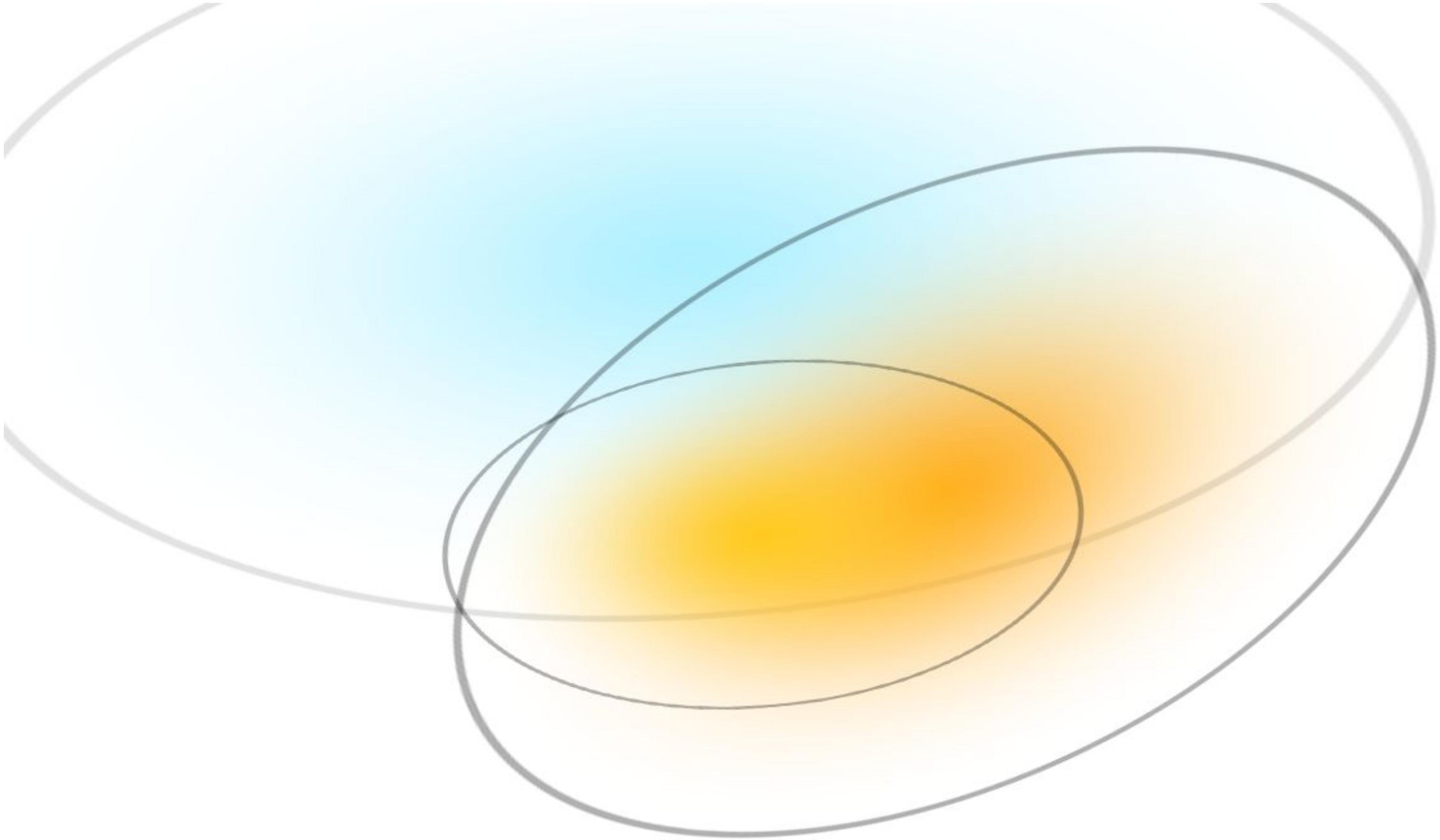
Images from https://huggingface.co/blog/gaussian-splatting

Gaussian Splatting Definition



- **Position**: where it's located (XYZ)
- **Covariance**: how it's stretched/scaled (3x3 matrix)
- **Color**: what color it is (RGB)
- **Alpha**: how transparent it is (α)

Gaussian Splatting



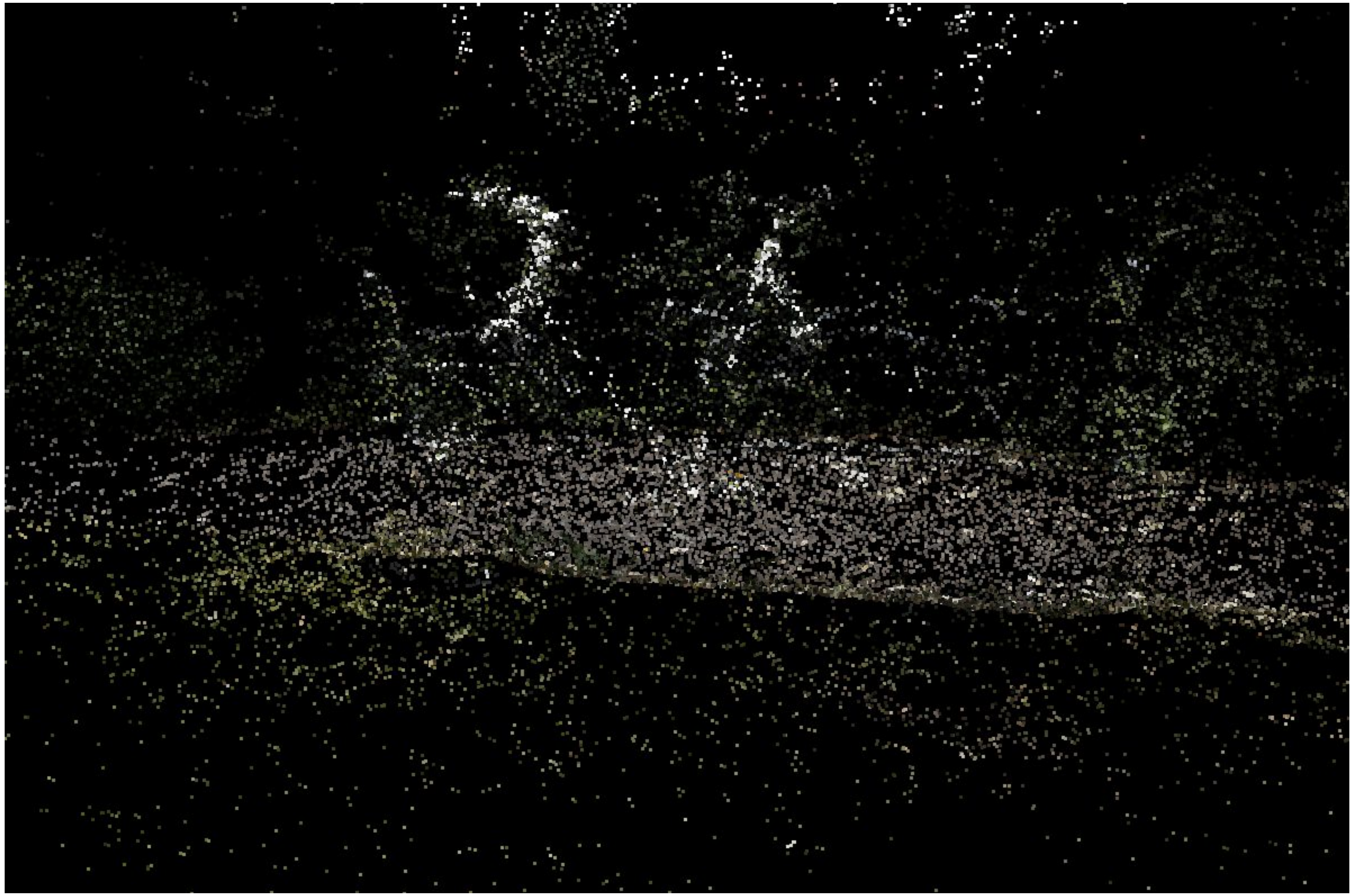
3 Gaussians

Gaussian Splatting

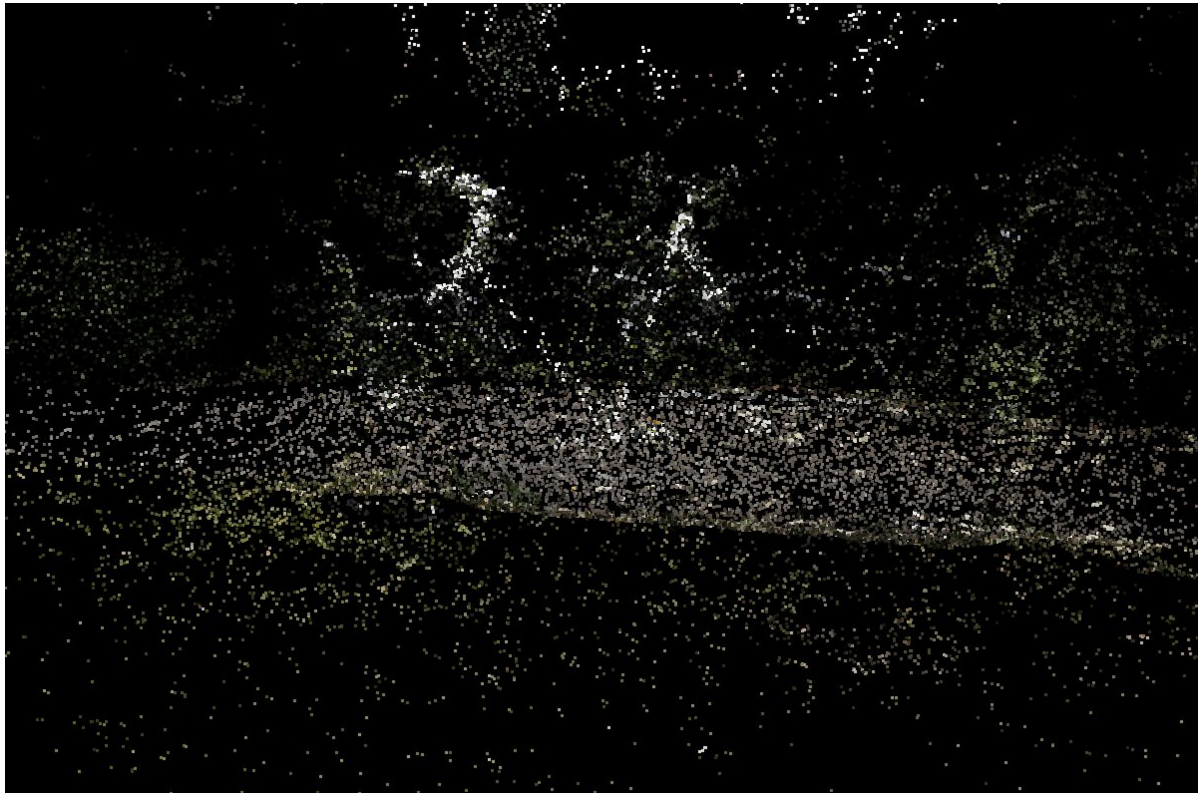
7 Million Gaussians

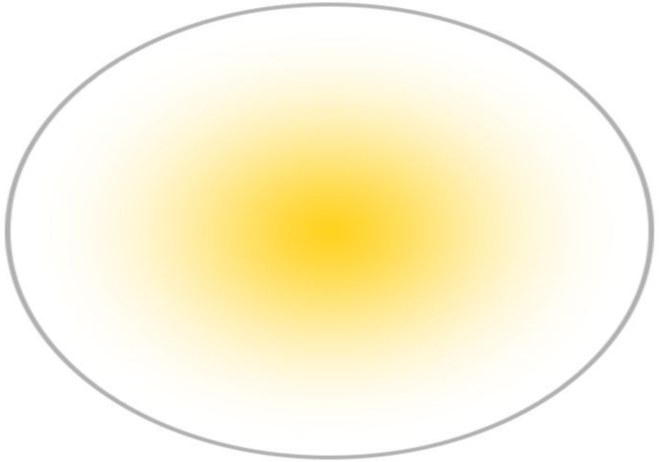
Step One: Structure From Motion

- Structure from Motion (SfM) method
- estimate a point cloud from a set of images



Step Two: Point Cloud → Gaussians





/ each point

(position, color)
(alpha, covariance) 🤖

Gaussian Splatting Training

1. Rasterize the gaussians to an image using differentiable gaussian rasterization
2. Calculate the loss based on the difference between the rasterized image and ground truth image
3. Adjust the gaussian parameters according to the loss
4. Apply automated densification and pruning
5. Clone and split gaussians when needed

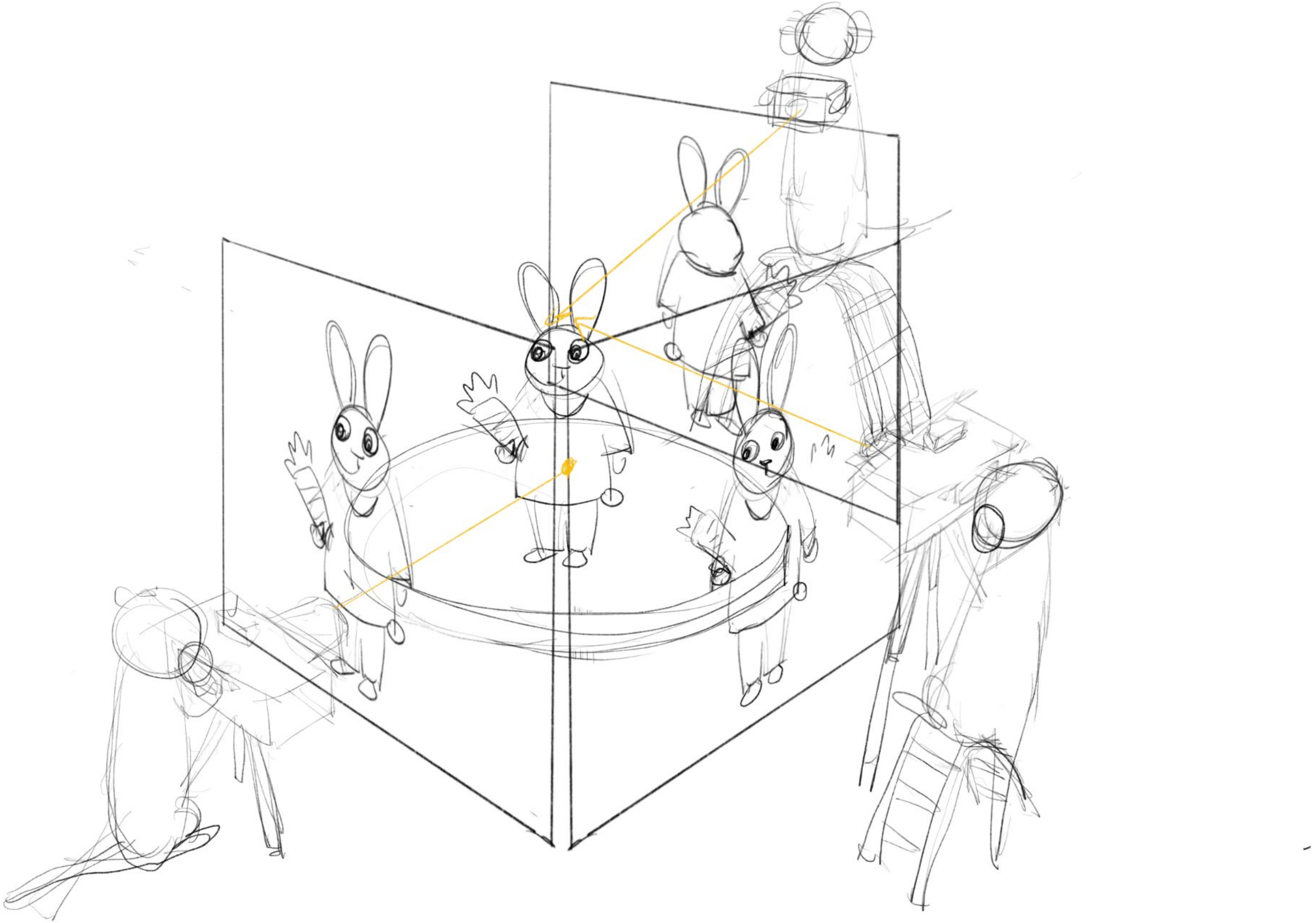
How to render a gaussian splat?

1. Project each gaussian into 2D from the camera perspective.
2. Sort the gaussians by depth.
3. For each pixel, iterate over each gaussian front-to-back, blending them together.

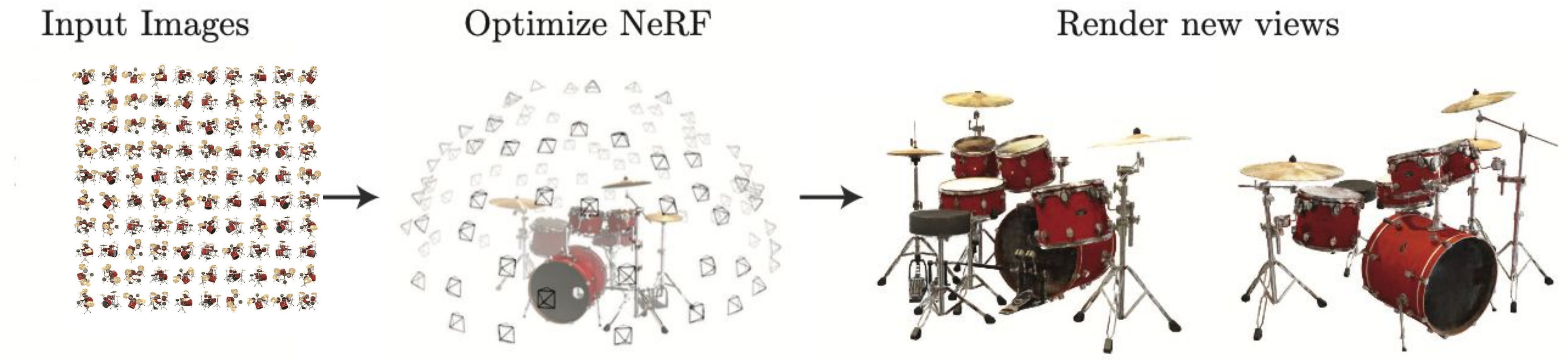
Fast

Differentiable

NeRF



NeRF



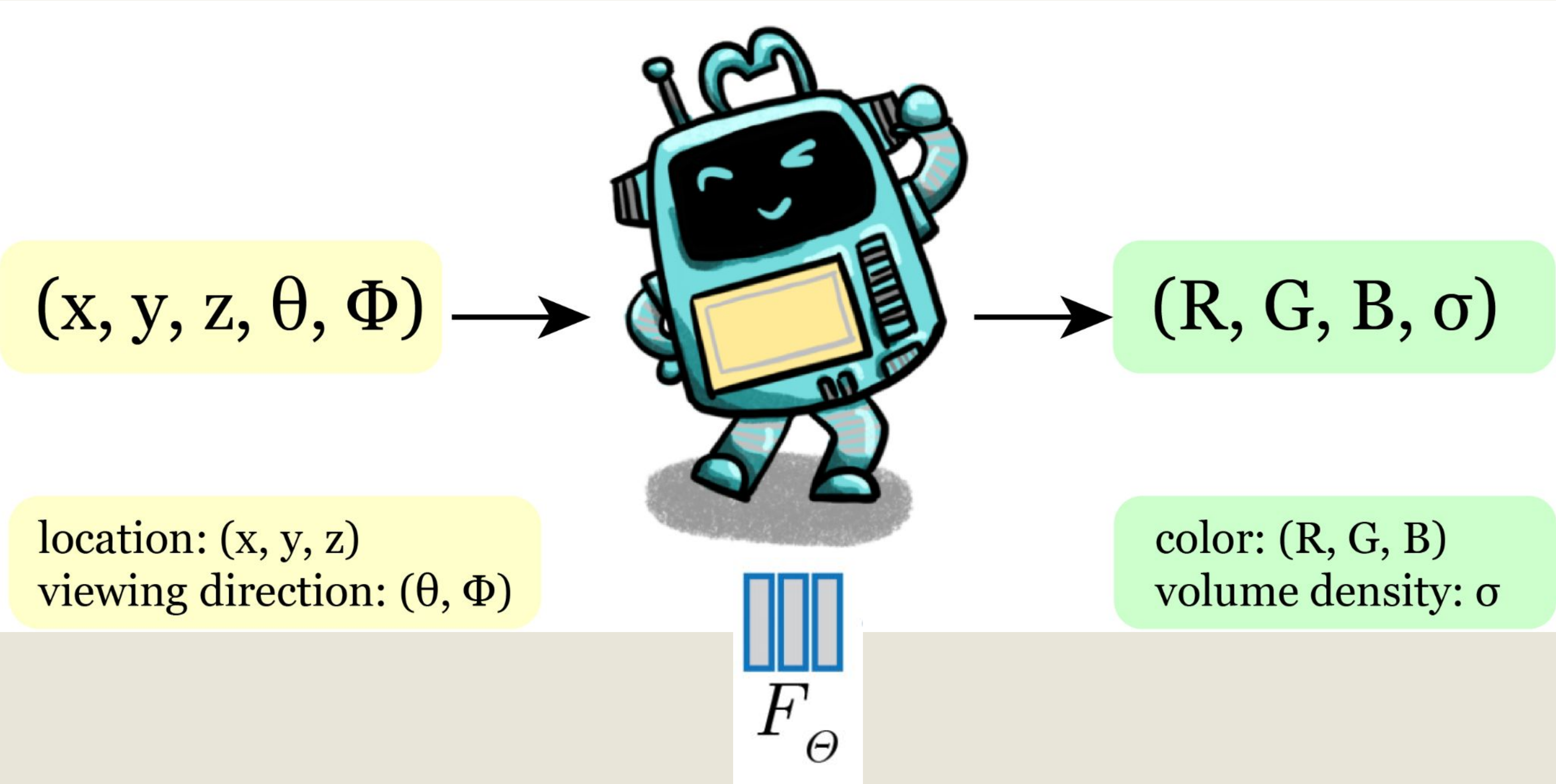
NeRF

Input: Set of images containing the object from different viewpoints

- images must rotate about a fixed origin
- need to know the depth value of pixels
- lighting should remain constant between scenes

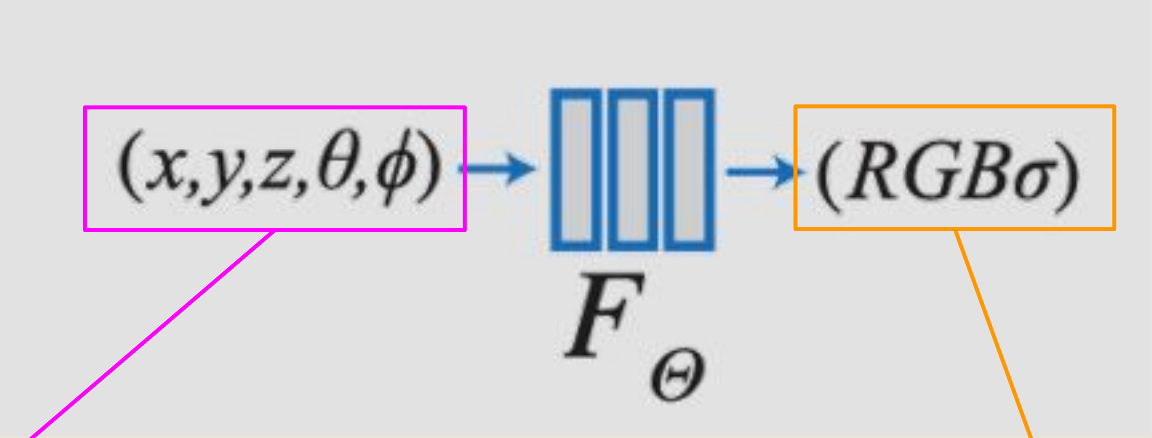
NeRF Training

We optimize a function that can describe the object from every view!



NeRF Training

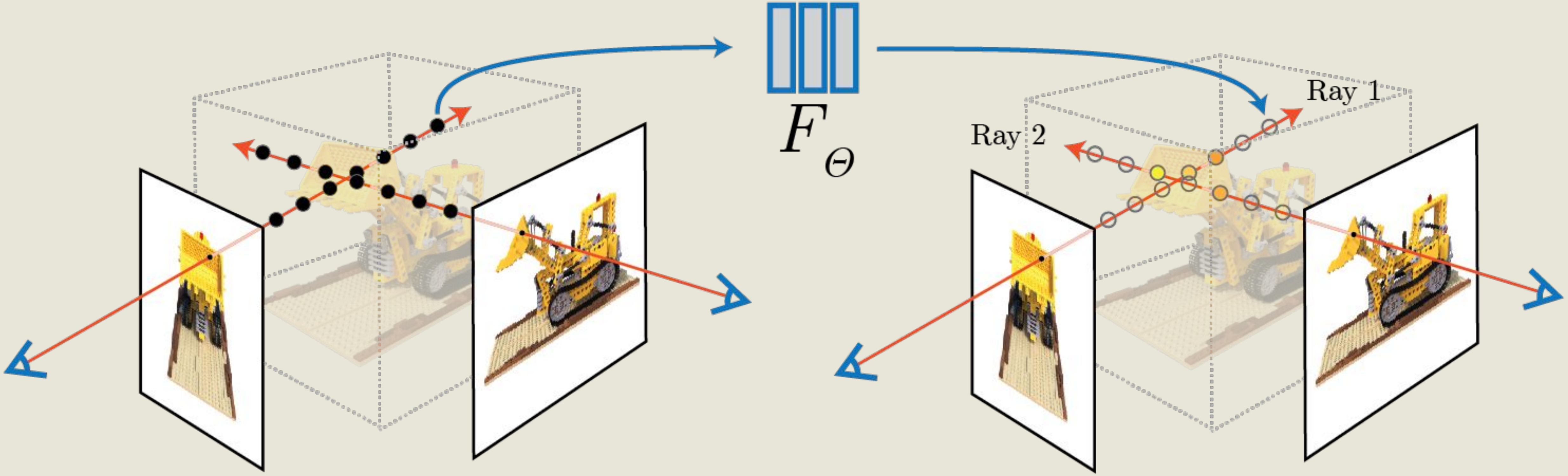
We optimize a function that can describe the object from every view!



location (x, y, z) and
viewing direction (Θ, ϕ)

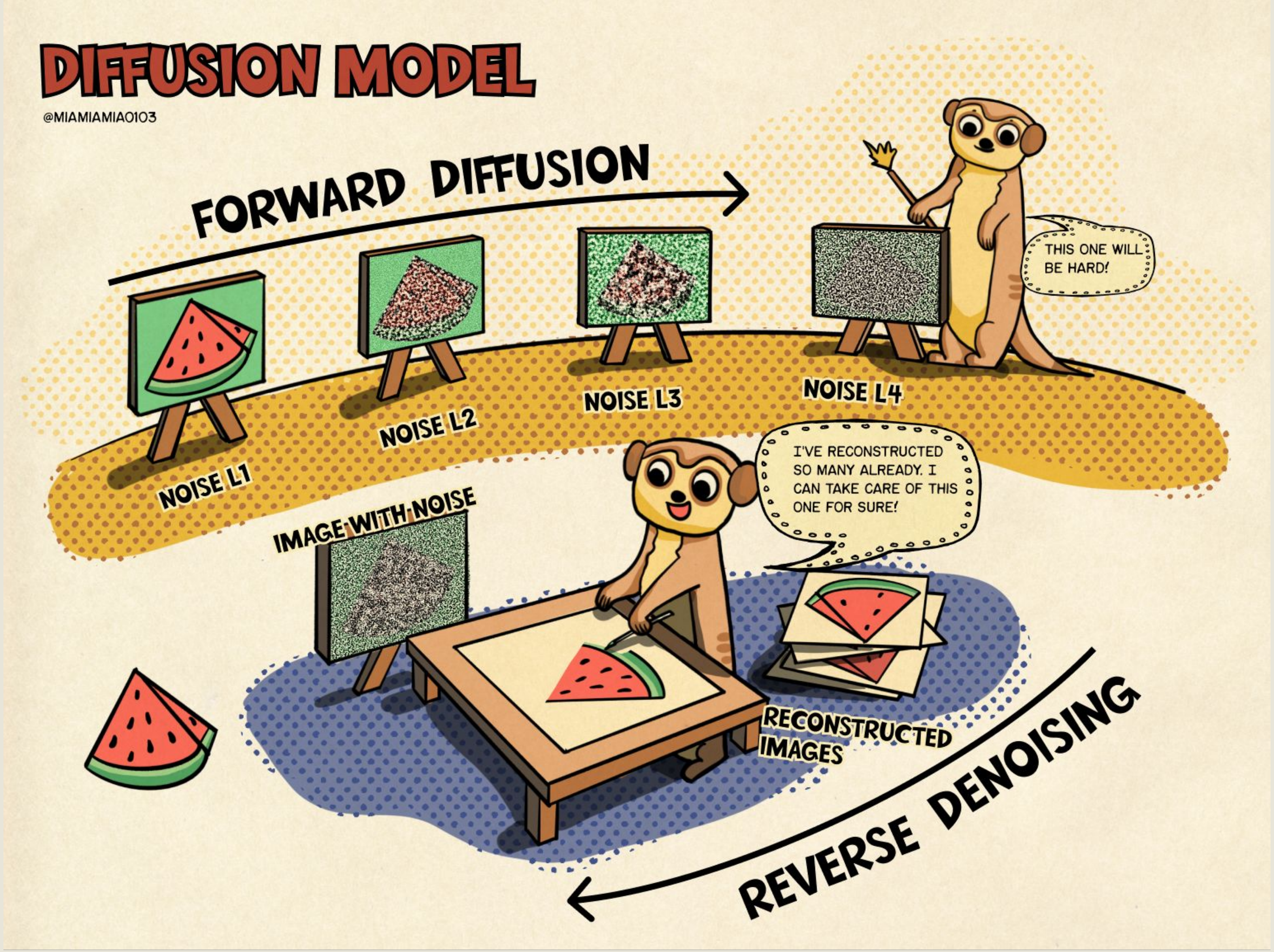
color (R,G,B) and
volume density (σ)

NeRF Training

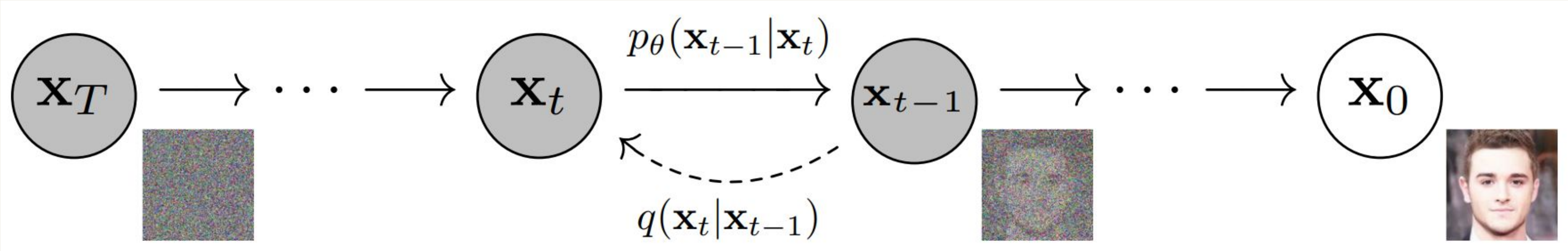


NeRF Training

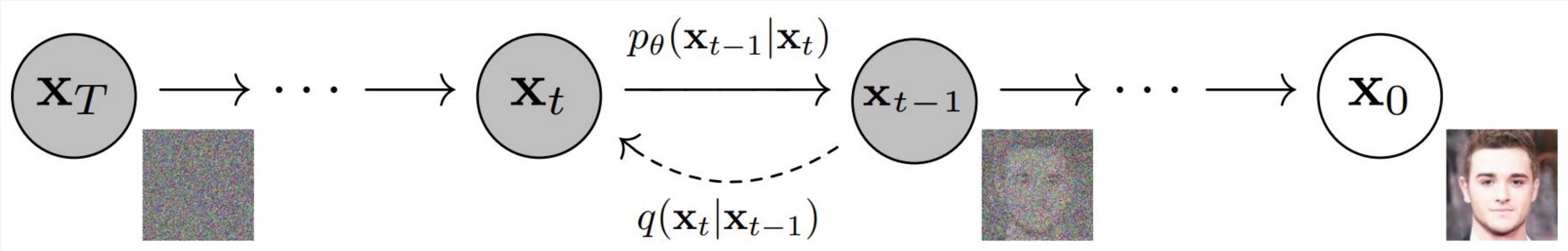
- Overfit - good only for that once scene
- Small Network - only need a small MLP
- Positional Encoding - sin/cos, hash-grid
- Entire process needs to be differentiable


DIFFUSION MODEL
@MIAMIAMIA0103
FORWARD DIFFUSION
THIS ONE WILL BE HARD!
NOISE L1
NOISE L2
NOISE L3
NOISE L4
I'VE RECONSTRUCTED SO MANY ALREADY. I CAN TAKE CARE OF THIS ONE FOR SURE!
IMAGE WITH NOISE
RECONSTRUCTED IMAGES
REVERSE DENOISING

Diffusion Model (GLIDE)



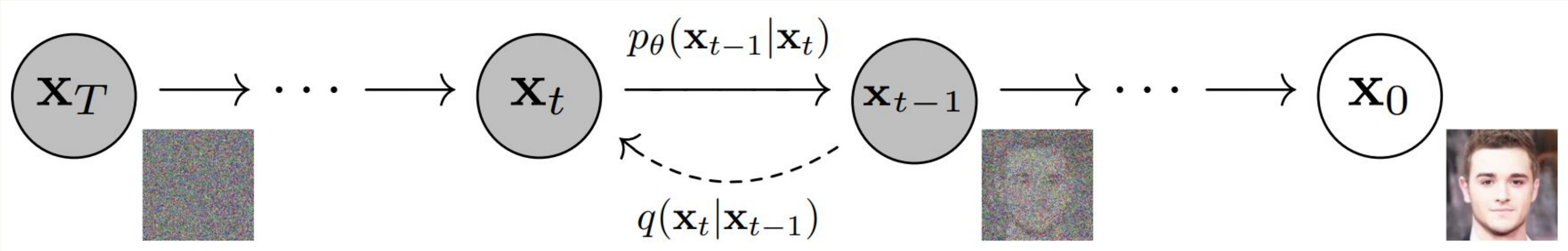
Diffusion - Forward: Image to Noise



$$x_t | x_0 \sim N\left(\sqrt{\bar{\alpha}_t} x_0, (1 - \bar{\alpha}_t) I\right)$$

$$x_{t-1} | x_t, x_0 \sim N(\tilde{\mu}_t(x_t, x_0), \tilde{\beta}_t I)$$

Diffusion - Backward: Noise to Image



$$p_\theta(x_T) = N(x_T|0, I)$$

$$p_\theta(x_{t-1}|x_t) = N(x_{t-1}|\mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$$

Training diffusion models

- We predict the noise added and use MSE loss between predicted noise and the actual noise added to the image

$$\textbf{Model:} \quad \epsilon_\theta\left(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon\right)$$

$$\textbf{Loss:} \quad MSE\left[\epsilon - \epsilon_\theta\left(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon\right)\right]$$

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \mathrm{Uniform}(\{1, \ldots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

DDPM Training Loop

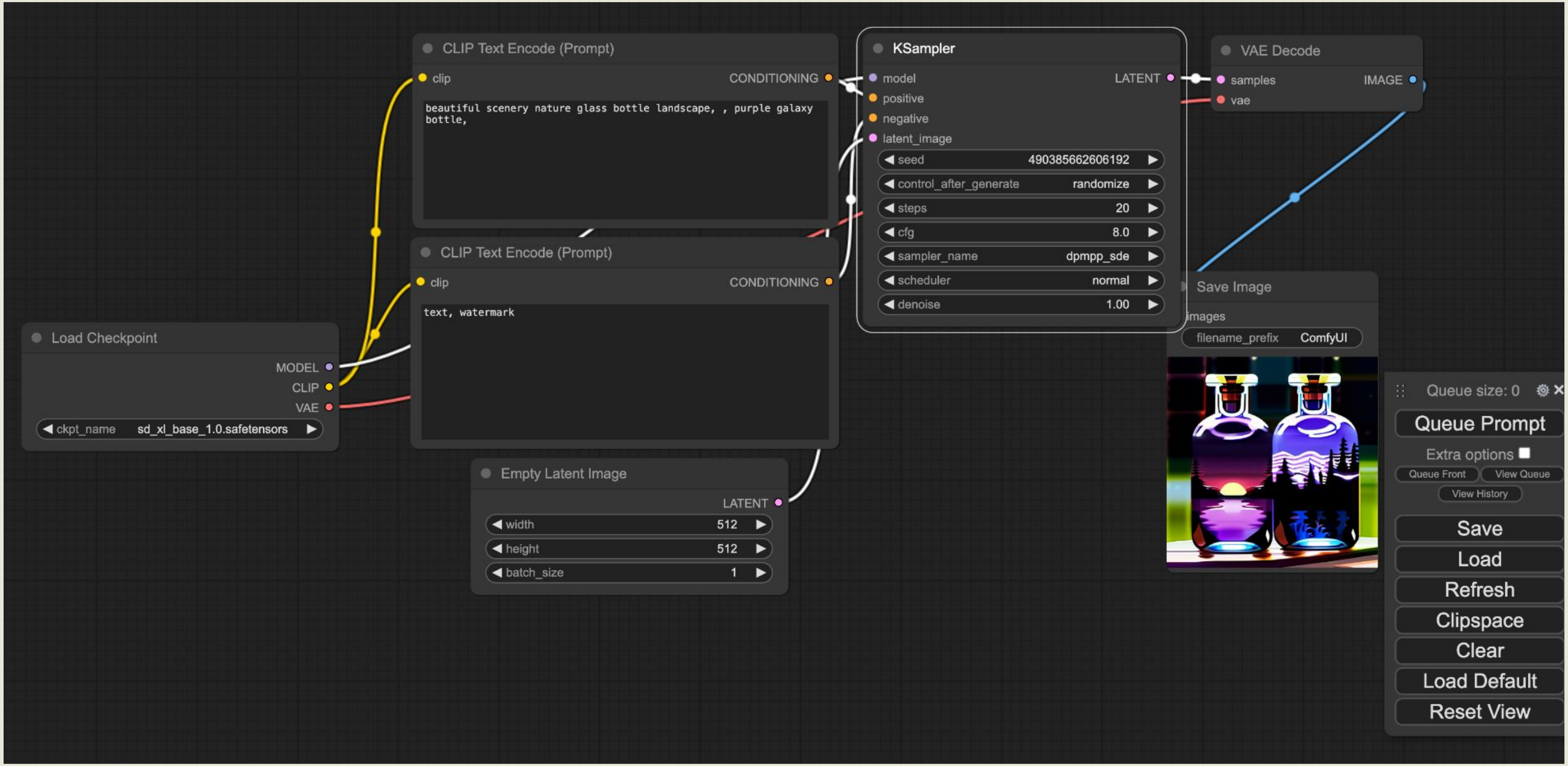
Sampling diffusion models

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\quad \mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

DDPM Sampling Loop

Using diffusion models without code!



comfyUI

DALL.E 2

TEXT DESCRIPTION

An astronaut   Teddy bears   A bowl of soup

riding a horse   lounging in a tropical resort in space   playing basketball with cats in space

in a photorealistic style   in the style of Andy Warhol   as a pencil drawing

→

DALL·E 2

DALL.E 2 (CLIP + GLIDE)

CLIP (Contrastive Language-Image Pre-training):

- Associates text with images
    - Trained on millions of captioned images.

GLIDE (Diffusion Model):

- Generate images

Transformer based TEXT CONDITIONING

- Generate images based on Text Encodings

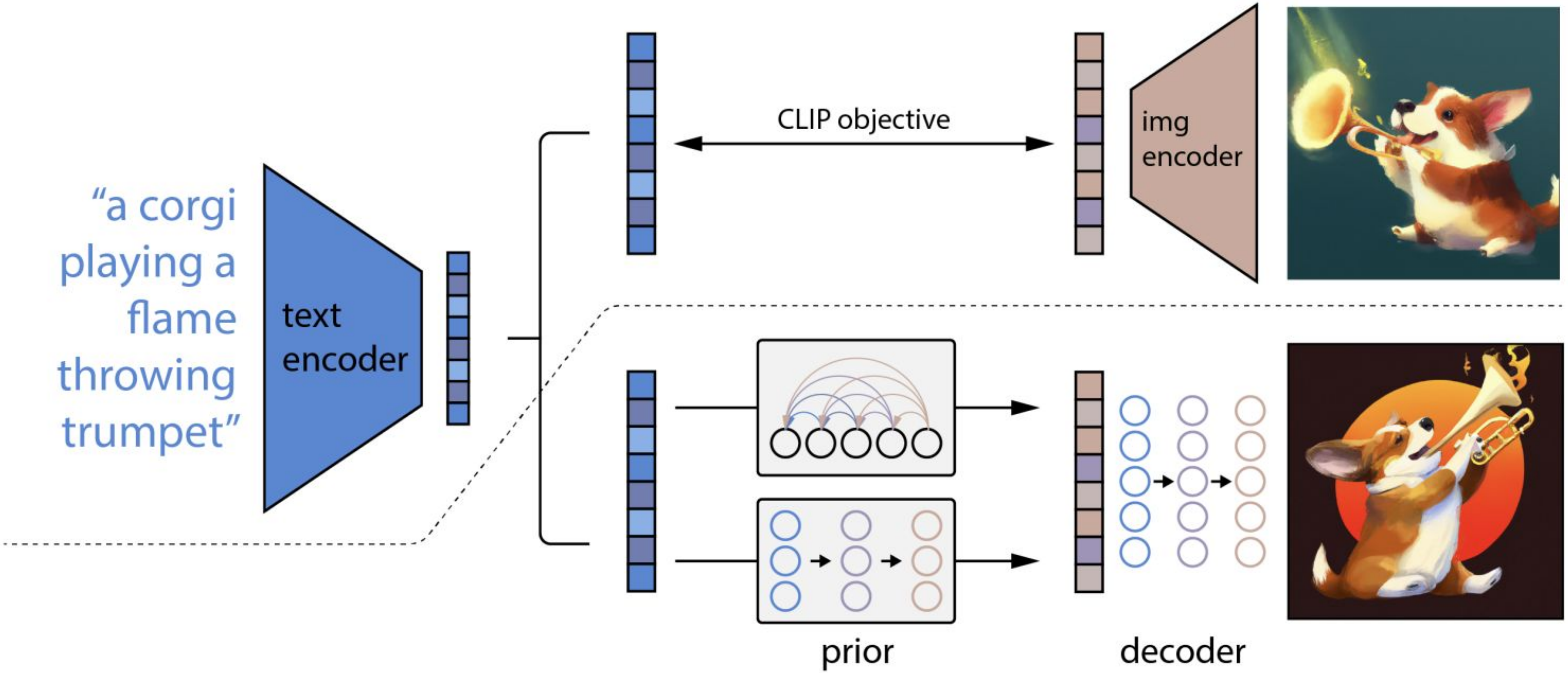
DALL.E 2 (CLIP)

1. Encode Images and Captions

2. Compute cosine similarity of each (image, text) pair

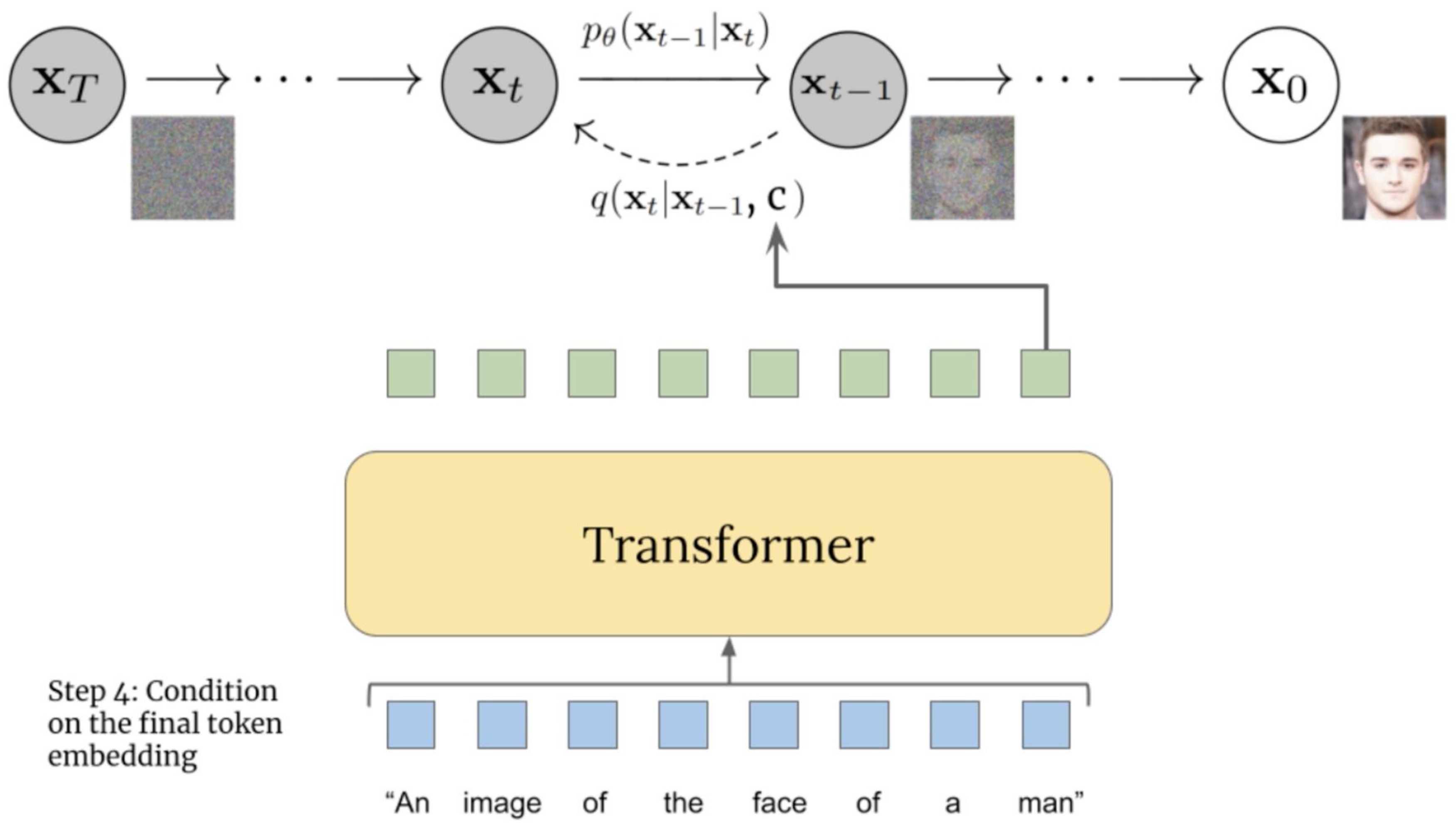
3. Contrastive Loss: Maximize similarity, minimize dissimilarity

DALL.E 2 (CLIP)



"a corgi playing a flame throwing trumpet"
text encoder
CLIP objective
img encoder
prior
decoder

DALL.E 2 (GLIDE)

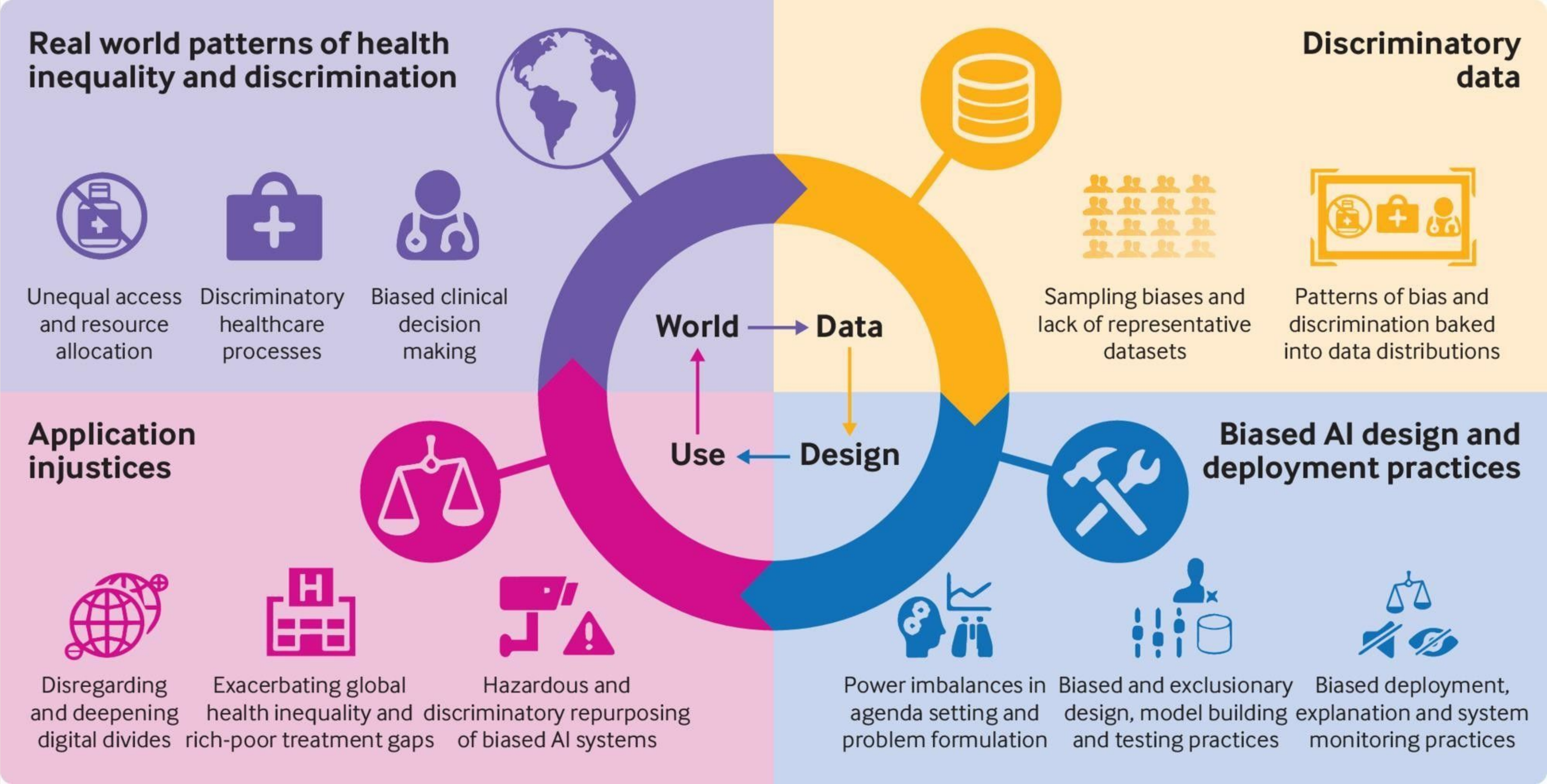


GLIDE training process.

Ethical Issues in AI/ML

-- Bias and Fairness: can perpetuate and amplify biases
-- Privacy and Security: private data, misuse
-- Job Displacement: forced to adapt
-- Responsibility and Accountability: who is responsible
-- Transparency & Explainability: distrust and skepticism
-- Exacerbate inequalities: benefits don’t spread widely

Bias & Fairness



Source: Wold Economic Forum

Summary

-- Basics: data, fully connected, regression, classification
-- Autoencoder: compression, latent space
-- CNN: building block for image-based training
-- Transformer: time series, language, text
-- Unet, resNet: CNN-like with better detail transfer
-- Variational AutoEncoder: Generative:(mean,variance)
-- NeRF, Gaussian Splats: Novel-View Synthesis
-- Diffusion: Generative: noise→sample

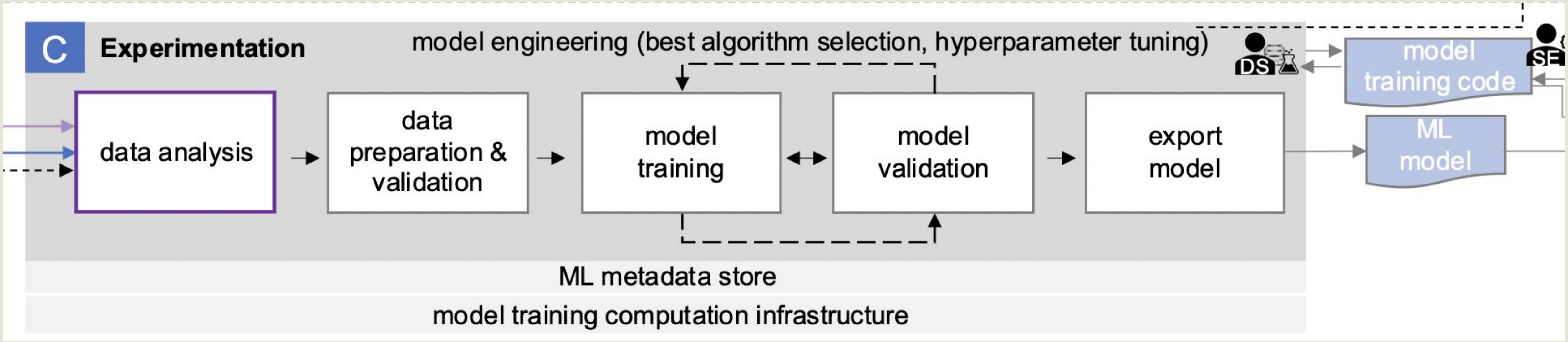
Summary (Practice)

-- Bias vs Variance: Overtraining, Undertraining
-- Data: Lots of it, augmentation, un-biased
-- Hyperparameters: layers, nodes, optimizer, L-Rate
-- Loss function: logits (log-likelihood), L2, L1
-- Training: epochs, batches, tfds, plotting
-- Distributions: find a closely matching distribution
-- Work like a scientist:
Hypothesis, experiment, observe, record, change:
Repeat

Summary (things we did not cover)

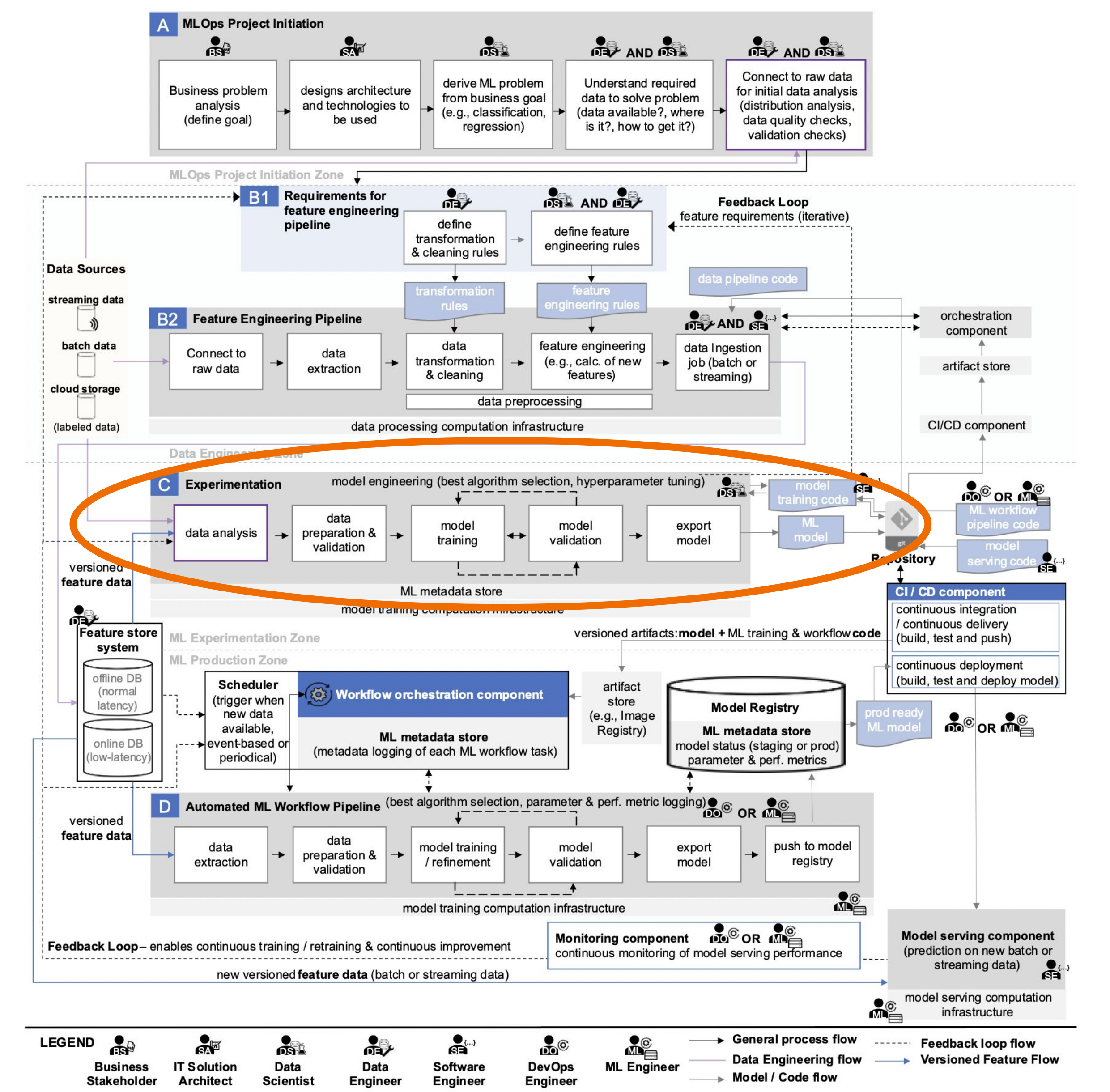
-- Cloud-based: Training and Deployment, ML-ops
-- Local: clusters, machines, environment
-- Tensorboard: for logging, visualizations, checkpoints
-- Intermediate layer visualization
-- Other methods: Random Forests, XGBoost
-- Reinforcement Learning: Agent, State, Env, Reward
-- More theory

What we covered today



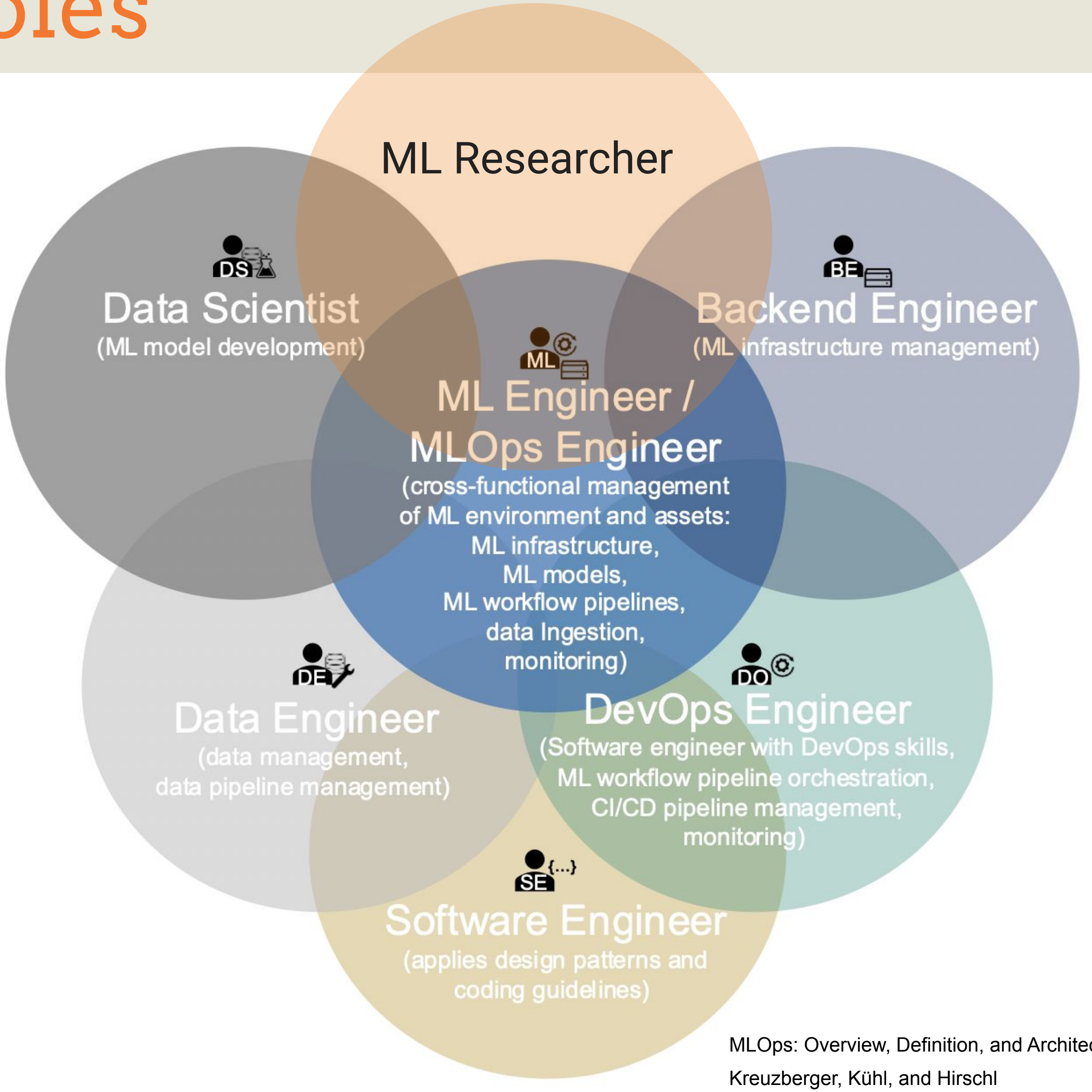
MLOps: Overview, Definition, and Architecture, Kreuzberger, Kühl, and Hirschl

Careers



MLOps: Overview, Definition, and Architecture, Kreuzberger, Kühl, and Hirschl

Jobs and Roles



MLOps: Overview, Definition, and Architecture,
Kreuzberger, Kühl, and Hirschl

Where to go from here:

Deep Learning: https://www.deeplearningbook.org
and some excellent lectures to go along:
Ian Goodfellow, Yoshua Bengio and Aaron Courville

Reinforcement Learning:
Reinforcement Learning: An Introduction ****Complete Draft****

Statistical Inference:
https://link.springer.com/book/10.1007/978-0-387-21736-9

A roadmap to reading:
https://github.com/floodsung/Deep-Learning-Papers-Reading-Roadmap

A More comprehensive list of resources:
https://www.kdnuggets.com/2020/03/24-best-free-books-understand-machine-learning.html

Video Tutorials (3Blue1Brown):
https://www.youtube.com/channel/UCYO_jab_esuFRV4b17AJtAw/videos

IEEE Computer Graphics & Applications

Special Issue on
Generative AI for Computer Graphics

Submission Deadline:
5 September 2024

Publication: March/April 2025

We invite submissions for a special issue on Generative AI for Computer Graphics, aimed at exploring the cutting-edge advancements at the intersection of artificial intelligence and computer graphics. Generative AI techniques have revolutionized the field, enabling unprecedented creativity and realism in generating images, animations, and 3D models. This special issue seeks to showcase the latest research, methodologies, and applications in this rapidly evolving domain.

We welcome original research contributions addressing various aspects of Generative AI for Computer Graphics, including but not limited to image synthesis, style transfer, 3D shape generation, texture synthesis, animation generation, and procedural content creation. Submissions employing novel neural network architectures, advanced optimization techniques, and innovative applications of generative models in computer graphics are particularly encouraged.

Topics of interest include:
- Generative techniques for realistic texture and material generation
- Style transfer techniques for artistic rendering and image manipulation
- Neural rendering methods for synthesizing photorealistic images and animations
- AI assisted sampling, denoising, path-guiding for rendering
- Use of AI for deformation and animation
- Character control and facial animation
- Digital twins, Avatars and other synthetic data generation techniques

Submission link:

https://mc.manuscriptcentral.com/cs-ieee

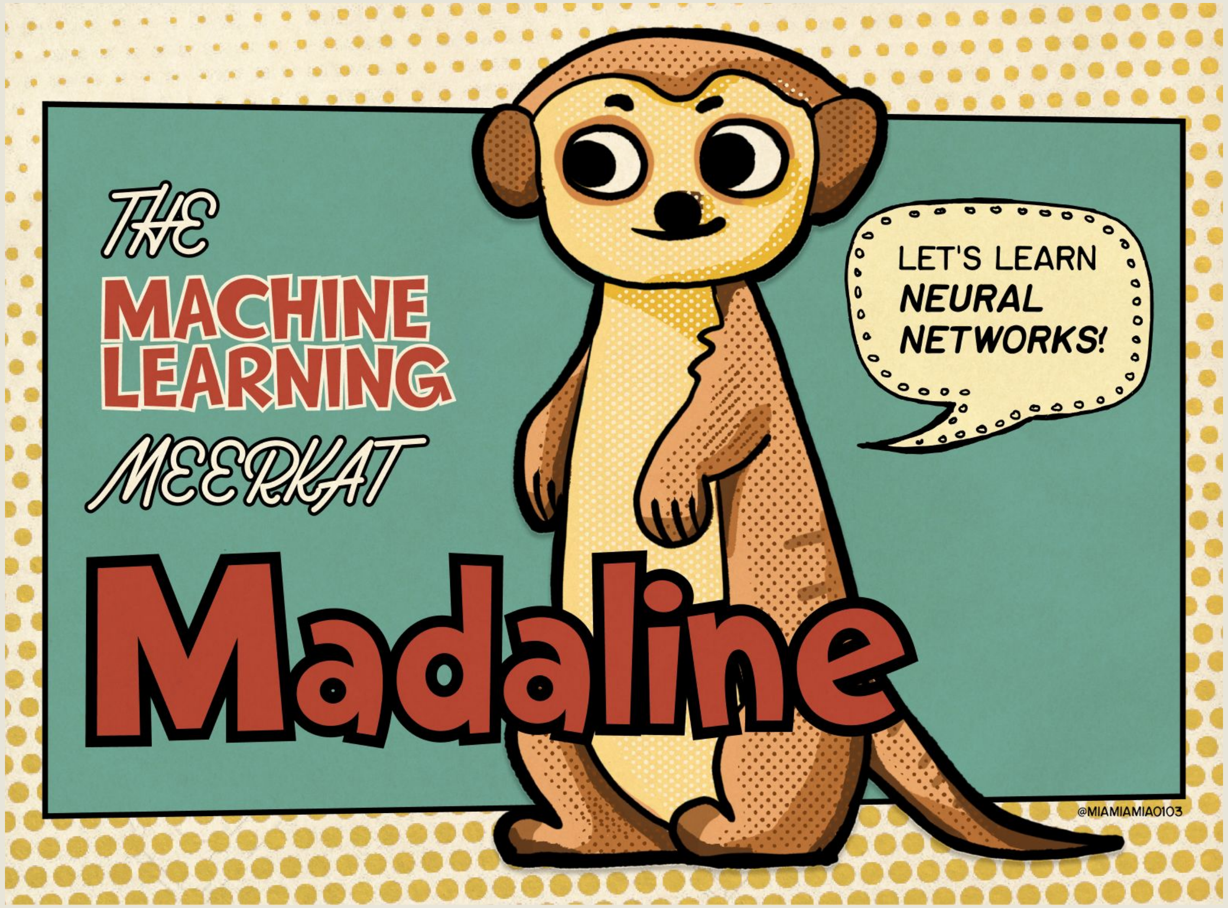
Contact the guest editor at

cga2-2025@computer.org

Guest editors:
Rajesh Sharma (lead guest editor), ETH Zurich, Switzerland
Tomasz Bednarz, NVIDIA, USA
Doug Roble, Meta, USA
Vinicius Azevedo, Disney Research Studios, Zurich, Switzerland

Thank You!

Rajesh Sharma
@xarmalarma



Mia Tang
@miamiamia0103