# Machine learning in science

Dr Niall Miller

nmille39@uwyo.edu / niall.j.miller@gmail.com
niallmiller.com

# What are we going to talk about?

- What is ML?
- What is AI?
- What is Neural Networks?
- What's all the hubbub about?

# What is Machine Learning (ML)?

Use maths to imitate the concept of learning by showing which values under a given frame work can produce a desired result.

- Least squares regression - baby’s first learning machine
- MCMC - …
- Gaussian Processes - covariance machine
- RNN - time series machine
- GPT - generative text thingy
- DDPM - generative pictures
- Scipy.optimize.curve_fit - probably in ~ 95%+ of arxiv papers > 2010
- Automatically fitting “y = mx + c” - what we're going to do!

# How do we teach a computer anything?

Prompt: A old style computer learning via reading a book

We have a value “y”

We have a something that should describe “y” with respect to “x” (F(x))

We don't know what features for F(x) provide a given “y”

We give a computer a random guess for F(x)

We inform the computer how wrong it was (wrong = y - F(x))

The computer attempts F(x) again given knowledge of how wrong it was

We inform the computer how wrong it was (wrong = y - F(x))

The computer attempts F(x) again given knowledge of how wrong it was

We inform the computer how wrong it was (wrong = y - F(x))

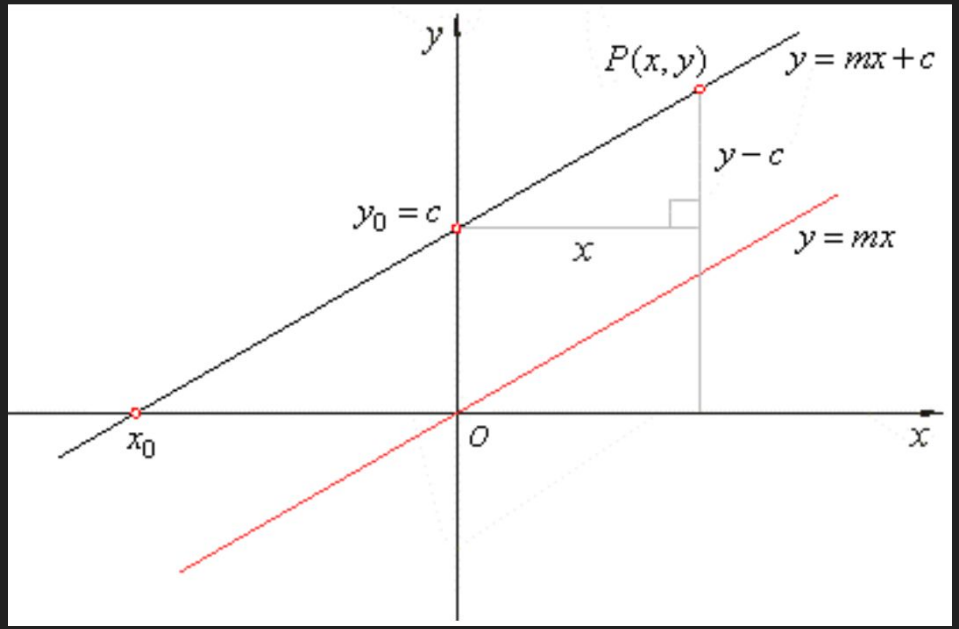
The computer attempts F(x) again given knowledge of how wrong it was

We inform the computer how wrong it was (wrong = y - F(x))

The computer attempts F(x) again given knowledge of how wrong it was

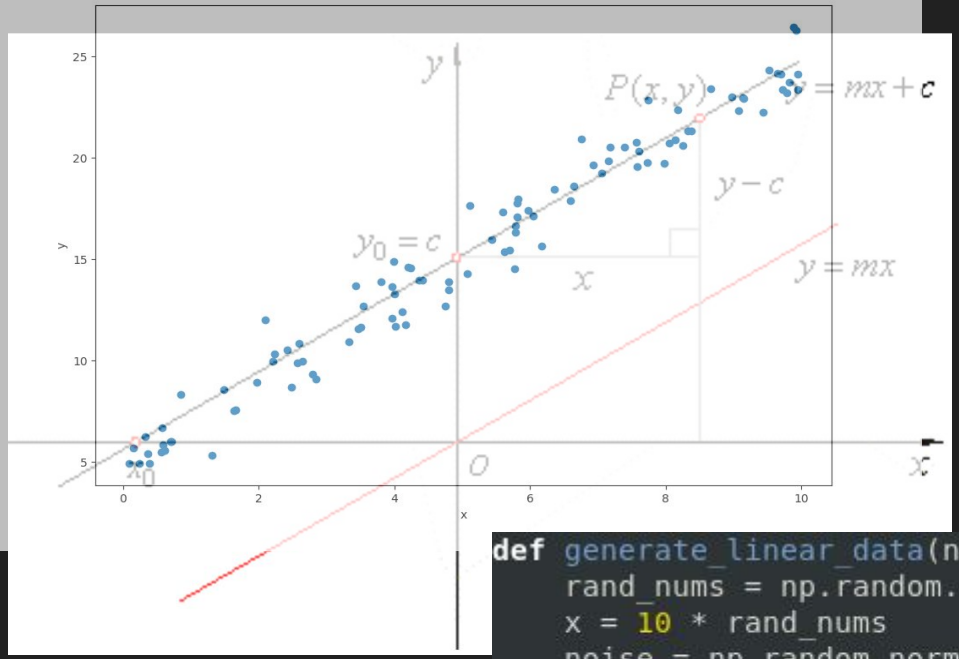
…We hope that eventually $F(x) \approx y$

# Linear Regression Example




Y = mX + c

# Linear Regression Example

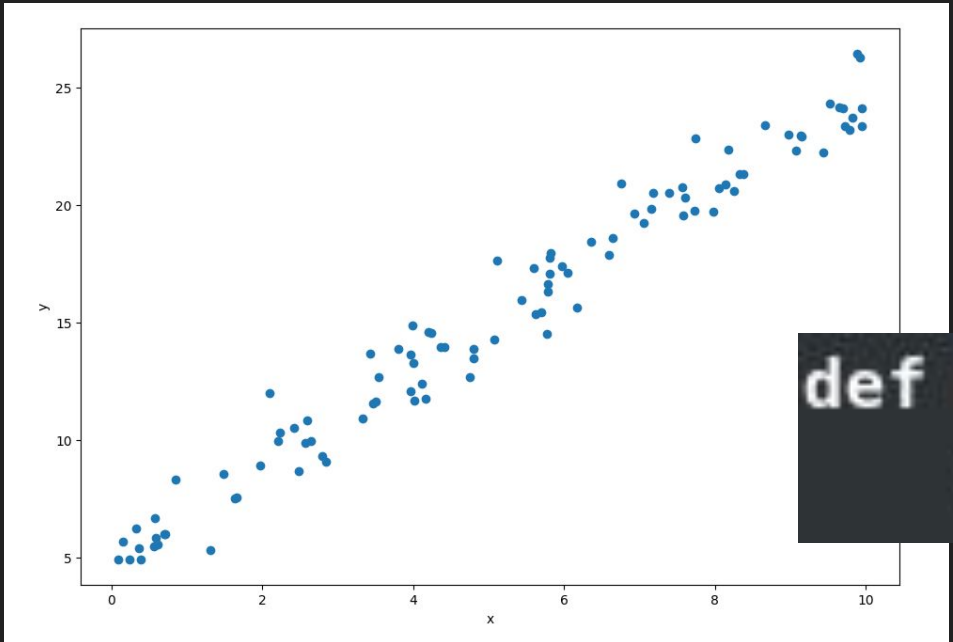



$$Y = mX + c = F(x)$$

Or in code…

```
def generate_linear_data(num_points, gradient, intercept, noise_std_dev):
    rand_nums = np.random.rand(num_points)
    x = 10 * rand_nums
    noise = np.random.normal(0, noise_std_dev, num_points)
    y = linear_eq(x, gradient, intercept) + noise
    return x, y
```

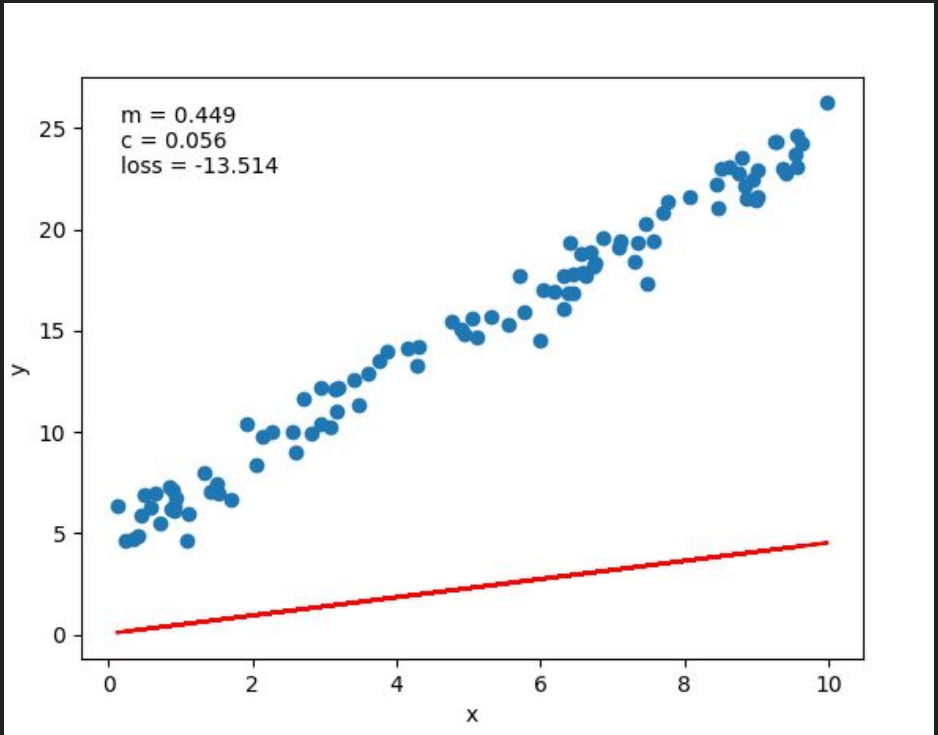
# Linear Regression Example




We can try to get the computer to figure out what combination of ‘m’ and ‘c’ best represents this data

```
def linear_eq(X, m, c):
    return m*X + c
```

# Linear Regression Example

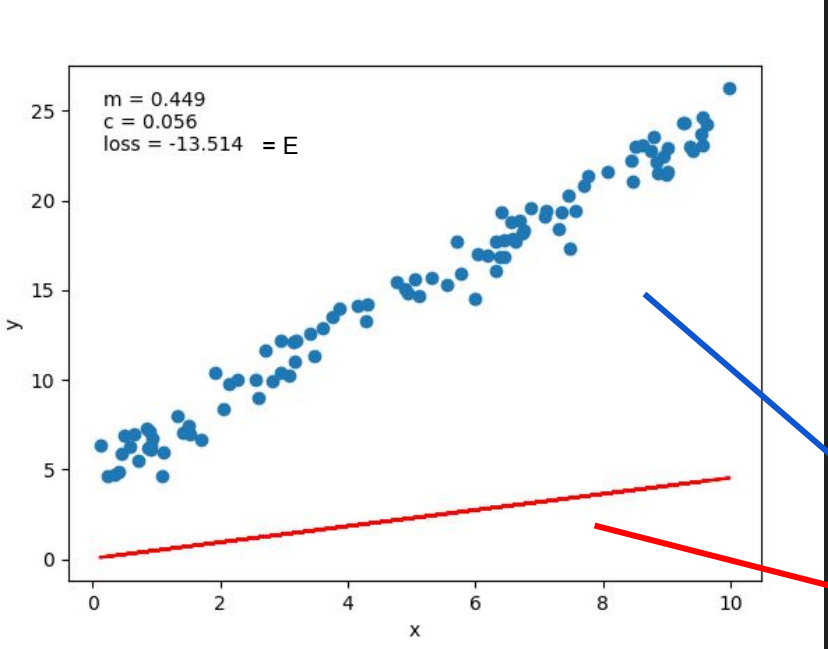



$Y_i = mx_i + c$

$$E = \frac{1}{n}\sum_{i=0}^{n}(y_i - \bar{y}_i)^2$$

$$E = \frac{1}{n}\sum_{i=0}^{n}(y_i - (mx_i + c))^2$$

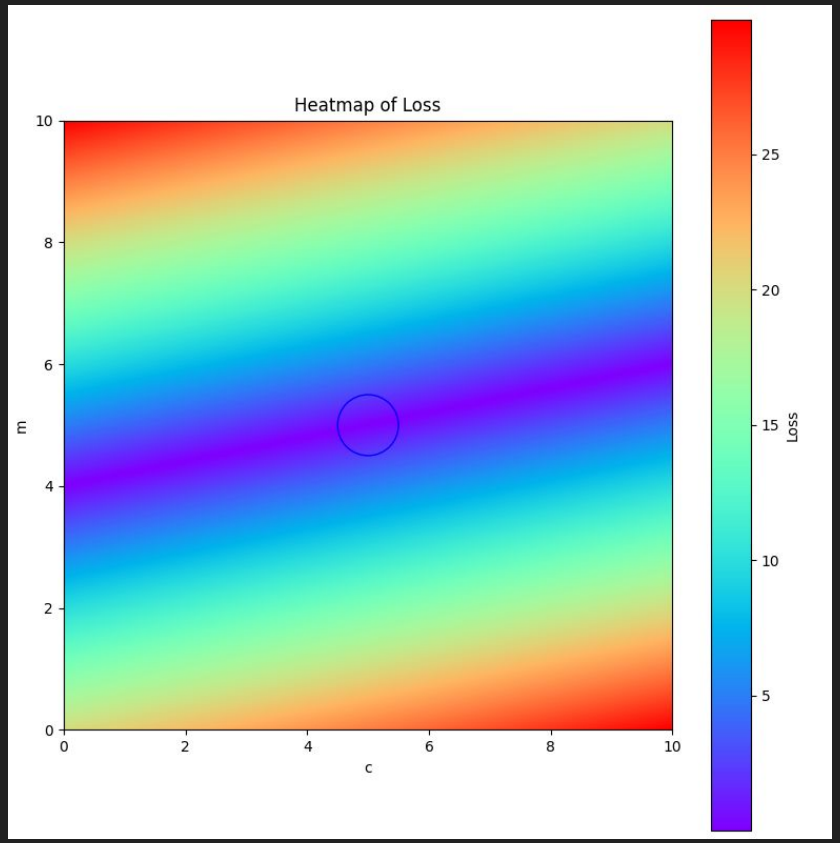
# Linear Regression Example




$$E = \frac{1}{n}\sum_{i=0}^{n}(y_i - \bar{y}_i)^2$$

$$E = \frac{1}{n}\sum_{i=0}^{n}(y_i - (mx_i + c))^2$$

# How do we inform the next step?




$$E = \frac{1}{n}\sum_{i=0}^{n}(y_i - (mx_i + c))^2$$

dE/dm =

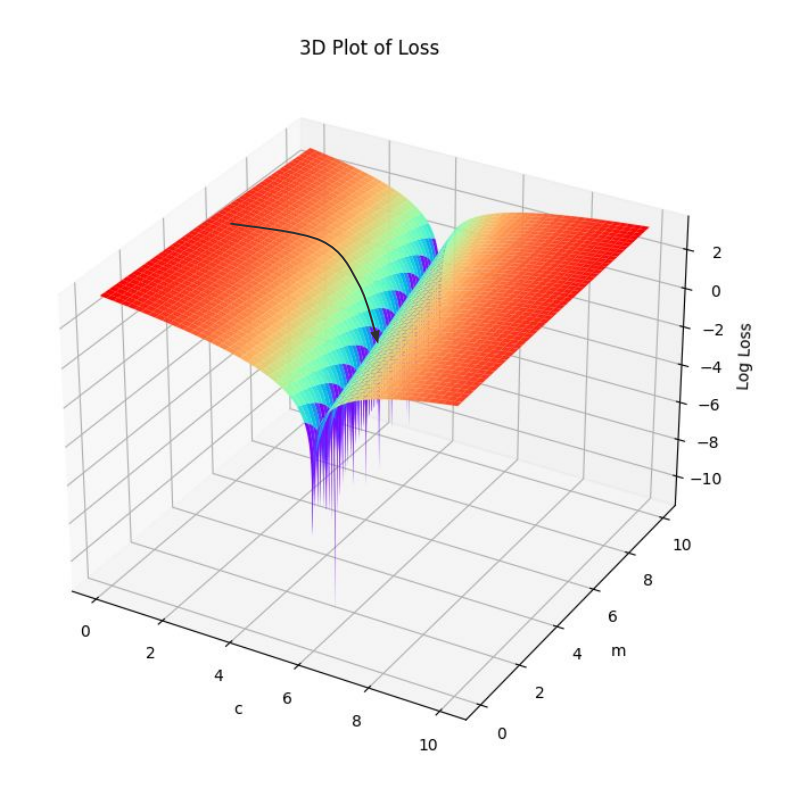
$$D_m = \frac{1}{n}\sum_{i=0}^{n} 2(y_i - (mx_i + c))(-x_i)$$

$$D_m = \frac{-2}{n}\sum_{i=0}^{n} x_i(y_i - \bar{y}_i)$$

dE/dc =

$$D_c = \frac{-2}{n}\sum_{i=0}^{n}(y_i - \bar{y}_i)$$

# How do we inform the next step?




dE/dm =

$$D_m = \frac{1}{n}\sum_{i=0}^{n} 2(y_i - (mx_i + c))(-x_i)$$

dE/dc =

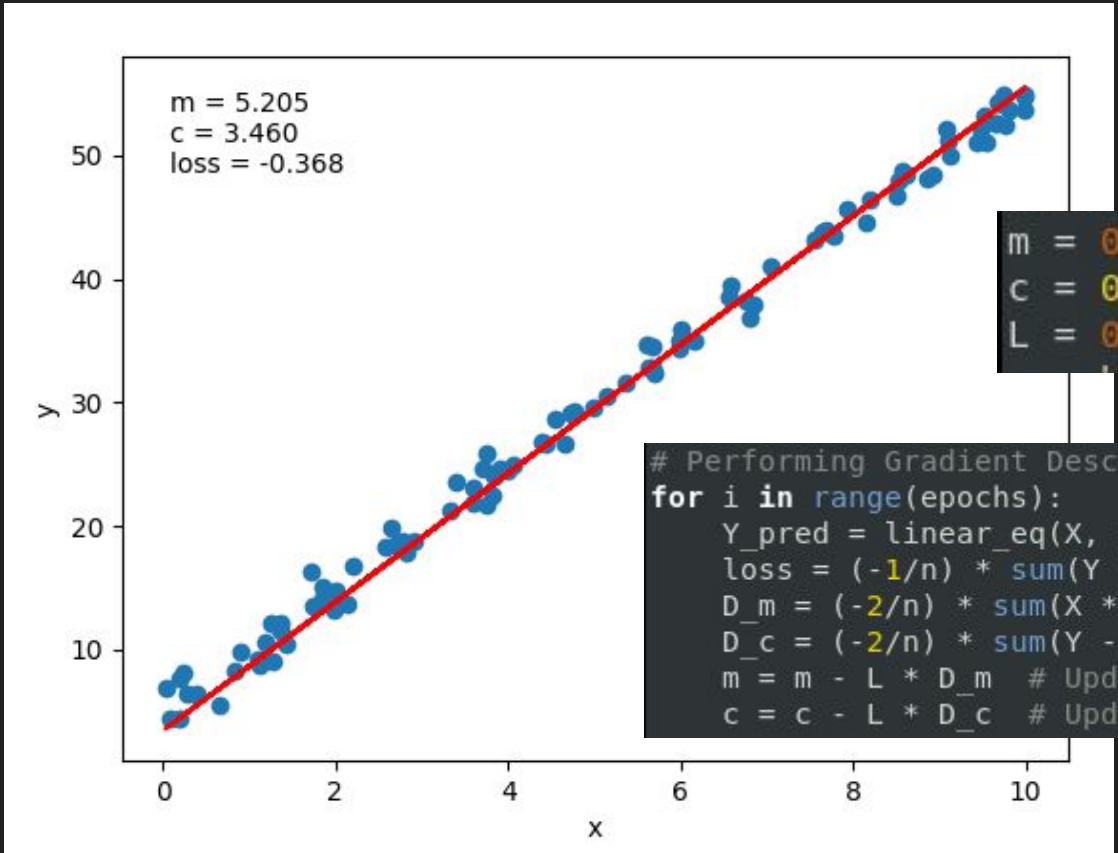
$$D_c = \frac{-2}{n}\sum_{i=0}^{n}(y_i - \bar{y}_i)$$

$$m = m - L \times D_m$$

$$c = c - L \times D_c$$

L = Step size = Learning rate = How much do we update

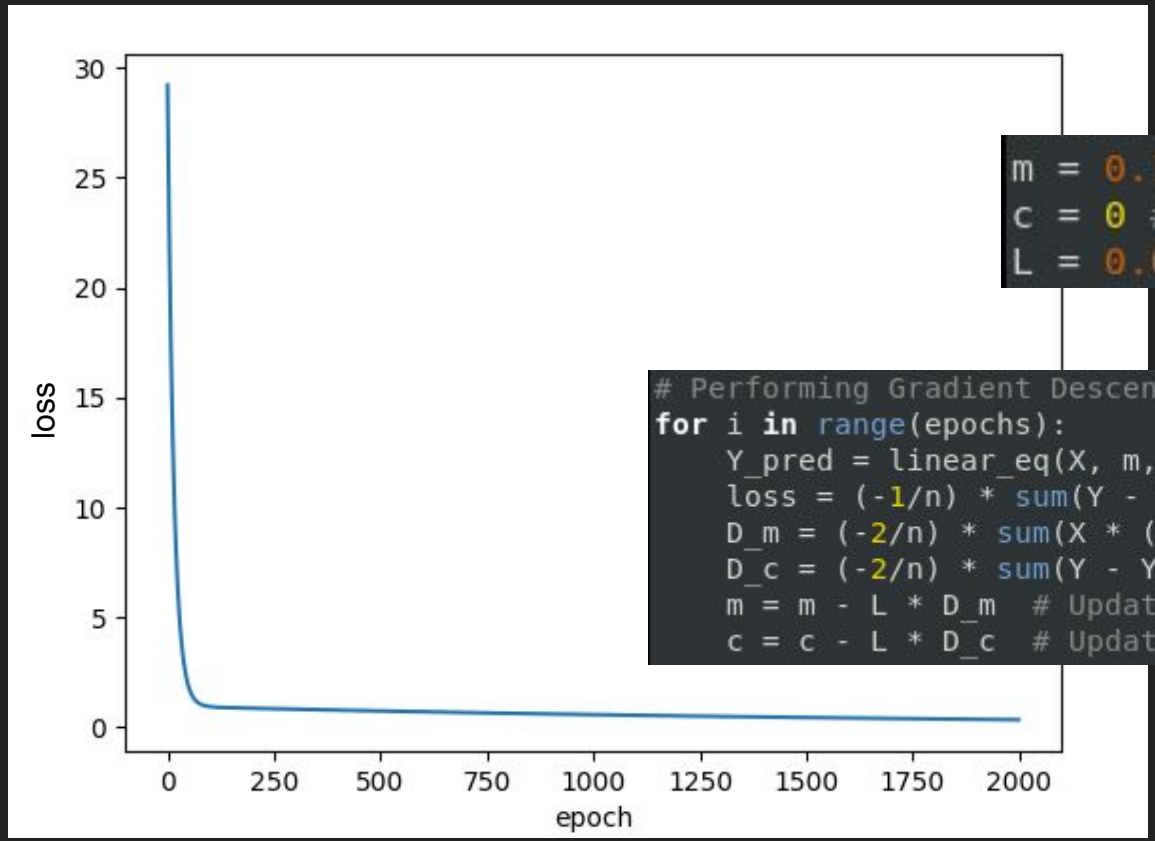
# How do we inform the next step?




```
m = 0.1 #Initial guess for gradient
c = 0 #Initial guess for intercept
L = 0.001  # The learning Rate
```

```
# Performing Gradient Descent
for i in range(epochs):
    Y_pred = linear_eq(X, m, c)  # The current predicted value of Y
    loss = (-1/n) * sum(Y - Y_pred) #MSE loss
    D_m = (-2/n) * sum(X * (Y - Y_pred))  # Derivative wrt m
    D_c = (-2/n) * sum(Y - Y_pred)  # Derivative wrt c
    m = m - L * D_m  # Update m
    c = c - L * D_c  # Update c
```

# How do we inform the next step?



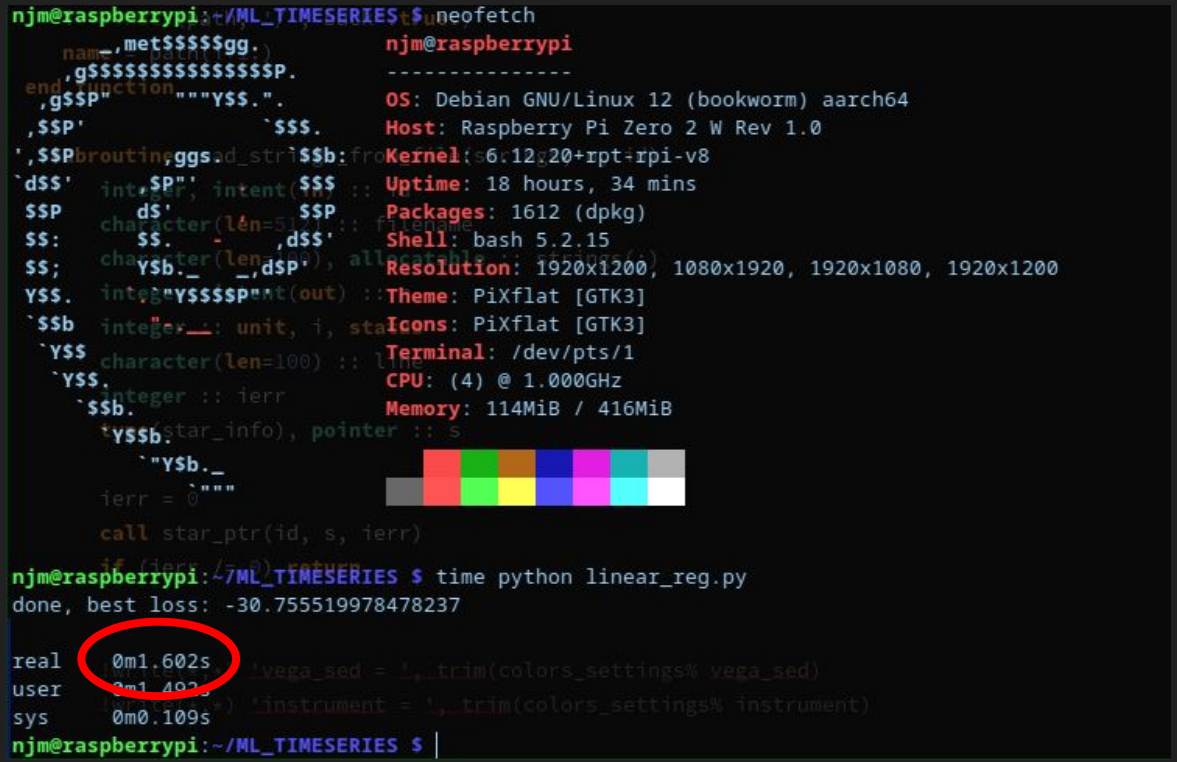

```
m = 0.1 #Initial guess for gradient
c = 0 #Initial guess for intercept
L = 0.001  # The learning Rate
```

```
# Performing Gradient Descent
for i in range(epochs):
    Y_pred = linear_eq(X, m, c)  # The current predicted value of Y
    loss = (-1/n) * sum(Y - Y_pred) #MSE loss
    D_m = (-2/n) * sum(X * (Y - Y_pred))  # Derivative wrt m
    D_c = (-2/n) * sum(Y - Y_pred)  # Derivative wrt c
    m = m - L * D_m  # Update m
    c = c - L * D_c  # Update c
```

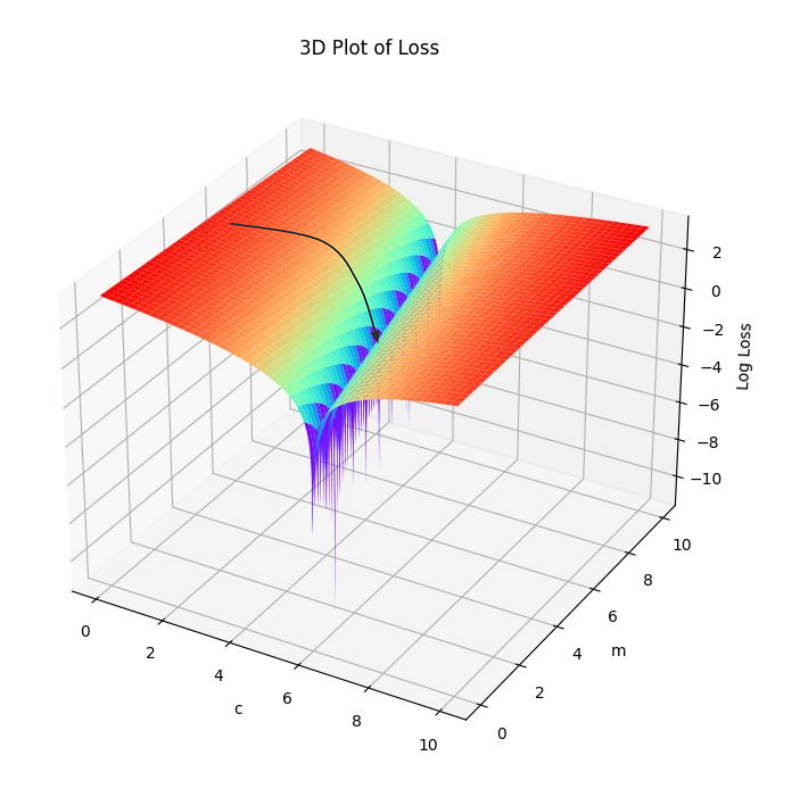
# Required specs and time:

- Computer made after moon landing
- Electricity
- ~0% of medbow

It took ~1.5 seconds to run on a tiny raspberrypi



# How do we inform the next step?




dE/dm =

$$D_m = \frac{1}{n}\sum_{i=0}^{n} 2(y_i - (mx_i + c))(-x_i)$$

dE/dc =

$$D_c = \frac{-2}{n}\sum_{i=0}^{n}(y_i - \bar{y}_i)$$

$$m = m - L \times D_m$$

$$c = c - L \times D_c$$

L = Step size = Learning rate = How much do we update

$$D_c = \frac{-2}{n} \sum_{i=0}^{n} (y_i - \bar{y}_i)$$

$$D_m = \frac{1}{n} \sum_{i=0}^{n} 2(y_i - (mx_i + c))(-x_i)$$


$$E = \frac{1}{n} \sum_{i=0}^{n} (y_i - (mx_i + c))^2$$

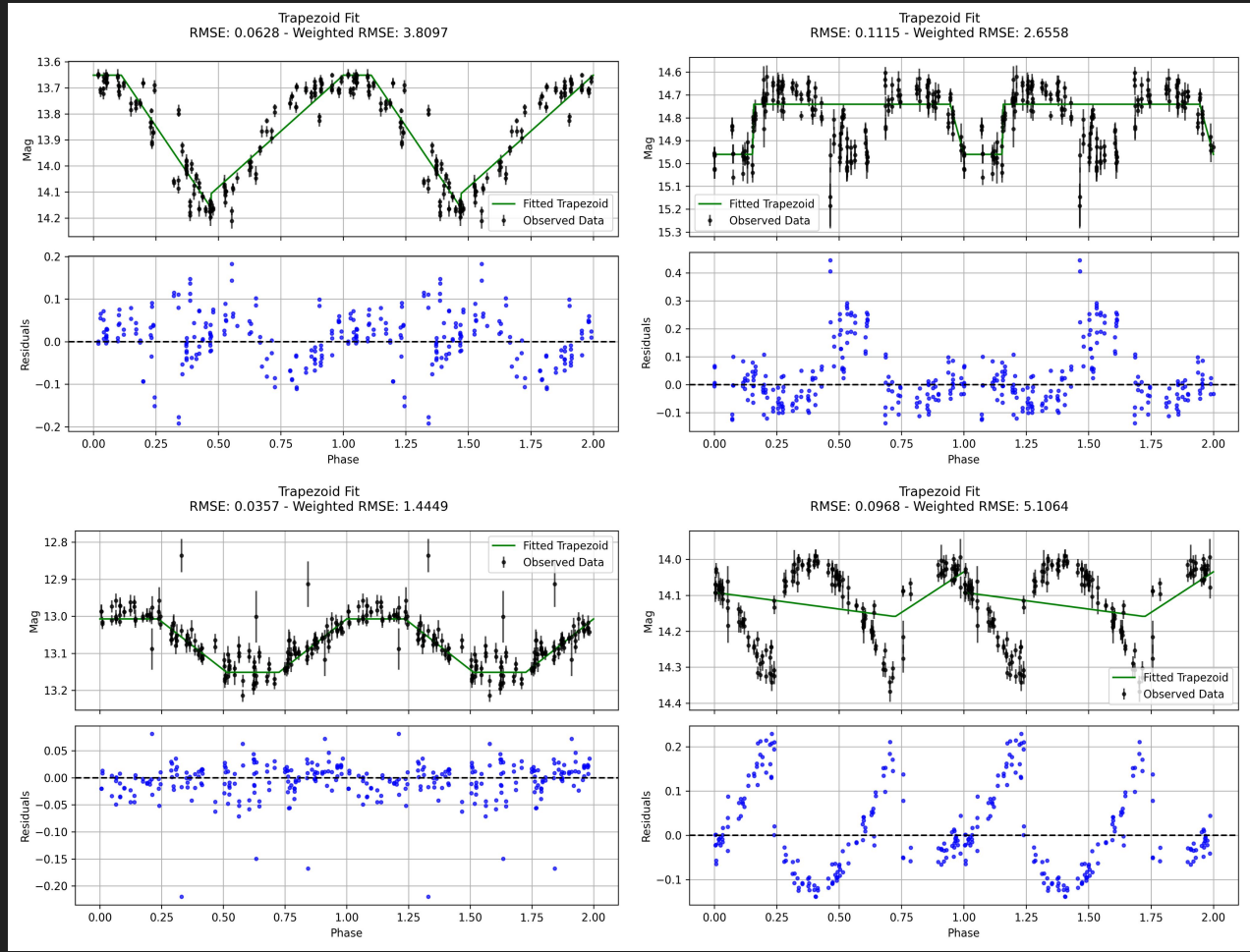
$$m = m - L \times D_m$$

$$c = c - L \times D_c$$

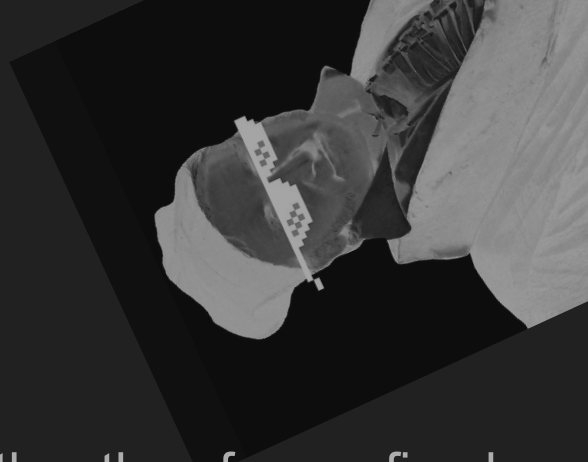
That was nice and easy, but that's mostly because this was a nice and easy problem…..

How about we use this fitting technique (with a trapezoid instead) to try to parameterise and classify stars?

It's pretty clear that this isn't sufficient and I, as a mere human, am unable to think of a reasonable equation/polynomial which would do this well. Nor do I know exactly how the noise behaves.



# Gaussian processes are this but more….

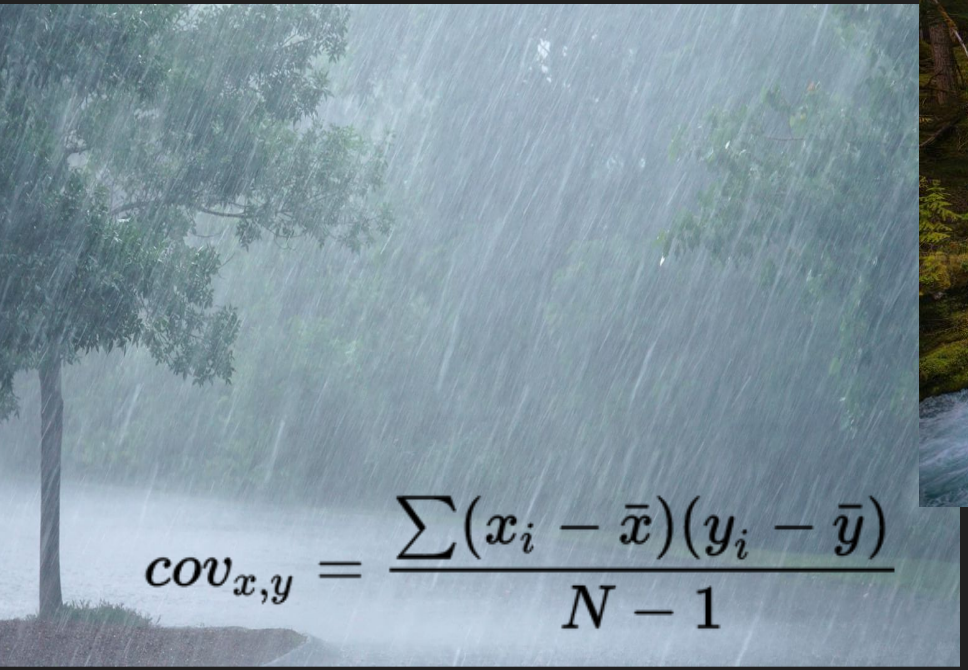


Fundamentally different in 2 key ways

- GPs define a distribution over functions (kernels) rather than form a fixed set of parameters and thus provide us with posterior distributions (uncertainties)
- GPs  fit to the *covariance* of the data and are thus much more flexible as a specific parametric form is not required.

In other words, GPs give us errors and are more flexible at capturing patterns.

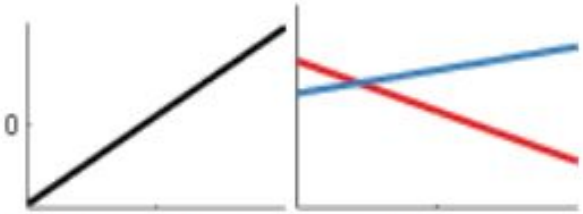
# Covariance - *thing happen and so other thing happen*



$$cov_{x,y} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{N-1}$$

## Linear Kernel

NB this kernel is literally just bayesian linear regression…




$$k_{\text{Lin}}(x, x') = \sigma_b^2 + \sigma_v^2(x - c)(x' - c)$$

# Kernels - https://www.cs.toronto.edu/~duvenaud/cookbook/

## The Kernel Cookbook:

### Advice on Covariance functions

by David Duvenaud

**Update: I've turned this page into a chapter of my thesis.**

If you've ever asked yourself: "How do I choose the covariance function for a Gaussian process?" this is the page for you. Here you'll find concrete advice on how to choose a covariance function for y

If you're looking for software to implement Gaussian process models, I recommend GPML for Matlab, or GPy for Python. These software packages deliberately do not provide a default kernel. You mi don't they include a sensible default?"

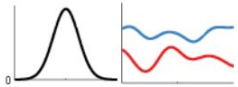
The answer is that the choice of kernel (a.k.a. covariance function) determines almost all the generalization properties of a GP model. You are the expert on your modeling problem - so you're the pe ough about kernels to choose a sensible one, read on.

#### Support Vector Machines

If your question is: "How do I choose a kernel for a Support Vector Machine"? Then a lot of the advice below might still be helpful, especially the first section on standard kernels. However, if you wan ve a hard time learning all the parameters by cross-validation. This is why most SVM kernels have only one or two parameters.

## Standard Kernels

### Squared Exponential Kernel

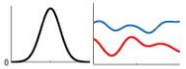


A.K.A. the Radial Basis Function kernel, the Gaussian kernel. It has the form:

$$k_{\mathrm{SE}}(x,x') = \sigma^2 \exp\left(-\frac{(x-x')^2}{2\ell^2}\right)$$


## Standard Kernels

### Squared Exponential Kernel

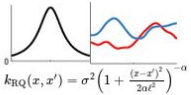

A.K.A. the Radial Basis Function kernel, the Gaussian kernel. It has the form:

$$k_{\mathrm{SE}}(x,x') = \sigma^2 \exp\left(-\frac{(x-x')^2}{2\ell^2}\right)$$


Neil Lawrence says that this kernel should be called the "Exponentiated Quadra

- The lengthscale $\ell$ determines the length of the 'wiggles' in your function. I
- The output variance $\sigma^2$ determines the average distance of your function

### Rational Quadratic Kernel



$$k_{\mathrm{RQ}}(x,x') = \sigma^2\left(1+\frac{(x-x')^2}{2\alpha\ell^2}\right)^{-\alpha}$$

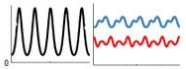
This kernel is equivalent to adding together many SE kernels with different leng

#### Pitfalls of the SE and RQ kernels

Most people who set up a GP regression or classification model end up using th derivatives (for example, the abs() function), then either your lengthscale will er extrapolate in smooth regions if there is even a small non-smooth region in you

If your data is more than two-dimensional, it may be hard to detect this proble

### Periodic Kernel



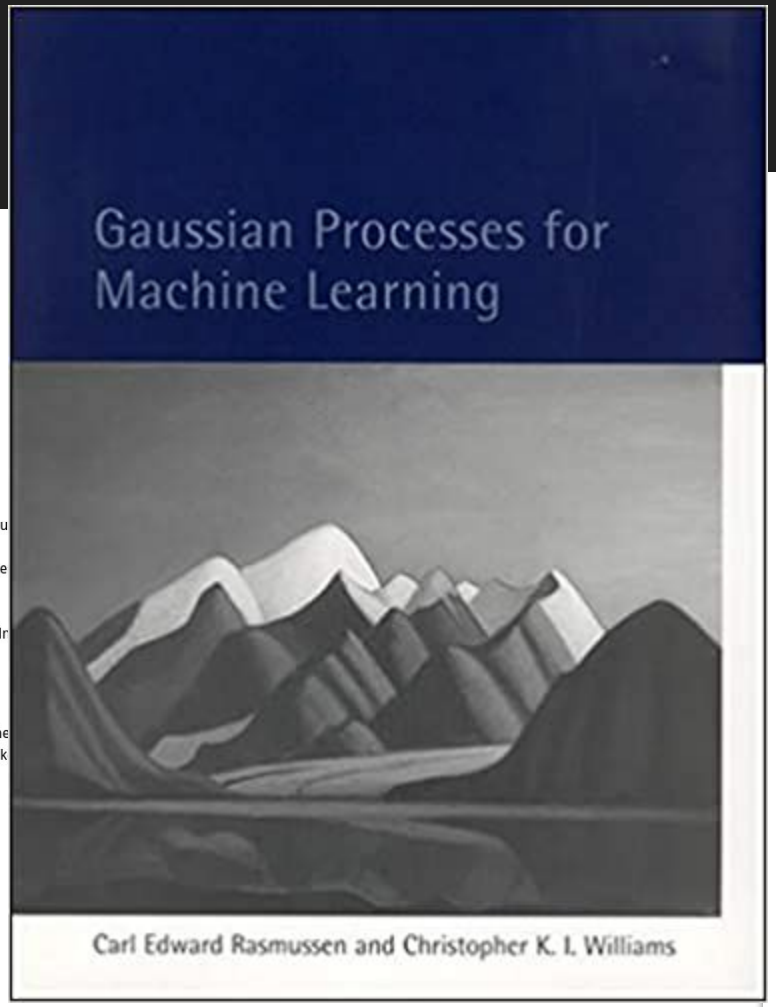
$$k_{\mathrm{Per}}(x,x') = \sigma^2 \exp\left(-\frac{2\sin^2(\pi|x-x'|/p)}{\ell^2}\right)$$

The periodic kernel (derived by David Mackay) allows one to model functions w

- The period $p$ simply determines the distnace between repititions of the fu
- The lengthscale $\ell$ determines the lengthscale function in the same way as

### Locally Periodic Kernel

# Kernels




If you've ever asked yourself: "How do I choose the covariance function for a Gau

If you're looking for software to implement Gaussian process models, I recomme
they include a sensible default?"

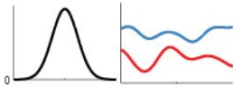
The answer is that the choice of kernel (a.k.a. covariance function) determines alr
about kernels to choose a sensible one, read on.

### Support Vector Machines

If your question is: "How do I choose a kernel for a Support Vector Machine"? The
hard time learning all the parameters by cross-validation. This is why most SVM k

## Standard Kernels

### Squared Exponential Kernel

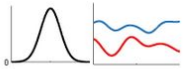


A.K.A. the Radial Basis Function kernel, the Gaussian kernel. It has the form:

$$k_{\mathrm{SE}}(x, x') = \sigma^2 \exp\left(-\frac{(x-x')^2}{2\ell^2}\right)$$

## Standard Kernels

### Squared Exponential Kernel



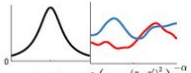
A.K.A. the Radial Basis Function kernel, the Gaussian kernel. It has the form:

$$k_{\mathrm{SE}}(x, x') = \sigma^2 \exp\left(-\frac{(x-x')^2}{2\ell^2}\right)$$

Neil Lawrence says that this kernel should be called the "Exponentiated Quadra

- The lengthscale $\ell$ determines the length of the 'wiggles' in your function. I
- The output variance $\sigma^2$ determines the average distance of your function

### Rational Quadratic Kernel



$$k_{\mathrm{RQ}}(x, x') = \sigma^2 \left(1 + \frac{(x-x')^2}{2\alpha\ell^2}\right)^{-\alpha}$$

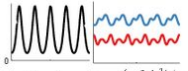
This kernel is equivalent to adding together many SE kernels with different leng

#### Pitfalls of the SE and RQ kernels

Most people who set up a GP regression or classification model end up using th
derivatives (for example, the abs() function), then either your lengthscale will er
extrapolate in smooth regions if there is even a small non-smooth region in you

If your data is more than two-dimensional, it may be hard to detect this proble

### Periodic Kernel



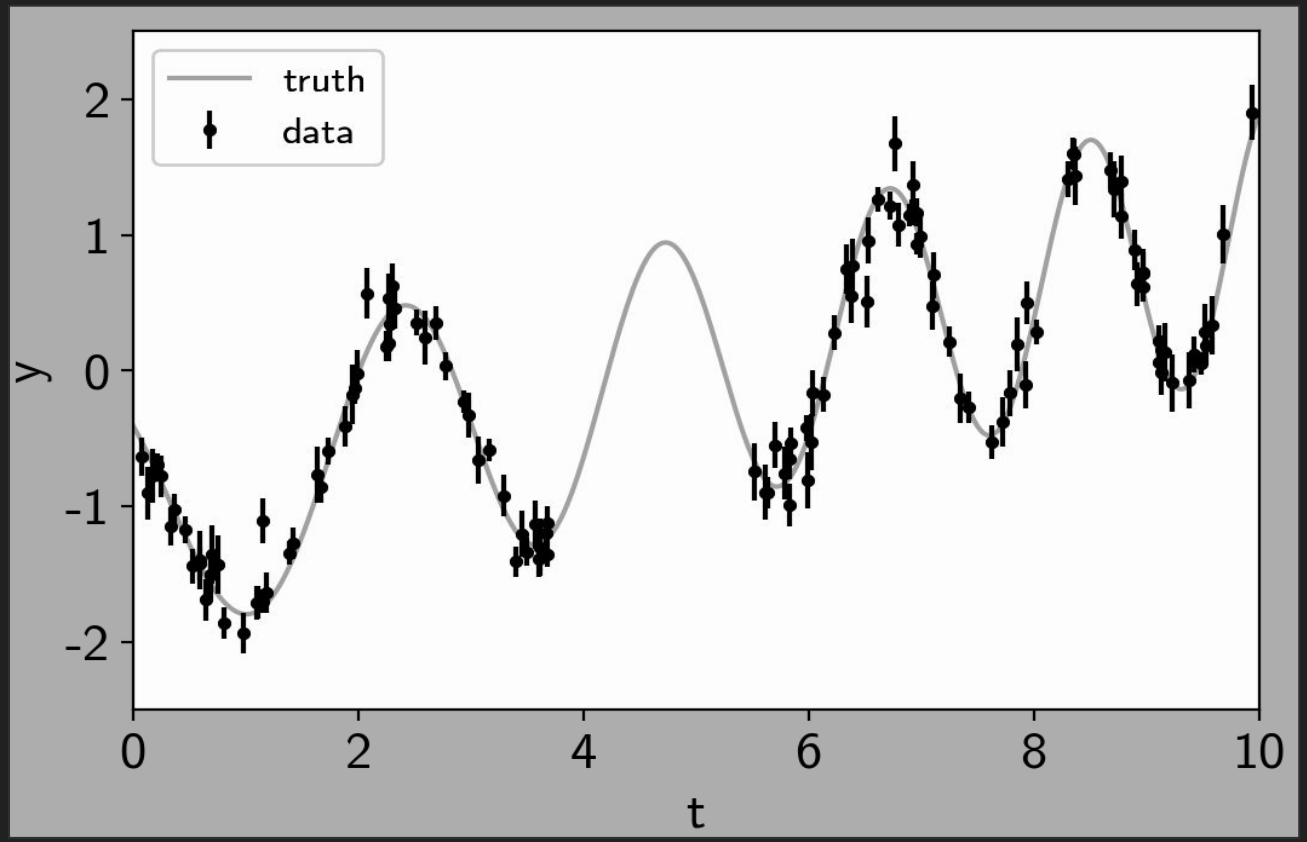
$$k_{\mathrm{Per}}(x, x') = \sigma^2 \exp\left(-\frac{2\sin^2(\pi|x-x'|/p)}{\ell^2}\right)$$

The periodic kernel (derived by David Mackay) allows one to model functions w

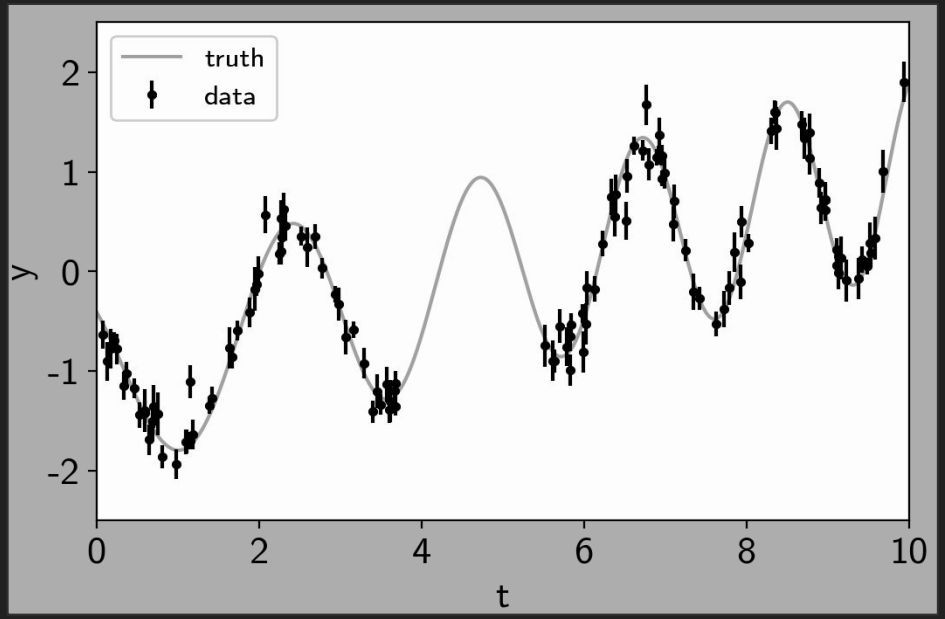
- The period $p$ simply determines the distnace between repititions of the fu
- The lengthscale $\ell$ determines the lengthscale function in the same way as

### Locally Periodic Kernel

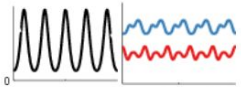
# What can we do with GP?



# What can we do with GP?




**Periodic Kernel**

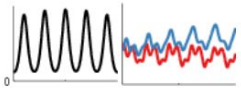


$$k_{\text{Per}}(x, x') = \sigma^2 \exp\left(-\frac{2\sin^2(\pi|x-x'|/p)}{\ell^2}\right)$$

The periodic kernel (derived by David Mackay) allows one to model functions which repeat themselves ex

- The period $p$ simply determines the distnace between repititions of the function.
- The lengthscale $\ell$ determines the lengthscale function in the same way as in the SE kernel.

**Locally Periodic Kernel**



A SE kernel times a periodic results in functions which are periodic, but which can slowly vary over time.

$$k_{\text{LocalPer}}(x, x') = k_{\text{Per}}(x, x') k_{\text{SE}}(x, x') = \sigma^2 \exp\left(-\frac{2\sin^2(\pi|x-x'|/p)}{\ell^2}\right) \exp\left(-\frac{(x-x')^2}{2\ell^2}\right)$$

Most periodic functions don't repeat themselves exactly. To add some flexibility to our model, we can cor
shape of the repeating part of the function can now change over time.

SHO kernel:

$$S(\omega) = \sqrt{\frac{2}{\pi}} \frac{S_1 {\omega_1}^4}{(\omega^2 - {\omega_1}^2)^2 + 2\,{\omega_1}^2\,\omega^2} + \sqrt{\frac{2}{\pi}} \frac{S_2 {\omega_2}^4}{(\omega^2 - {\omega_2}^2)^2 + {\omega_2}^2\,\omega^2/Q^2}$$


ぎっとはぶ
GitHub

# How is Deep Learning different from classical methods?

Deep Learning

-Linearity not required

-Covariance of data not necessary apriori

-Scalable to computational limits

-Black box

-Trained

Classical

-Knowledge of linearity required

-Knowledge of covariance required (i.e an equation)

-Easy to understand

-Only data of interest required

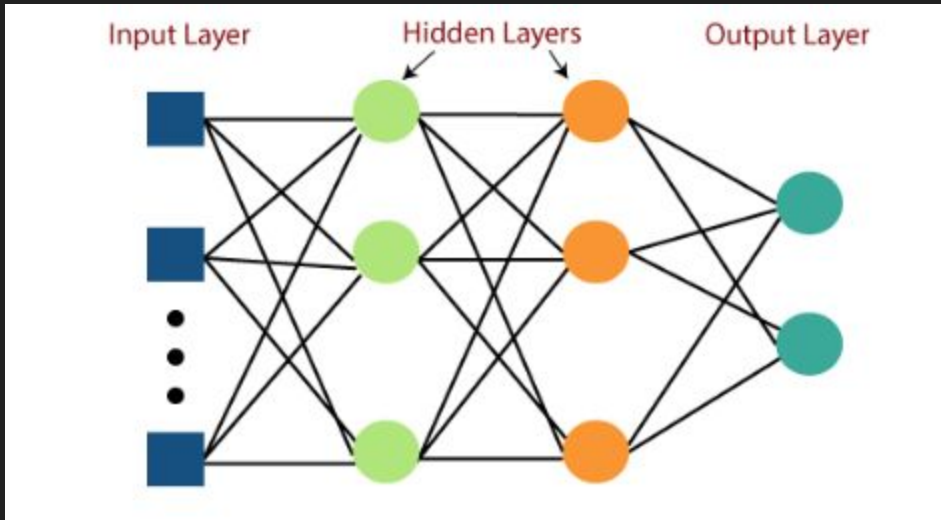
# Deep learning is just fitting without the structure of a fitted EQ -

We do the exact same task as classical methods

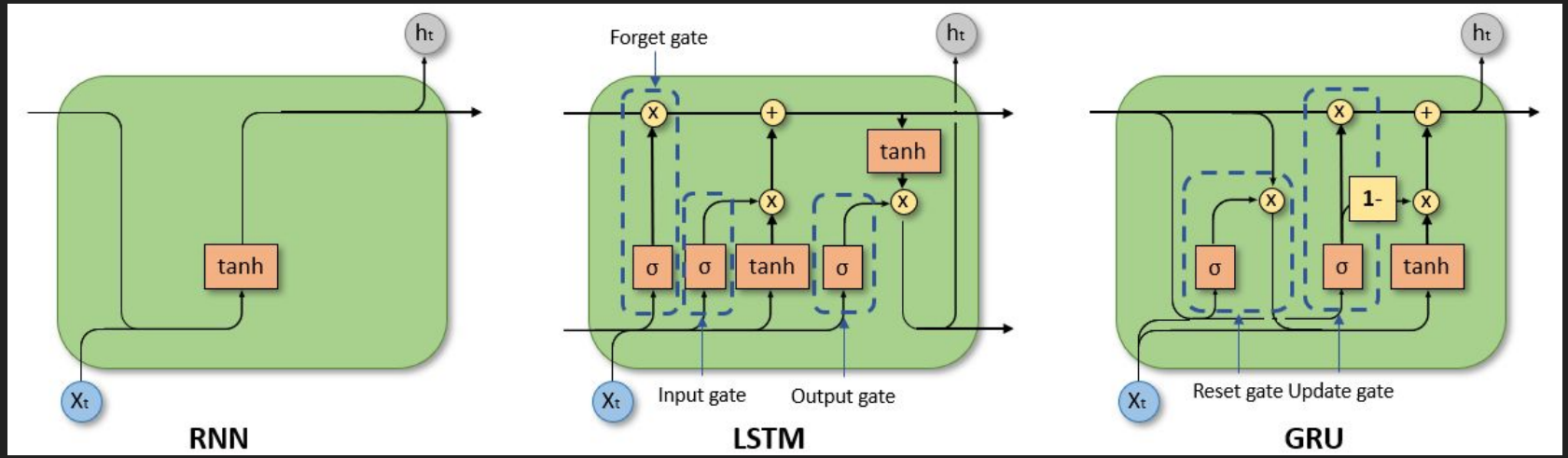
Instead of fitting to F(x) we create a system which acts as a generalisation for all possible F(x).  - *Some smushy thing that is far less rigid than a simple equation.*

AKA

Universal Function Approximators

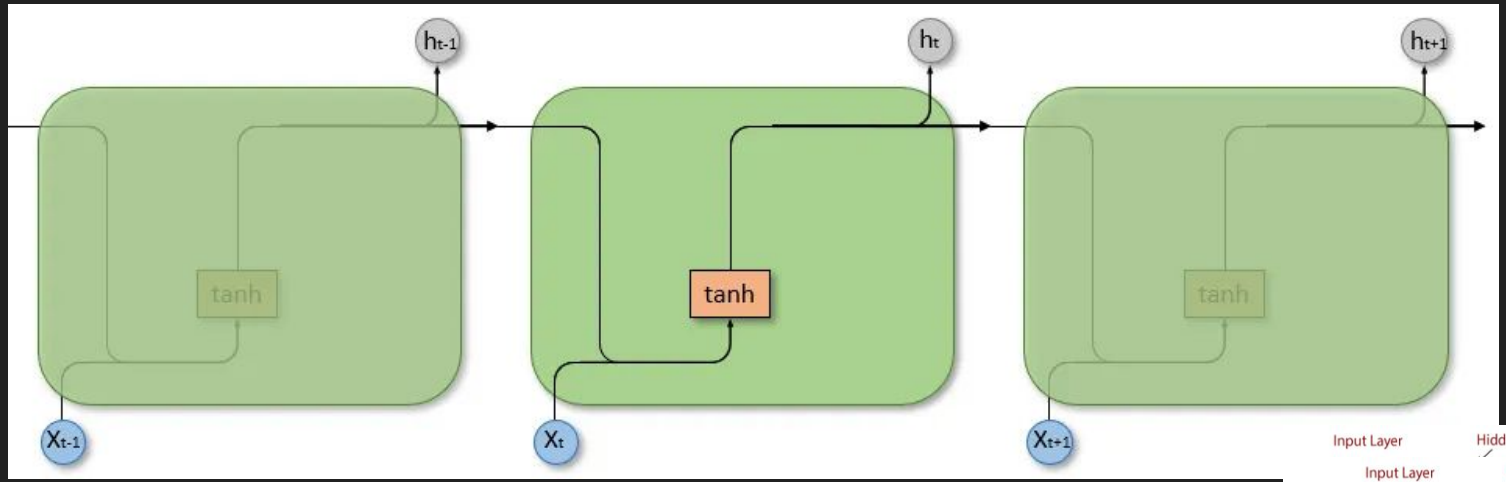


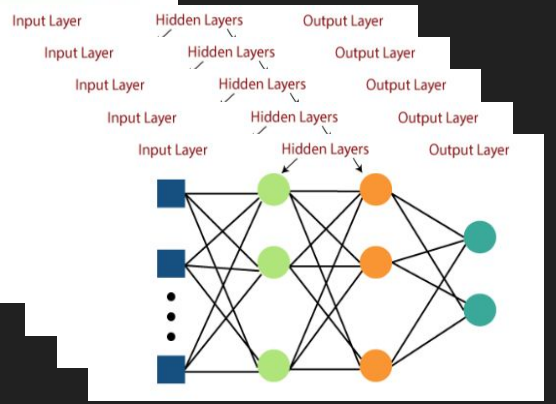
# Recurrent Neural Networks – What if an MLP could ‘member?




Recurrent Neural Network (RNN)
Long Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)

# RNN = ~~Rolled up~~ Recurrent Neural Network




It's the recurrence of a standard neural network – an MLP

# RNN = ~~Rolled up~~ Recurrent Neural Network

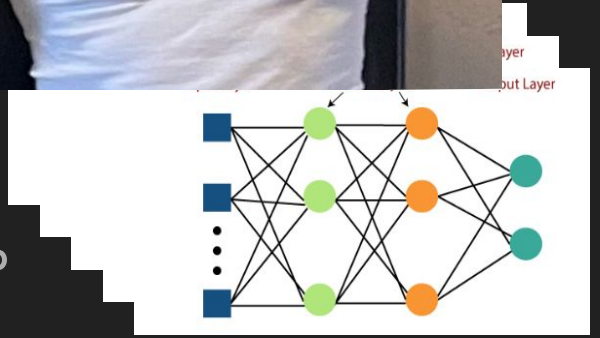
Forbes

EDITORS' PICK | INNOVATION > AI

## Juergen Schmidhuber, Renowned 'Father Of Modern AI,' Says His Life's Work Won't Lead To Dystopia

By Hessie Jones, Contributor. ⓘ Hessie Jones is a strategist, entrepreneur... Follow Author

May 23, 2023, 08:03am EDT

neural network – an MLP

# Recurrent Neural Network (RNN) - likely your second NN

For each example of X and Y I have…

Given X, what does the RNN think Y is?

As we know, loss = how wrong it was

We can use that to calculate the direction

Inform the network on how wrong it was

```
for epoch in range(epochs):
    model.train()
    for train_features, train_labels in train_dl:
        opt.zero_grad()
        yhat = model(train_features)
        loss = criterion(yhat, train_labels)
        loss.backward()
        opt.step()
```

(à la the prior gradient descent )

# Recurrent Neural Network (RNN) - likely your second NN

For each example of X and Y I have…

Given X, what does the RNN think Y is?

As we know, loss = how wrong it was

We can use that to calculate the direction

Inform the network on how wrong it was

```
for epoch in range(epochs):
    model.train()
    for train_features, train_labels in train_dl:
        opt.zero_grad()
        yhat = model(train_features)
        loss = criterion(yhat, train_labels)
        loss.backward()
        opt.step()
```

(à la the prior gradient descent )

# Recurrent Neural Network (RNN) - They work because memory

Input data size

ML model size
I.e. how many dims is it allowed to represent this as

```python
# -- Model --
class SimpleRNN(nn.Module):
    def __init__(self, input_size, hidden_size):
        super().__init__()
        self.rnn = nn.RNN(input_size, hidden_size, batch_first=True)
        self.fc = nn.Linear(hidden_size, 1)

    def forward(self, x):
        out, _ = self.rnn(x)
        return self.fc(out[:, -1])
```

One full pass through the entire training dataset.

```python
# -- Training loop --
losses = []
for epoch in range(EPOCHS):
    model.train()
    pred = model(X_torch)
    loss = loss_fn(pred, Y_torch)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

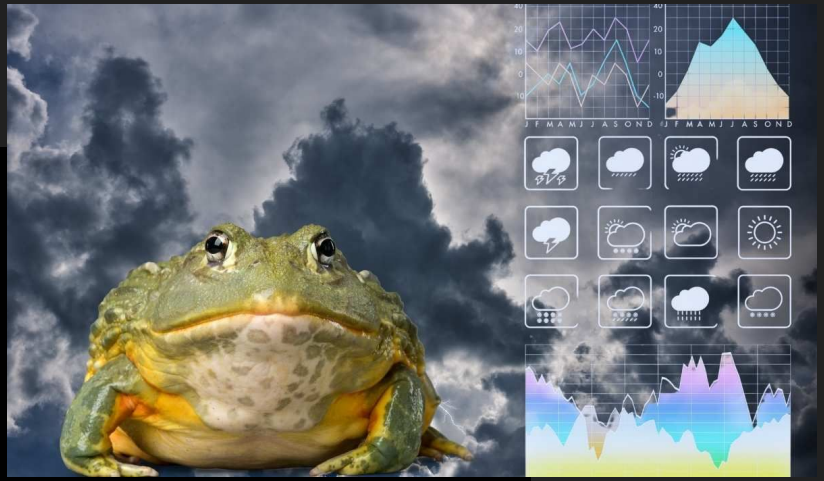
# Required specs and time (i.e. why only ML now?):

Specs:

Min : mid-tier GPU (RTX 3060–3080) & 16GB RAM

Recommended : High-end GPU (A100, V100, H100) & 32GB+ RAM

Time to predict the weather!



GitHub

# How is Deep Learning different from classical methods?

Deep Learning

-Linearity not required

-Covariance of data not necessary apriori

-Scalable to computational limits

-Black box

-Trained

Classical

-Knowledge of linearity required

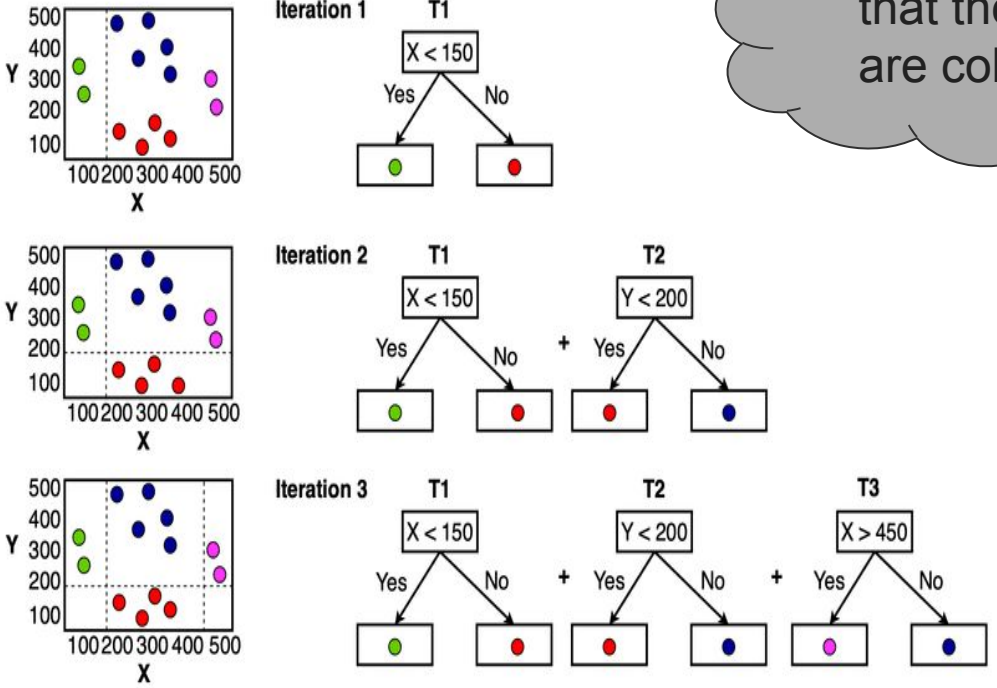
-Knowledge of covariance required
(i.e an equation)

-Easy to understand

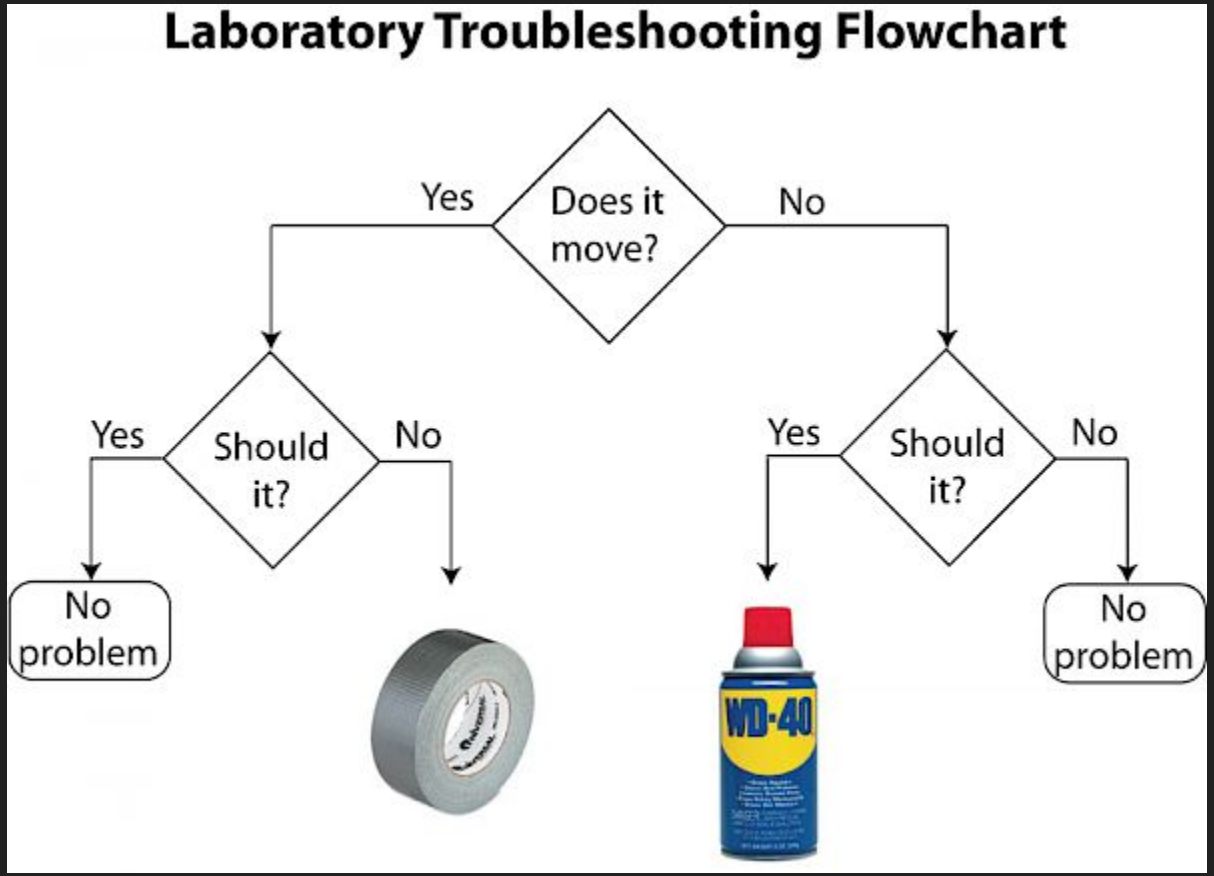
-Only data of interest required

DECISION TREES??

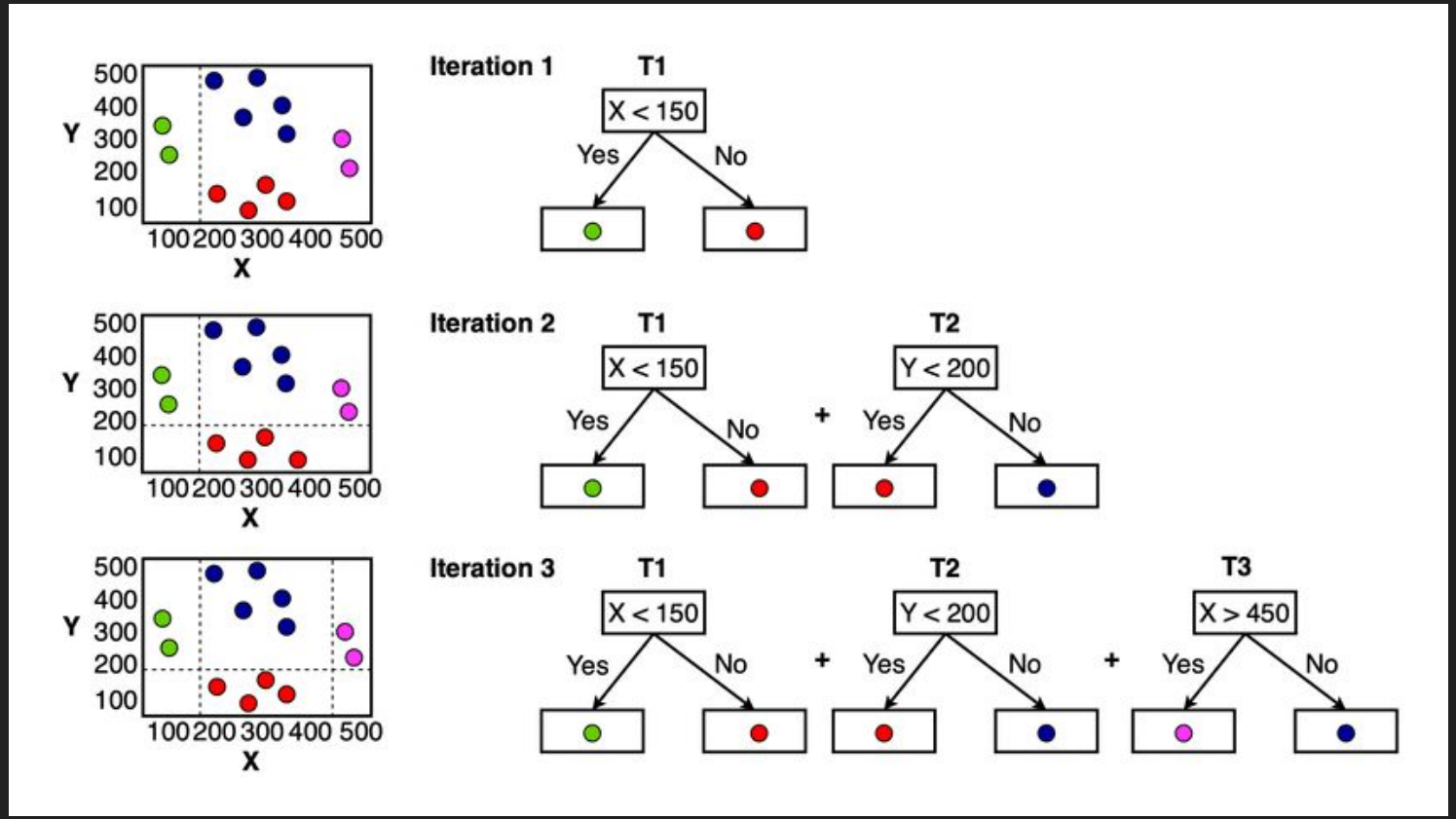
# Decision trees




# Decision trees



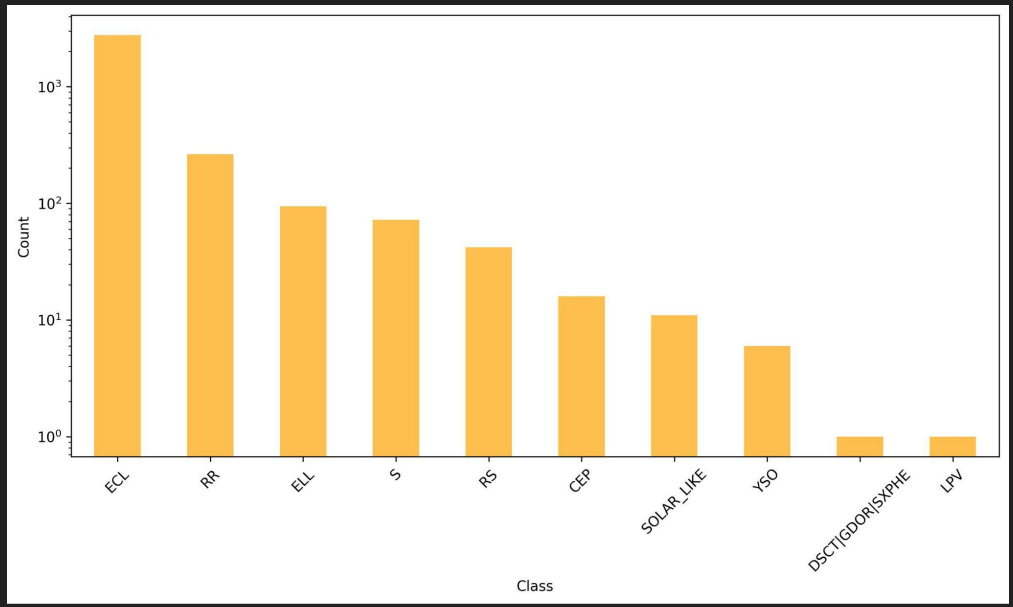
# Decision trees

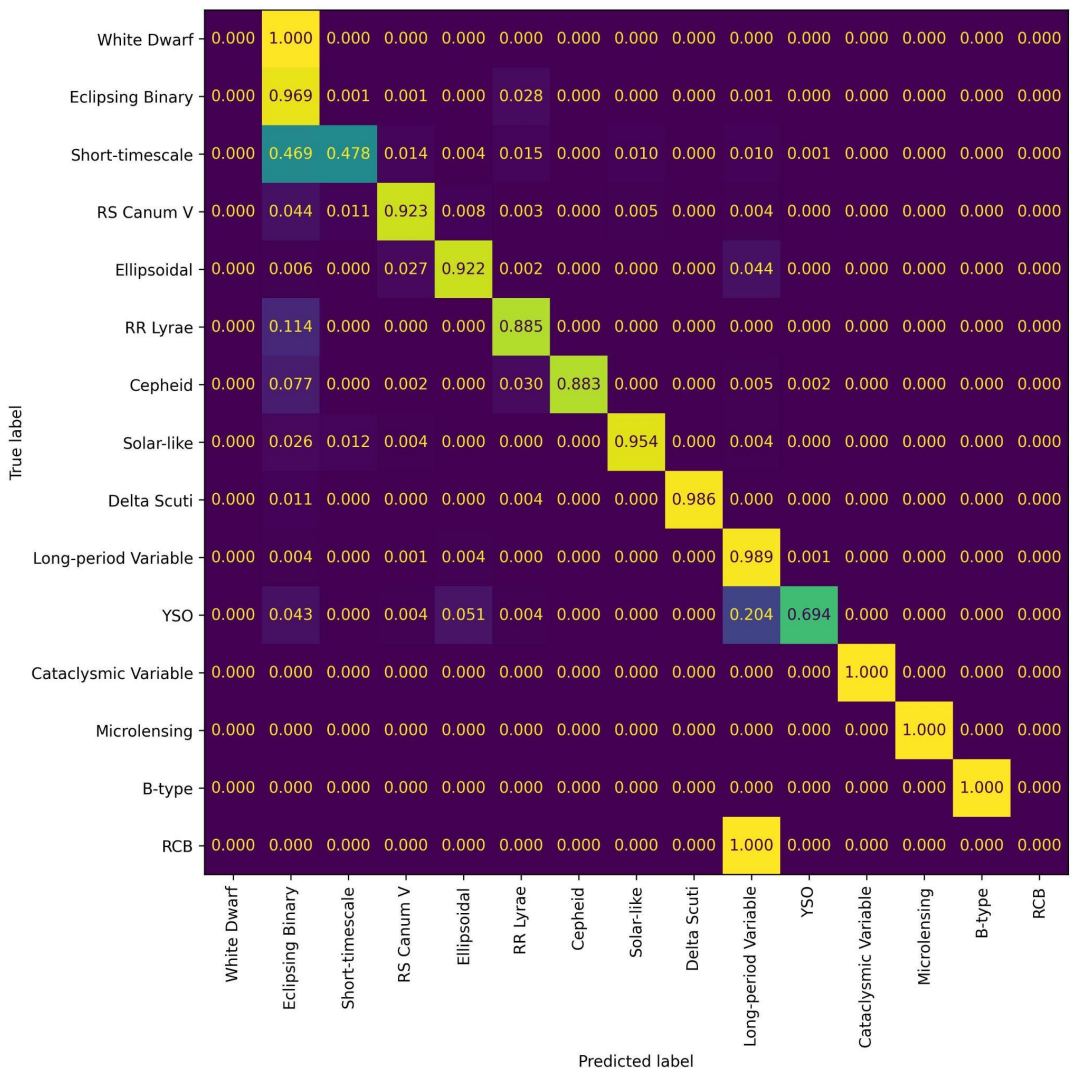


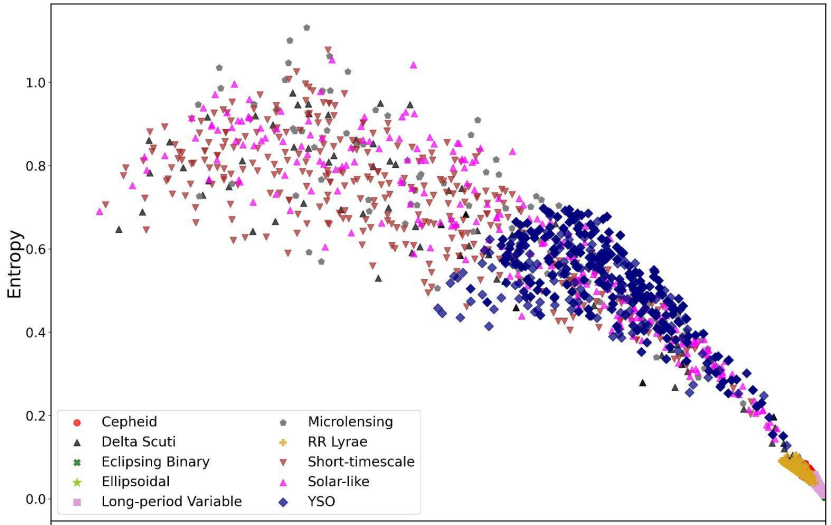
| | |
|---|---|
| *z med mag-ks med mag | Median z band magnitude - median ks mag |
| *y med mag-ks med mag | Median y band magnitude - median ks mag |
| *j med mag-ks med mag | Median j band magnitude - median ks mag |
| *h med mag-ks med mag | Median h band magnitude - median ks mag |
| *l | galactic longitude |
| *b | galactic latitude |
| Cody M | AM Cody 'M' value |
| stet k | Stetson 'K' value |
| eta e | Von Neumann 'eta e' vallue |
| med BRP | Median buffer range percentage |
| range cum sum | Range of a cumulative sum |
| max slope | Maximum slope between two points |
| MAD | Median Absolute Deviation |
| mean var | Mean Variance |
| percent amp | Percentage Amplitude |
| true amplitude | Amplitude |
| roms | RObust Median Statistic |
| p to p var | Peak-to-peak variability |
| lag auto | Lag-1 autocorrelation |
| AD | Anderson-Darling |
| std nxs | Normalized excess variance |
| weight mean | Weighted Mean |
| weight std | Weighted Standard deviation |
| weight skew | Weighted Skew |
| weight kurt | Weighted Kurtosis |
| mean | Mean |
| std | Standard deviation |
| skew | Skew |
| kurt | Kurtosis |
| true period | Period |

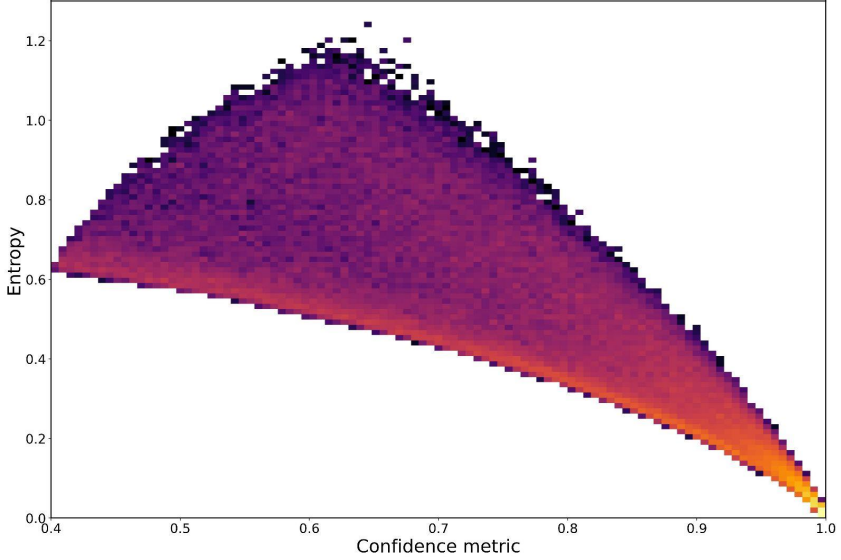
E 3.4: Table showing each of the features used in the embedding process. Features with a '*' are taken from the VIRAC catalogue.

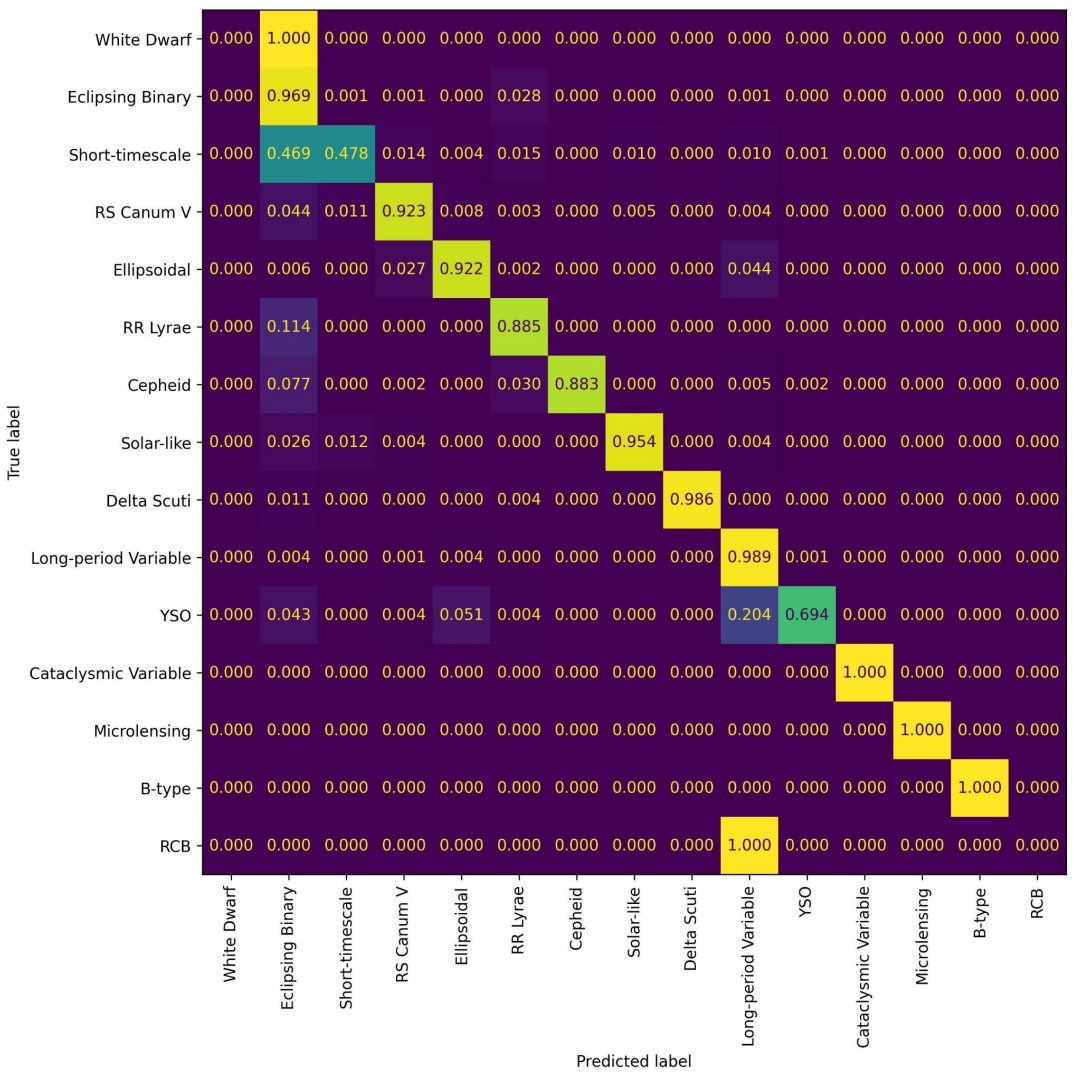
I thinks that these features can paramaterise all of the below classes from gaia

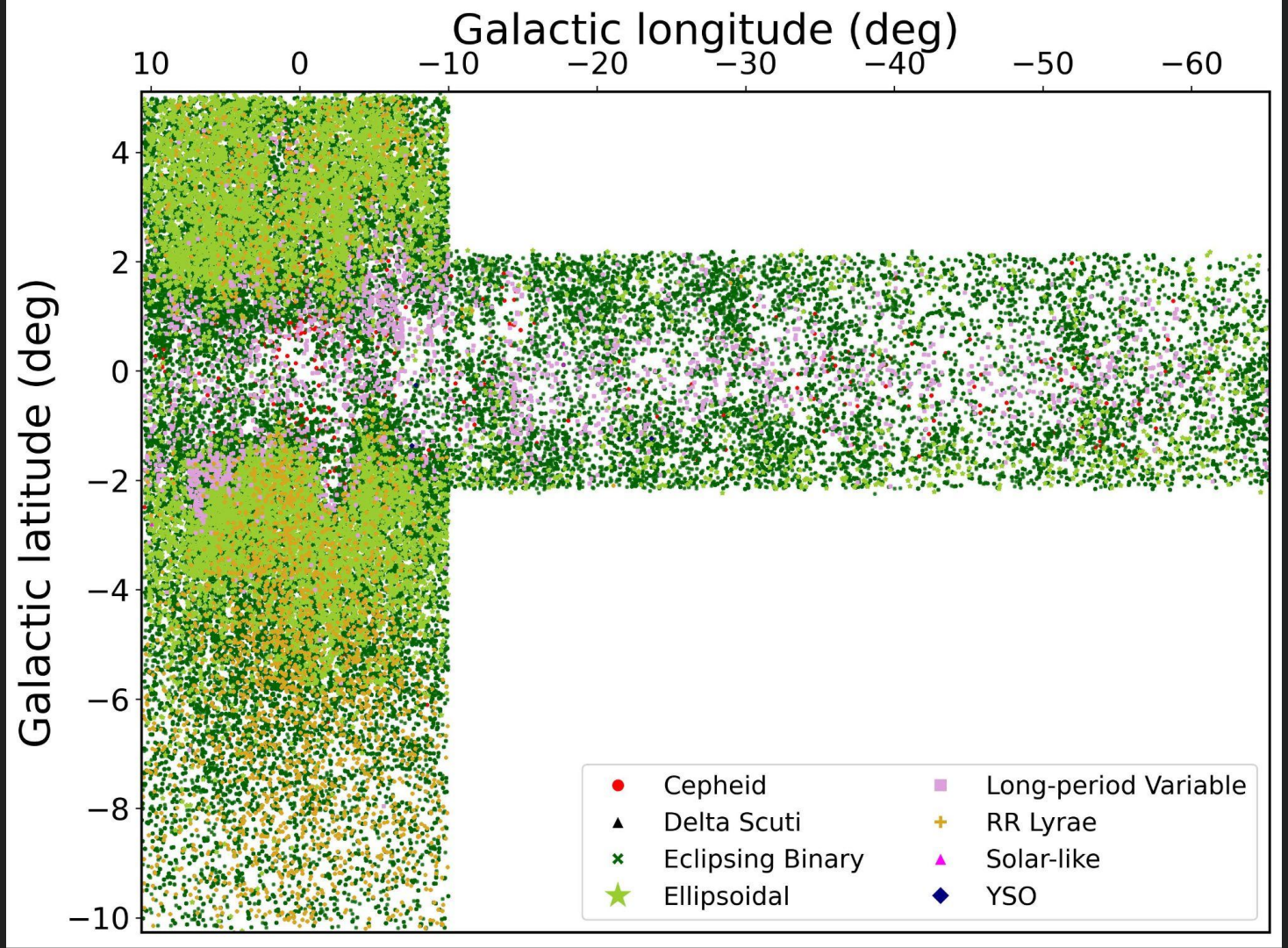



True label
White Dwarf 0.000 1.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
Eclipsing Binary 0.000 0.969 0.001 0.001 0.000 0.028 0.000 0.000 0.000 0.001 0.000 0.000 0.000 0.000 0.000
Short-timescale 0.000 0.469 0.478 0.014 0.004 0.015 0.000 0.010 0.000 0.010 0.001 0.000 0.000 0.000 0.000
RS Canum V 0.000 0.044 0.011 0.923 0.008 0.003 0.000 0.005 0.000 0.004 0.000 0.000 0.000 0.000 0.000
Ellipsoidal 0.000 0.006 0.000 0.027 0.922 0.002 0.000 0.000 0.000 0.044 0.000 0.000 0.000 0.000 0.000
RR Lyrae 0.000 0.114 0.000 0.000 0.000 0.885 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
Cepheid 0.000 0.077 0.000 0.002 0.000 0.030 0.883 0.000 0.000 0.005 0.002 0.000 0.000 0.000 0.000
Solar-like 0.000 0.026 0.012 0.004 0.000 0.000 0.000 0.954 0.000 0.004 0.000 0.000 0.000 0.000 0.000
Delta Scuti 0.000 0.011 0.000 0.000 0.000 0.004 0.000 0.000 0.986 0.000 0.000 0.000 0.000 0.000 0.000
Long-period Variable 0.000 0.004 0.000 0.001 0.004 0.000 0.000 0.000 0.000 0.989 0.001 0.000 0.000 0.000 0.000
YSO 0.000 0.043 0.000 0.004 0.051 0.004 0.000 0.000 0.000 0.204 0.694 0.000 0.000 0.000 0.000
Cataclysmic Variable 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 1.000 0.000 0.000 0.000
Microlensing 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 1.000 0.000 0.000
B-type 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 1.000 0.000
RCB 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 1.000 0.000 0.000 0.000 0.000 0.000
White Dwarf
Eclipsing Binary
Short-timescale
RS Canum V
Ellipsoidal
RR Lyrae
Cepheid
Solar-like
Delta Scuti
Long-period Variable
YSO
Cataclysmic Variable
Microlensing
B-type
RCB
Predicted label


Entropy
Cepheid
Delta Scuti
Eclipsing Binary
Ellipsoidal
Long-period Variable
Microlensing
RR Lyrae
Short-timescale
Solar-like
YSO


Entropy
Confidence metric


True label
Predicted label
White Dwarf
Eclipsing Binary
Short-timescale
RS Canum V
Ellipsoidal
RR Lyrae
Cepheid
Solar-like
Delta Scuti
Long-period Variable
YSO
Cataclysmic Variable
Microlensing
B-type
RCB

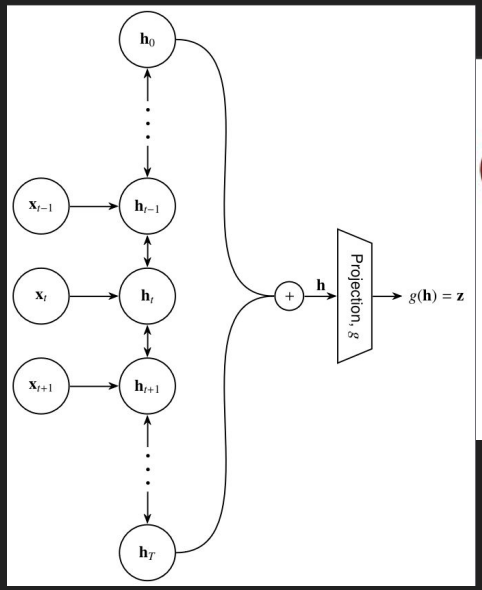

Galactic longitude (deg)
10
0
−10
−20
−30
−40
−50
−60
Galactic latitude (deg)
4
2
0
−2
−4
−6
−8
−10
Cepheid
Delta Scuti
Eclipsing Binary
Ellipsoidal
Long-period Variable
RR Lyrae
Solar-like
YSO

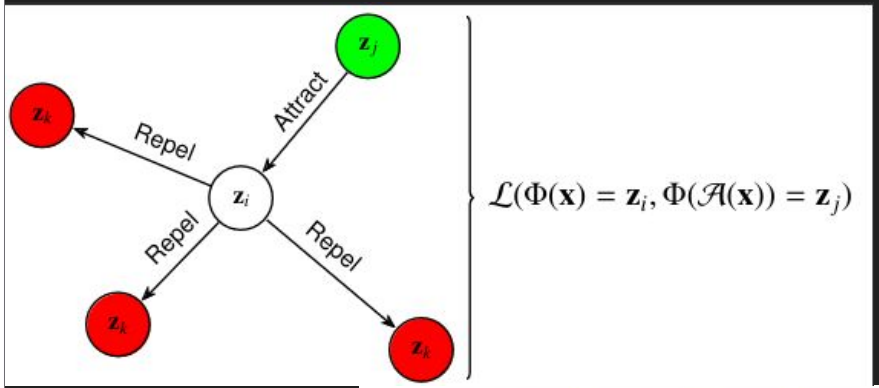
# Required specs and time:

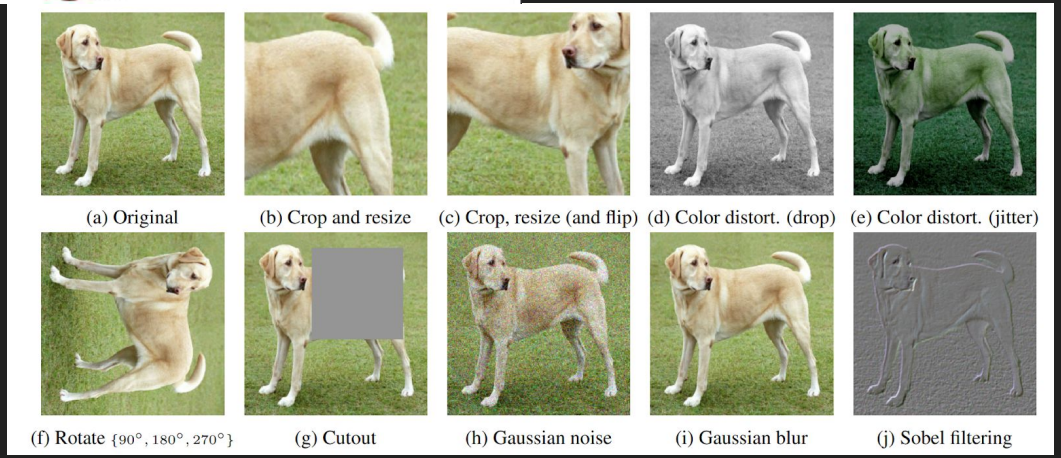
MLPs were first as they're the easiest to understand.
Consequently, they are the cheapest to run.

| Resource | MLP Needs | MedBow Has | % of Need |
|---|---|---|---|
| **Total GPUs** | **0** (CPU is fine) | 32 A100s, 32 V100s, lots of CPUs | **3200%** overkill |
| **VRAM** | **0** (only needs CPU RAM) | 80–320 GB VRAM | **∞% excess** |
| **RAM** | ~100 MB | 192 GB per node | **200,000%** oversupply |
| **Disk** | ~10–100 MB for data + checkpoints | 100s of TB available | **1,000,000% excess** |
| **Training time** | Minutes to 1–2 hours (CPU) | MedBow can do it in seconds (GPU) | **99.9% faster than needed** |

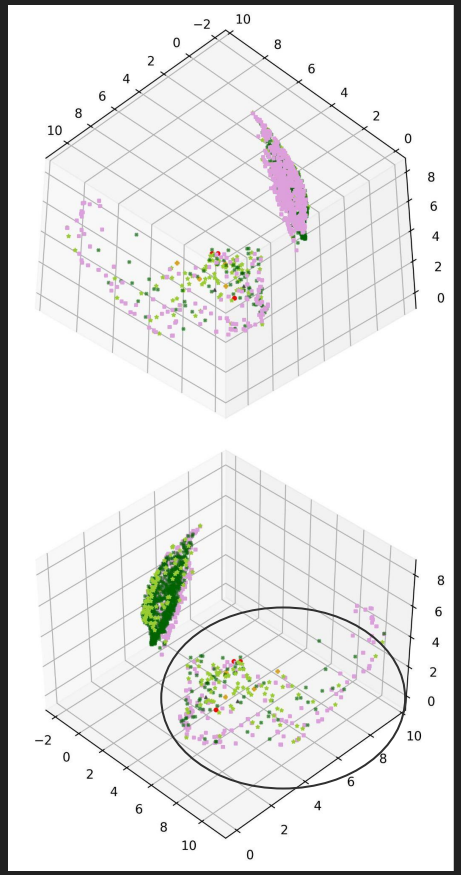
# Self supervised learning

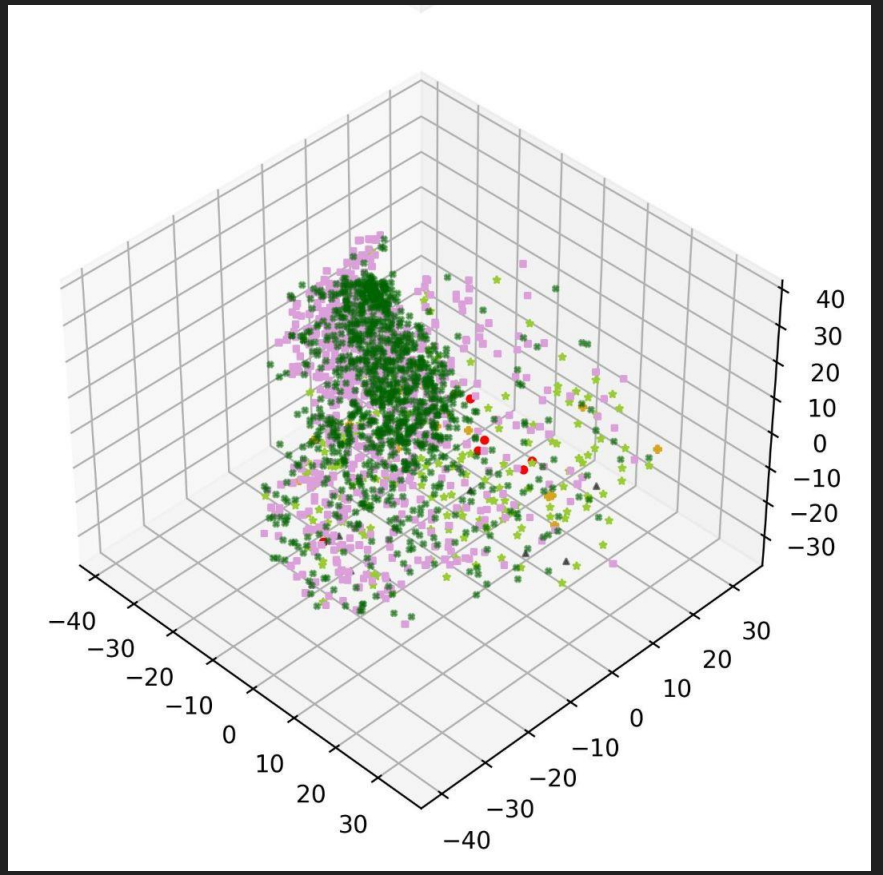


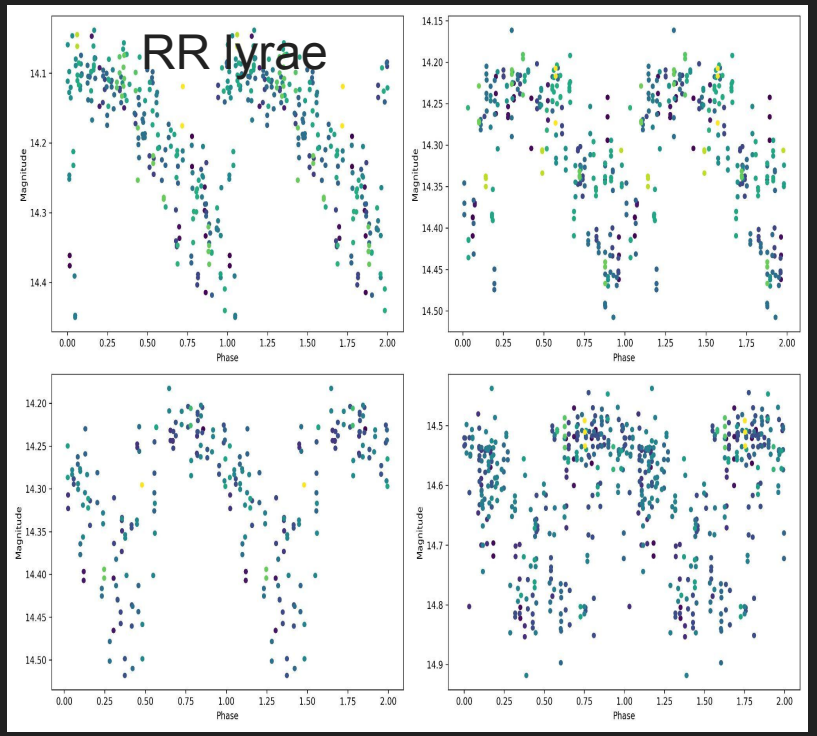



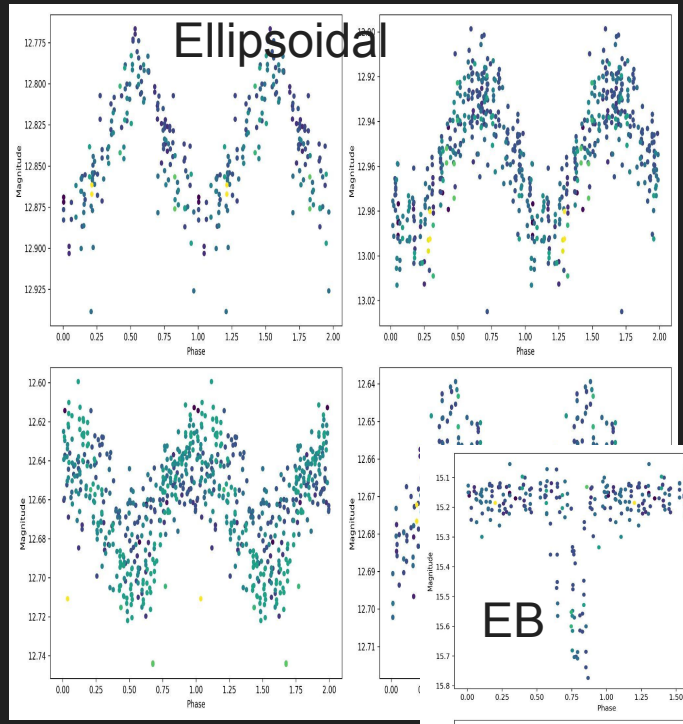



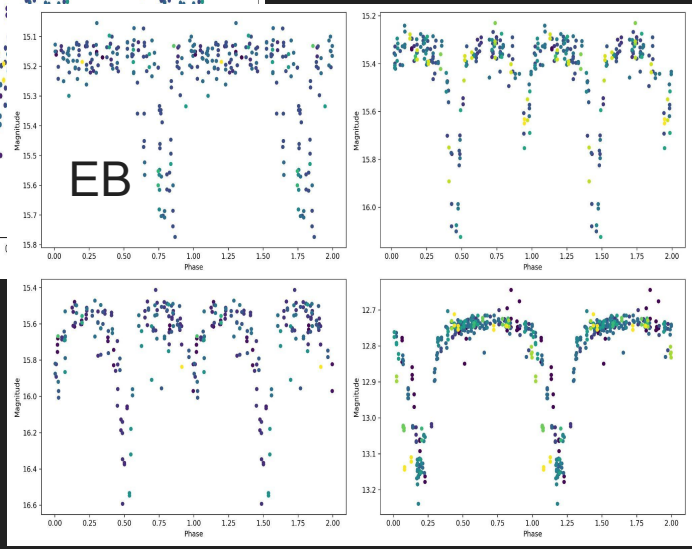
# Classification - classification of unknown unknowns






RR lyrae
Magnitude
Phase


Ellipsoidal
Magnitude
Phase


EB
Magnitude
Phase

# Required specs and time:

Minimum Specs (for small batches, test runs, debugging):

CPU: 4-core

RAM: 16 GB

GPU: CUDA-capable NVIDIA GPU +8 GB VRAM

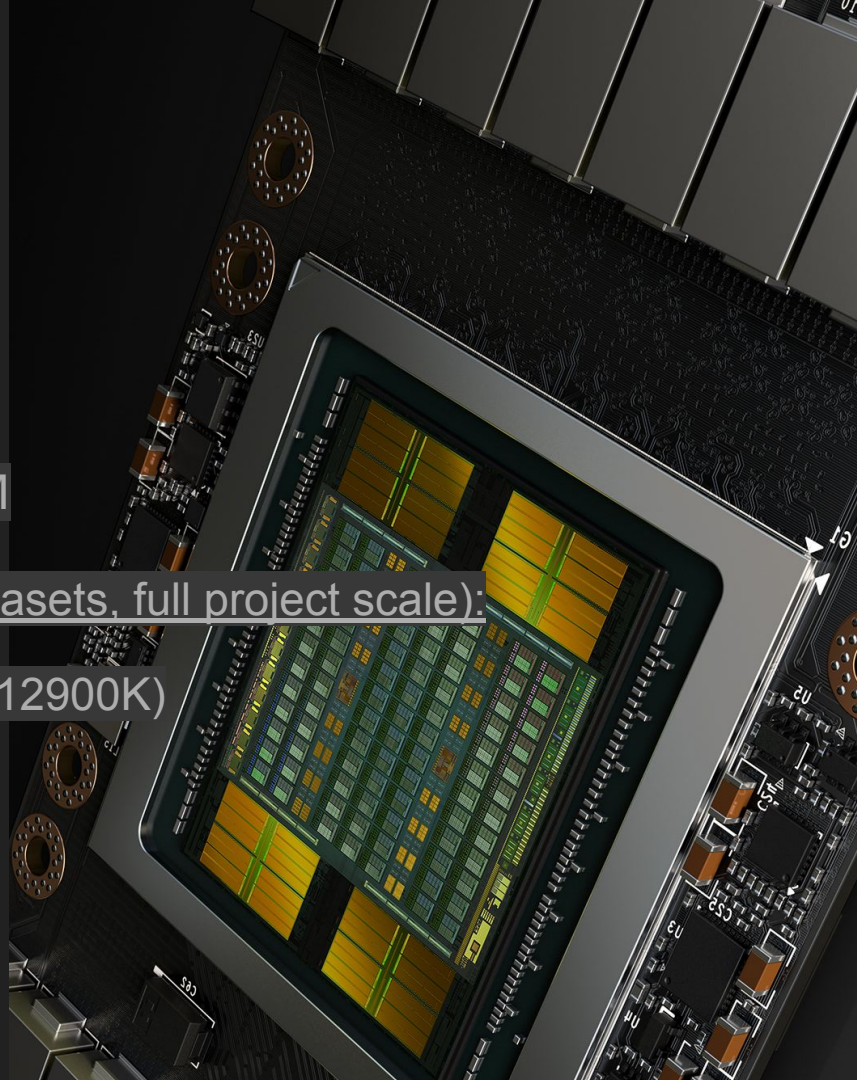
Recommended Specs (for serious training, large datasets, full project scale):

CPU: 8-core or better (e.g., Ryzen 7 5800X, Intel i9-12900K)

RAM: 64 GB

GPU: High-end NVIDIA GPU with +32 GB VRAM

(e.g., 2 * Tesla V100 [what I actually used])



# Required specs and time:

| Slurm Partition name | Requestable features | Node count | Socket/ Node | Cores/ Socket | Threads/ Core | Total Cores/ Node | RAM (GB) | Processor (x86_64) | Local Disks | OS | Use Case | Key Attributes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| mb | amd, epyc | 25 | 2 | 48 | 1 | 96 | 1024 | 2x 48-Core/96-Thread 4th Gen AMD EPYC 9454 | 4TB SSD | RHEL 9.3 | For compute jobs running the latest and greatest MedicineBow hardware | MB Compute with 1TB RAM |
| mb-a30 | amd, epyc | 8 | | | | | 768 | | | | DL Inference, AI, Mainstream Acceleration | MB Compute with 24GB RAM/GPU & A30 GPU |
| mb-l40s | amd, epyc | 5 | | | | | 768 | | | | DL Inference, Omniverse/ Rendering, Mainstream | MB Compute with 48GB RAM/GPU & L40S GPU |
| mb-h100 | amd, epyc | 6 | | | | | 1228 | | | | DL Training and Inference, DA, AI, Mainstream Acceleration | MB Compute with 80GB RAM/GPU & Nvidia SXM5 H100 GPU |

Woah, we could train this on ONE node here

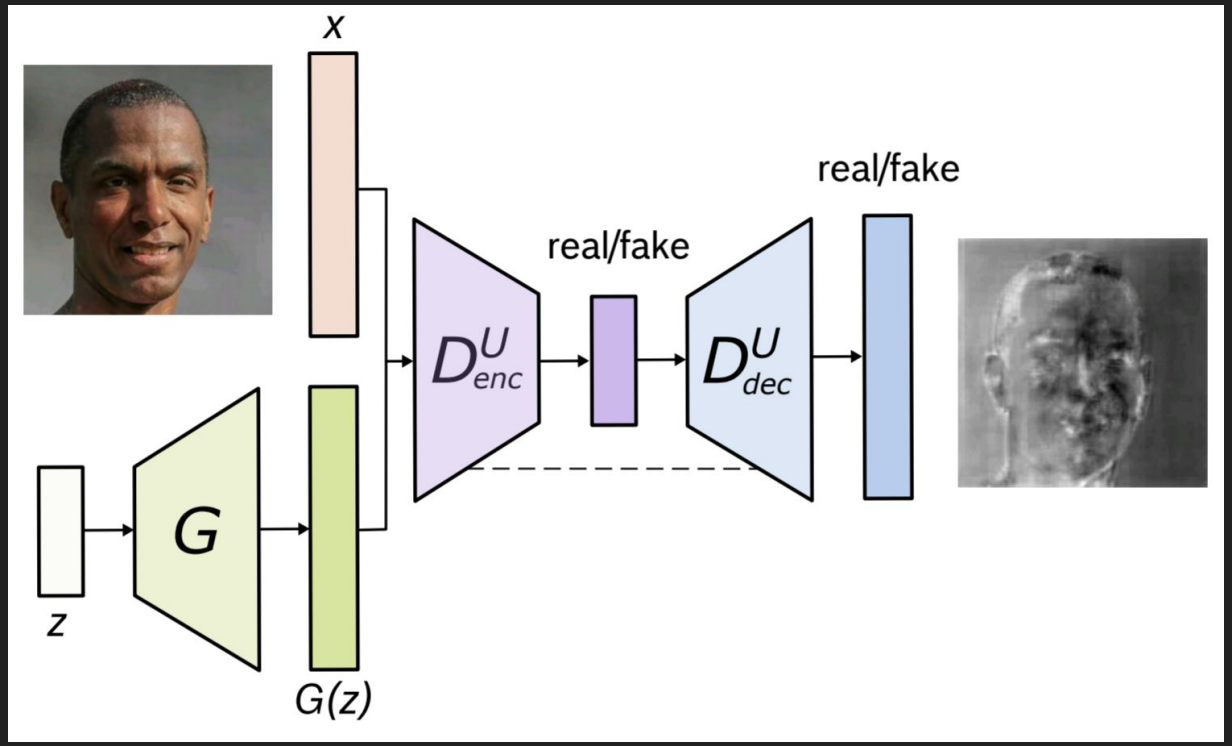
CPU: 8-core or better (e.g., Ryzen 7 5800X, Intel i9-12900K)

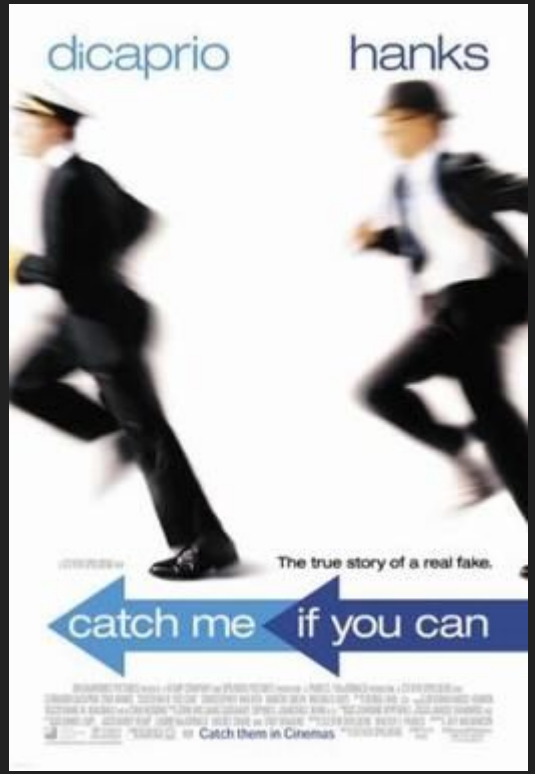
RAM: 64 GB

GPU: High-end NVIDIA GPU with +32 GB VRAM

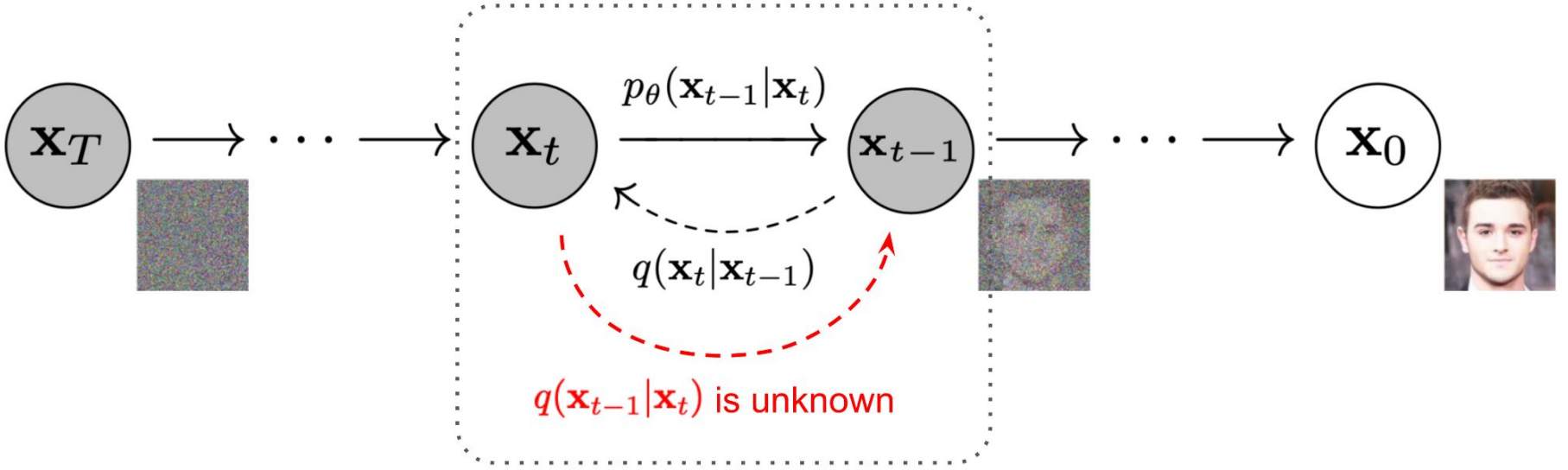
(e.g., 2 * Tesla V100 [what I actually used])

# Fake things - Generative AI - The one your mum knows.

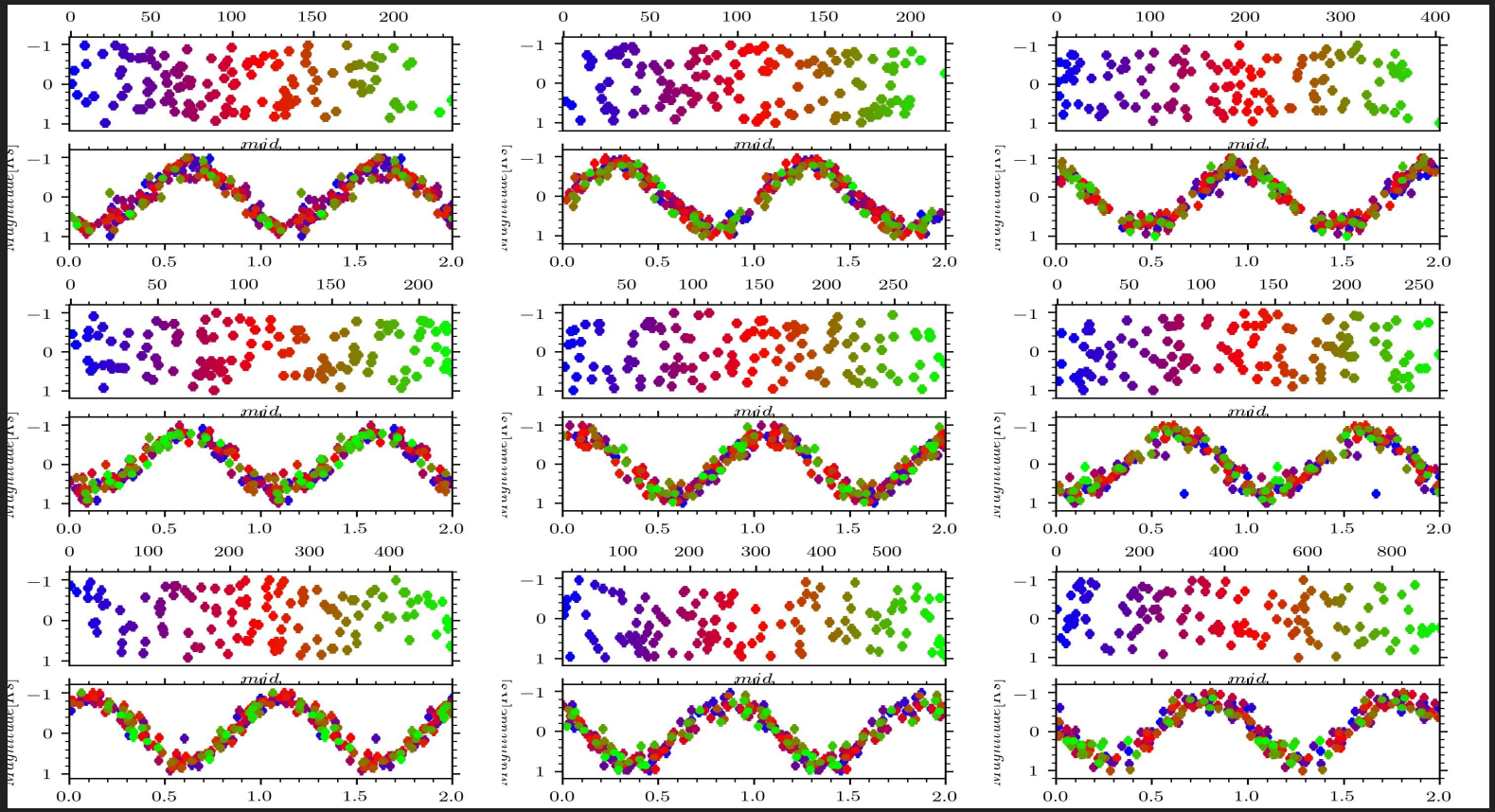


Use variational lower bound

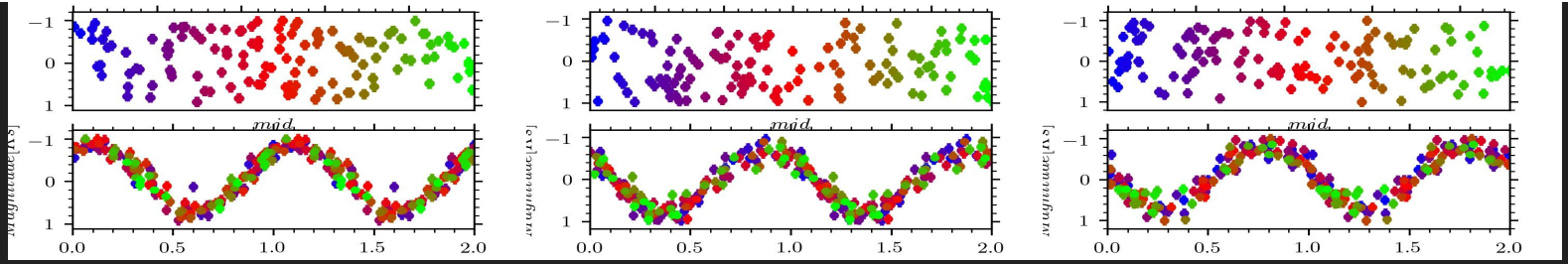


# Synthetic light curves

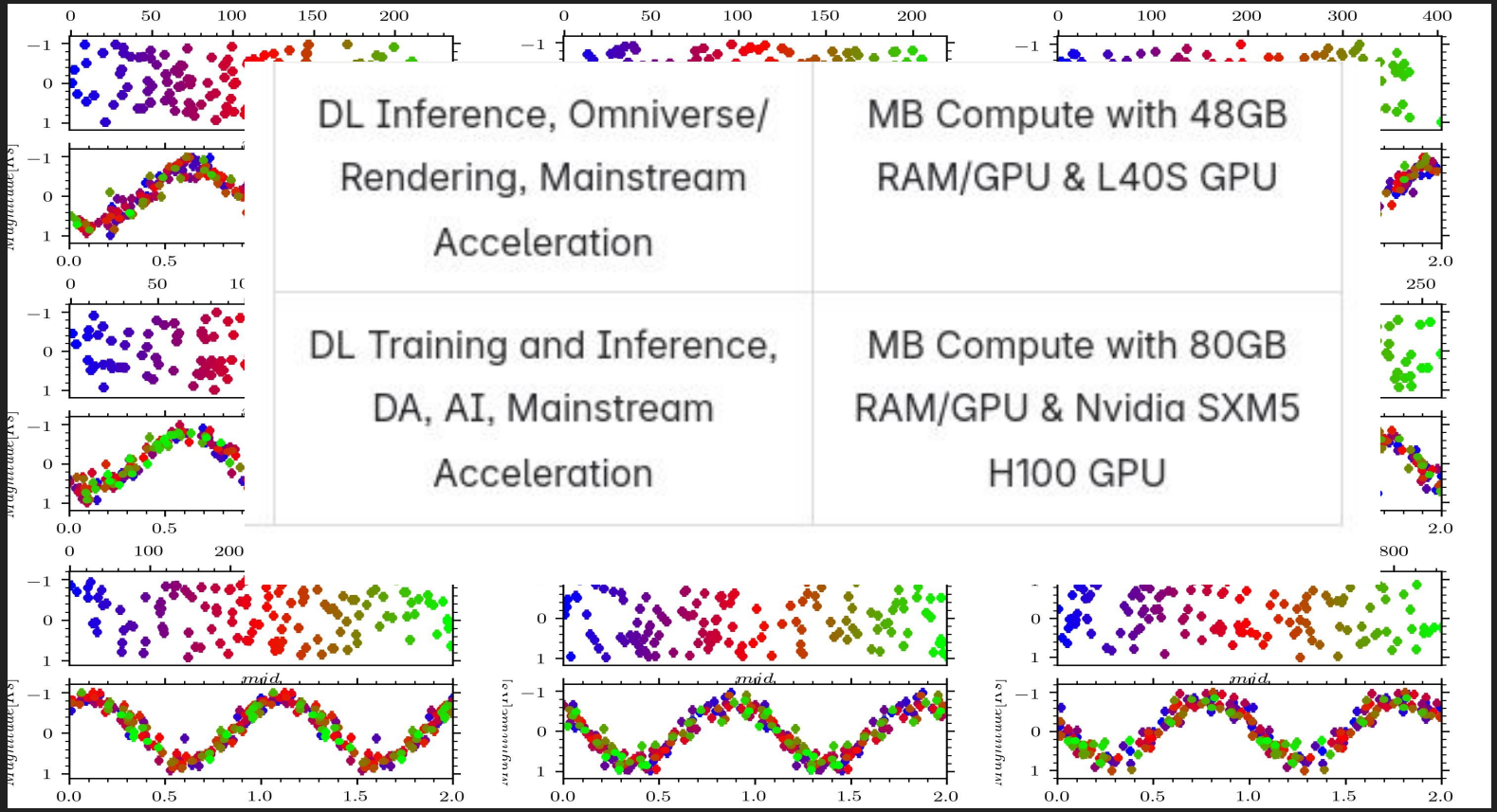


# Synthetic light curves


| Slurm Partition name | Requestable features | Node count | Socket/ Node | Cores/ Socket | Threads/ Core | Total Cores/ Node | RAM (GB) | Processor (x86_64) | Local Disks | OS | Use Case | Key Attributes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| mb | amd, epyc | 25 | 2 | 48 | 1 | 96 | 1024 | 2x 48-Core/96-Thread 4th Gen AMD EPYC 9454 | 4TB SSD | RHEL 9.3 | For compute jobs running the latest and greatest MedicineBow hardware | MB Compute with 1TB RAM |
| mb-a30 | amd, epyc | 8 | | | | | 768 | | | | DL Inference, AI, Mainstream Acceleration | MB Compute with 24GB RAM/GPU & A30 GPU |
| mb-l40s | amd, epyc | 5 | | | | | 768 | | | | DL Inference, Omniverse/ Rendering, Mainstream Acceleration | MB Compute with 48GB RAM/GPU & L40S GPU |
| mb-h100 | amd, epyc | 6 | | | | | 1228 | | | | DL Training and Inference, DA, AI, Mainstream Acceleration | MB Compute with 80GB RAM/GPU & Nvidia SXM5 H100 GPU |

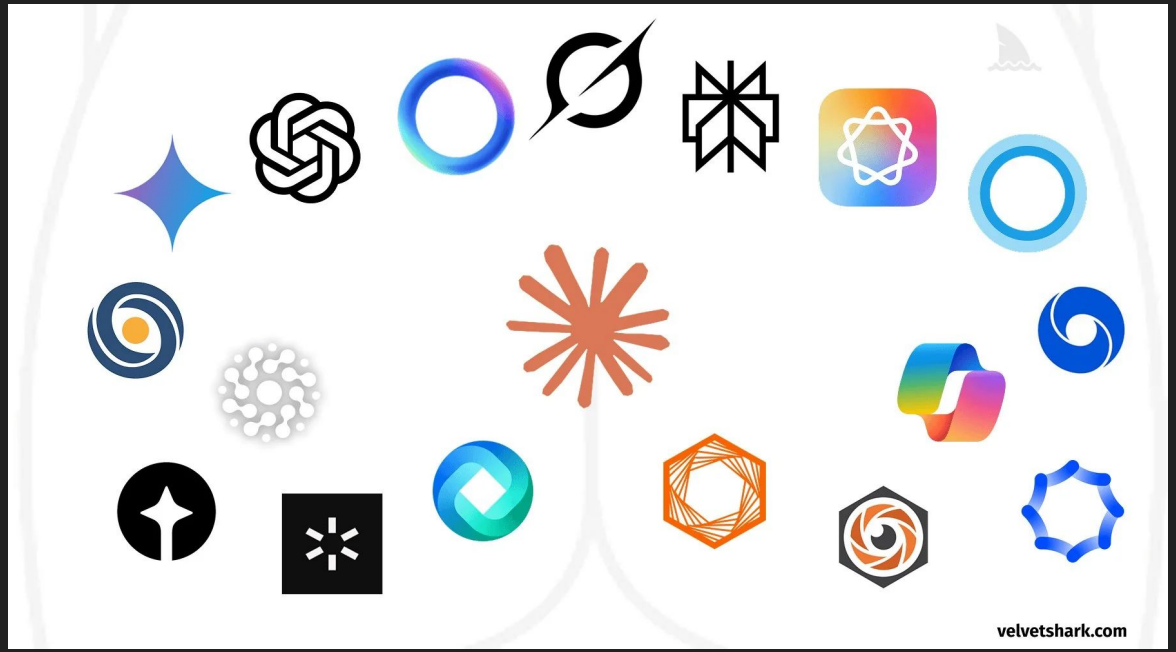


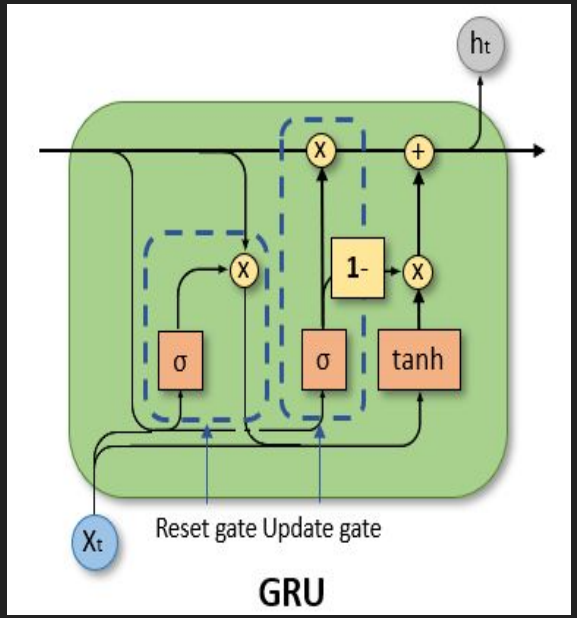
# Fake light curves



# Transformers, often generative and pre-trained

Like an RNN but sideways.

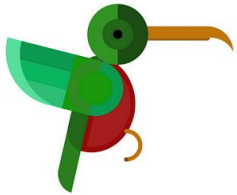


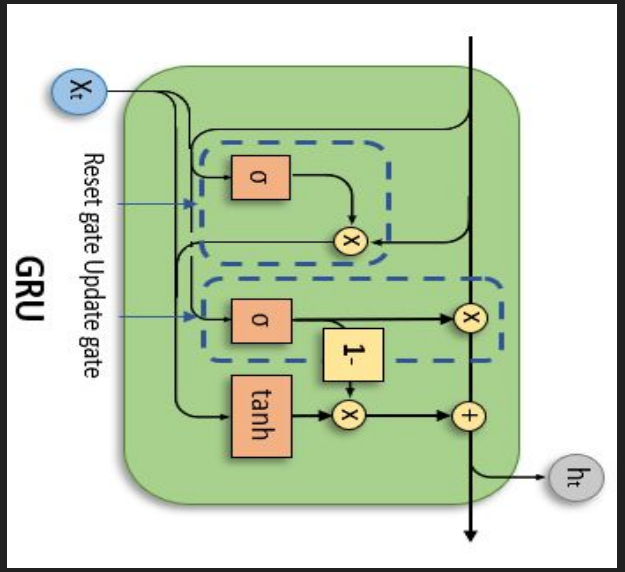
# Transformers, often generative and pre-trained

## LLM VS. PARROTS

A rigorous side-by-side comparison



| | Large Language Models | A real parrot |
|---|---|---|
| Repeats words it has encountered before | ✓ | ✓ |
| Responds to prompts by humans | ✓ | ✓ |
| Understands semantics (or phonetics), but has no grasp of the meaning of words | ✓ | ✓ |
| Potentially produces hundreds of thousands of metric tons of Carbon Dioxide | ✓ | |
| Urinates and defecates | | ✓ |



# Transformers, often generative and pre-trained

| Resource | Estimate |
| --- | --- |
| **Model Size** | ~1.8 trillion parameters (GPT-4 is roughly estimate *175B*) |
| **Training Dataset** | 13–23 trillion tokens (open datasets + filtered web |
| **VRAM Needed** | ~700–900 GB *active model memory* (just for forwar |
| **Total Compute** | ~25,000,000 GPU-hours |
| **Cluster Size** | 10,000+ A100s (80 GB) **for months** |
| **Training Time** | 2–3 months continuous 24/7 training |
| **Cost** (if renting) | ~$50–$100 million USD (assuming $2-$3/hour per |
| **Storage** | 1–2 PB (petabytes) for dataset storage, checkpoin |
| **CPU RAM** | Cluster-wide memory in **petabytes** (you need fast |
| **Interconnect** | **Infiniband HDR** or better (200 Gbps+ network fab |

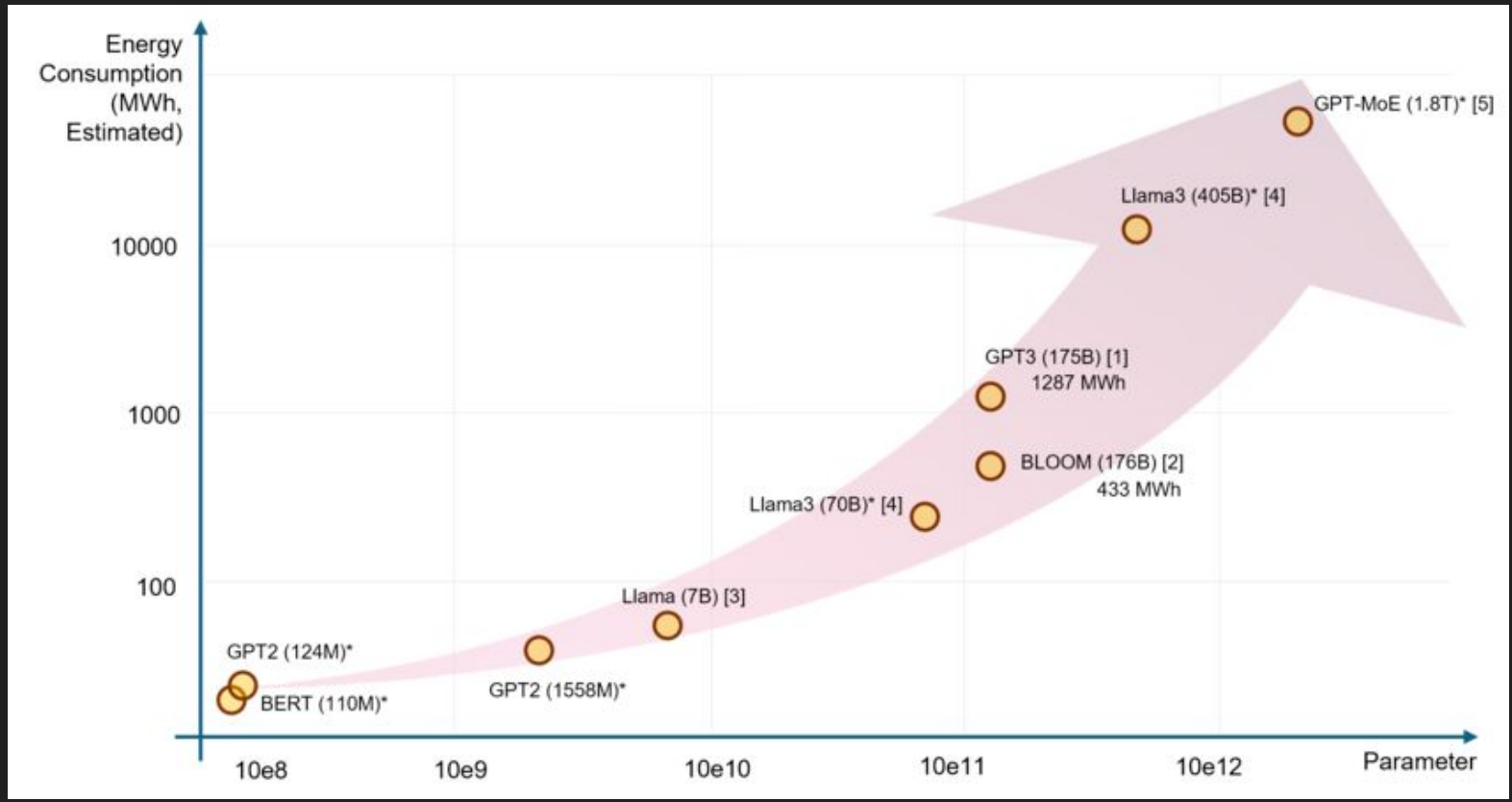
Actually maybe yes if we have nothing else.

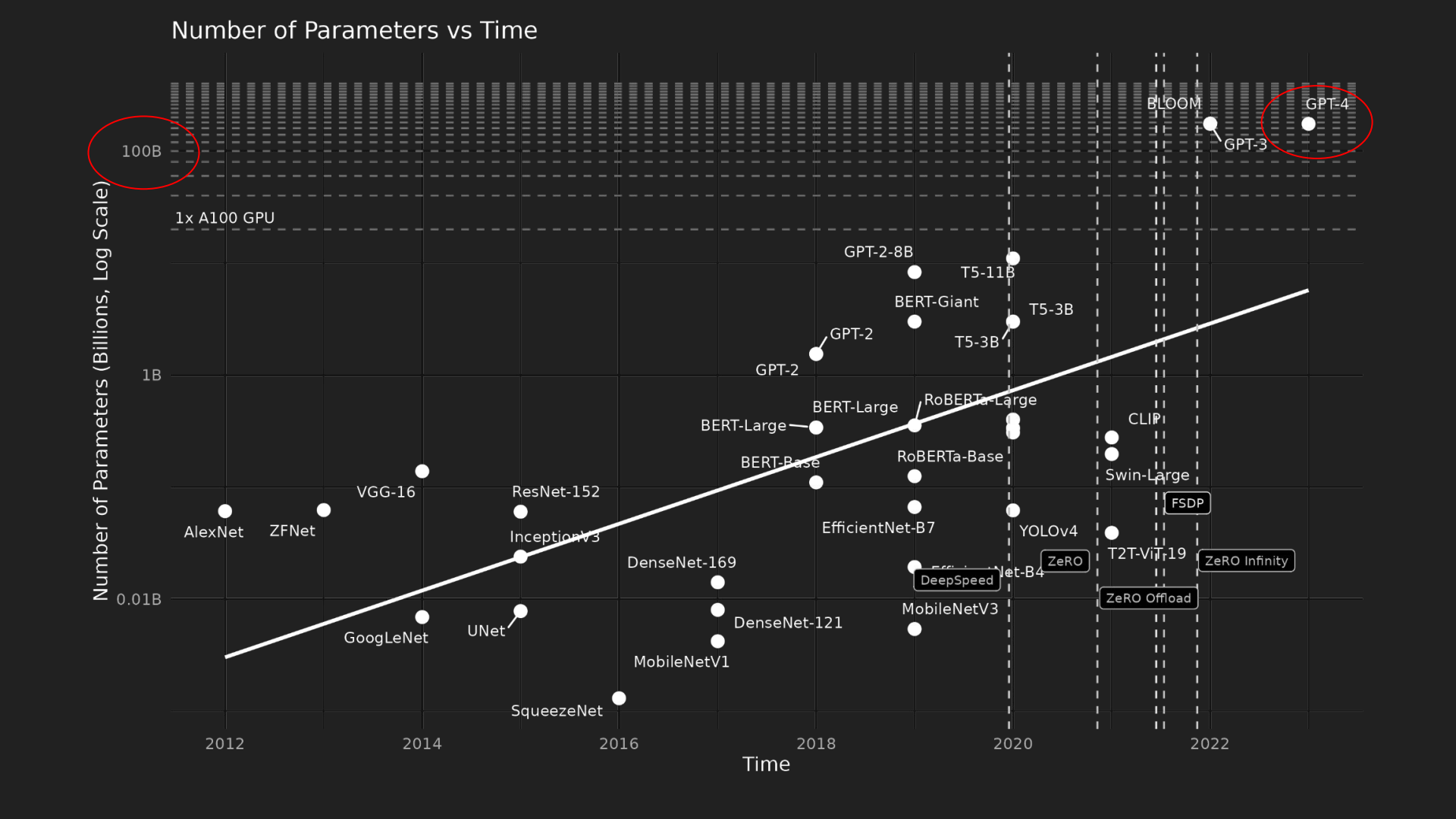
~0.32%

~1.1%

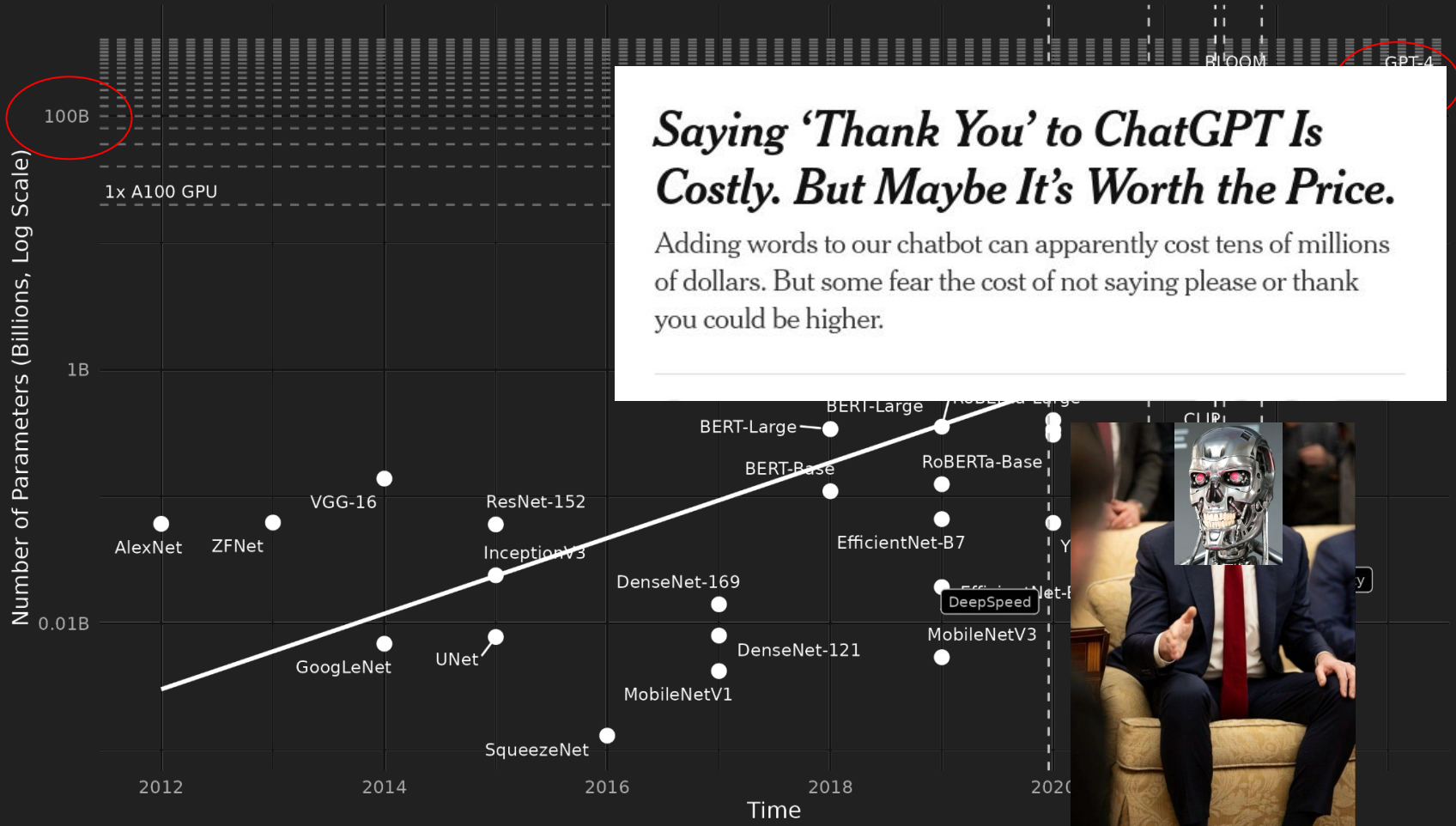
~5%

# Why you cant make one… yet…




Number of Parameters vs Time
Number of Parameters (Billions, Log Scale)
100B
1B
0.01B
1x A100 GPU
Time
2012
2014
2016
2018
2020
2022
BLOOM
GPT-4
GPT-3
GPT-2-8B
T5-11B
BERT-Giant
T5-3B
T5-3B
GPT-2
GPT-2
BERT-Large
BERT-Large
RoBERTa-Large
CLIP
RoBERTa-Base
BERT-Base
Swin-Large
VGG-16
ResNet-152
FSDP
AlexNet
ZFNet
EfficientNet-B7
YOLOv4
InceptionV3
T2T-ViT-19
ZeRO Infinity
DenseNet-169
ZeRO
DeepSpeed
ZeRO Offload
MobileNetV3
DenseNet-121
GoogLeNet
UNet
MobileNetV1
SqueezeNet

Number of Parameters vs Time




## *Saying 'Thank You' to ChatGPT Is Costly. But Maybe It's Worth the Price.*

Adding words to our chatbot can apparently cost tens of millions of dollars. But some fear the cost of not saying please or thank you could be higher.

Codey wodey