Name: Barathkrishna Satheeshkumar
Andrew ID: bsathees

Problem 1

MAE = (6 + 1 + 8 + 1 + 6)/5 = 22/5 = **4.4**

MSE = (36 + 1 + 64 + 1 + 36)/5 = 138/5 = **27.6**

MAPE = (150 + 12.5 + 114.29 + 6.67 + 50)/5 = (333.46)/5 **≈ 66.692%**

Problem 2

Matrix 1

Problem 3

Option 4: 8, 10, 0, 2, 0.8, 1.0, 0.889

M9-L1 Problem 1

Here, you will implement three loss functions from scratch in numpy: MAE, MSE, and MAPE.

In [1]:
```
import numpy as np

y_gt1 = np.array([1,2,3,4,5,6,7,8,9,10])
y_pred1 = np.array([1,1.3,3.1,4.6,5.9,5.9,6.4,9.2,8.1,10.5])

y_gt2 = np.array([-3.23594765, -3.74125693, -2.3040903 ,  0.        ,  0.30190142, -1.68434859,  1.10160357,  0
y_pred2 = np.array([-3.17886560e+00, -3.72628642e+00, -2.28154027e+00, -2.42424242e-06, 2.96261368e-01, -1.7008
```

Mean Absolute Error

Complete the definition for `MAE(y_gt, y_pred)` below.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}\left|y_i - \hat{y}_i\right| = \frac{1}{n}\sum_{i=1}^{n}\left|e_i\right|$$

`MAE(y_gt1, y_pred1)` should return 0.560.

In [2]:
```
def MAE(y_gt, y_pred):
    # YOUR CODE GOES HERE
    return np.mean(np.abs(y_gt - y_pred))


print(f"MAE(y_gt1, y_pred1) = {MAE(y_gt1, y_pred1):.3f}")
print(f"MAE(y_gt2, y_pred2) = {MAE(y_gt2, y_pred2):.3f}")
```

```
MAE(y_gt1, y_pred1) = 0.560
MAE(y_gt2, y_pred2) = 0.290
```

Mean Squared Error

Complete the definition for `MSE(y_gt, y_pred)` below.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2 = \frac{1}{n}\sum_{i=1}^{n}\left(e_i\right)^2 = \frac{1}{n}e^T e$$

`MSE(y_gt1, y_pred1)` should return 0.454.

In [3]:
```
def MSE(y_gt, y_pred):
    # YOUR CODE GOES HERE
    return np.mean(np.square(y_gt - y_pred))


print(f"MSE(y_gt1, y_pred1) = {MSE(y_gt1, y_pred1):.3f}")
print(f"MSE(y_gt2, y_pred2) = {MSE(y_gt2, y_pred2):.3f}")
```

```
MSE(y_gt1, y_pred1) = 0.454
MSE(y_gt2, y_pred2) = 0.174
```

Mean Absolute Percentage Error

Complete the definition for `MAPE(y_gt, y_pred, epsilon)` below.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}\frac{\left|y_i - \hat{y}_i\right|}{\left|y_i\right| + \varepsilon} = \frac{1}{n}\sum_{i=1}^{n}\frac{\left|e_i\right|}{\left|y_i\right| + \varepsilon}$$

`MAPE(y_gt1, y_pred1, 1e-6)` should return 0.112.

In [4]:
```
def MAPE(y_gt, y_pred, epsilon=1e-6):
    # YOUR CODE GOES HERE
    return np.mean(np.divide(np.abs(y_gt - y_pred), np.abs(y_gt)+epsilon))

print(f"MAPE(y_gt1, y_pred1) = {MAPE(y_gt1, y_pred1):.3f}")
print(f"MAPE(y_gt2, y_pred2) = {MAPE(y_gt2, y_pred2):.3f}")
```

```
MAPE(y_gt1, y_pred1) = 0.112
MAPE(y_gt2, y_pred2) = 0.032
```

M9-L1 Problem 2

Recall the von Mises stress prediction problem from the module 6 homework. In this problem, you will compute the R^2 score for a few model predictions for a single shape in this dataset. You will also plot the predicted-vs-actual stress for each model.

In [7]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score

float32 = np.float32
```

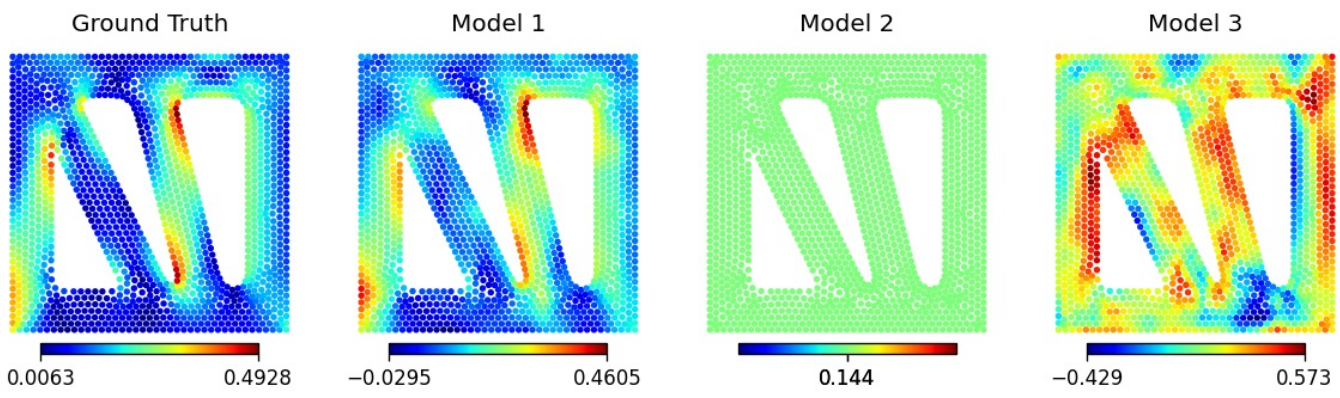
In [8]:
```
xs= np.load("data/L1P2/xs.npy")
ys= np.load("data/L1P2/ys.npy")
gt = np.load("data/L1P2/gt.npy")
model1= np.load("data/L1P2/model1.npy")
model2= np.load("data/L1P2/model2.npy")
model3= np.load("data/L1P2/model3.npy")
```

Visualizing data

Run the following cell to load the data and visualize the 3 model predictions.

- `gt` is the ground truth von Mises stress vector
- `model1` is the vector of stress predictions for model 1
- `model2` is the vector of stress predictions for model 2
- `model3` is the vector of stress predictions for model 3

In [9]:
```
def plot_shape(x, y, stress, lims=None):

    if lims is None:
        lims = [min(stress),max(stress)]

    plt.scatter(x,y,s=5,c=stress,cmap="jet",vmin=lims[0],vmax=lims[1])
    plt.colorbar(orientation="horizontal", shrink=.75, pad=0,ticks=lims)
    plt.axis("off")
    plt.axis("equal")

def plot_all(x, y, gt, model1, model2, model3):
    plt.figure(figsize=[12,3.2], dpi=120)
    plt.subplot(141)
    plot_shape(x, y, gt)
    plt.title("Ground Truth")

    plt.subplot(142)
    plot_shape(x, y, model1)
    plt.title("Model 1")

    plt.subplot(143)
    plot_shape(x, y, model2)
    plt.title("Model 2")

    plt.subplot(144)
    plot_shape(x, y, model3)
    plt.title("Model 3")

    plt.show()

plot_all(xs, ys, gt, model1, model2, model3)
```



Computing R^2

Calculate the R^2 value for each model and print the results.

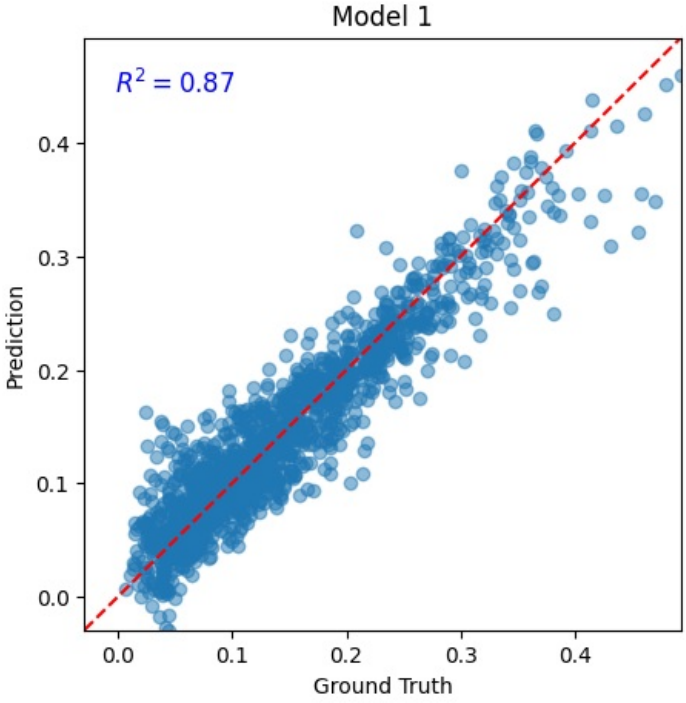
In [10]:
```
# YOUR CODE GOES HERE
def R2(y_gt, y_pred):
    num = np.sum((y_gt - y_pred) ** 2)
    den = np.sum((y_gt - np.mean(y_gt)) ** 2)

    return 1 - (num/den)

r2_m1_ = R2(gt, model1)
r2_m2_ = R2(gt, model2)
r2_m3_ = R2(gt, model3)
```

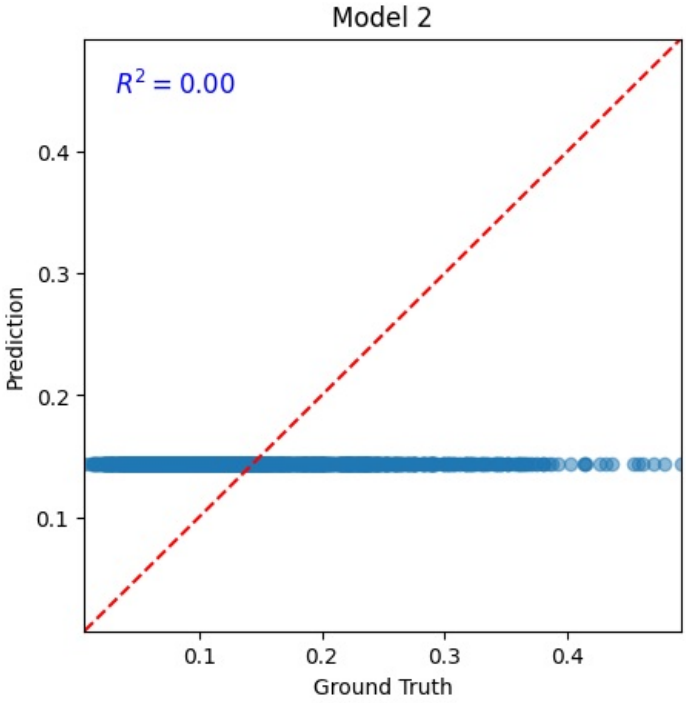
Plotting predictions vs ground truth

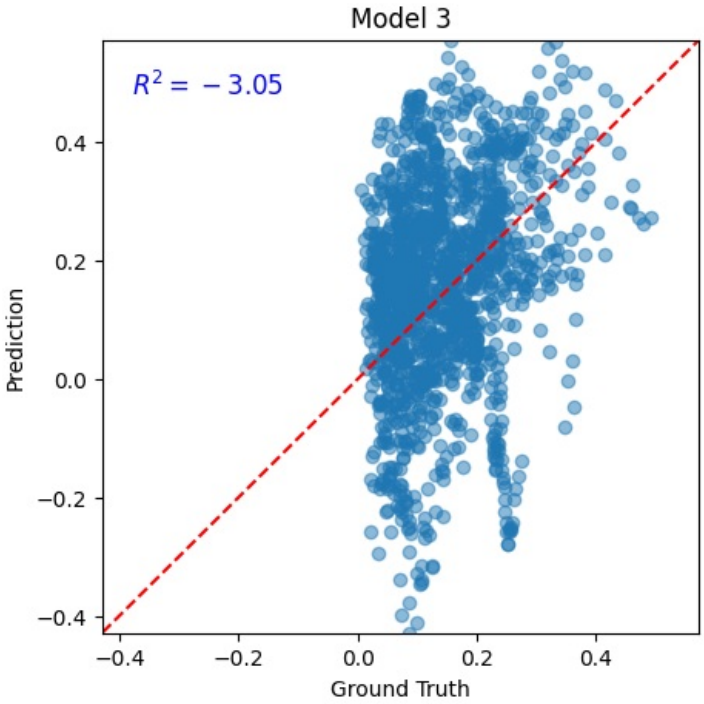
Complete the function definition below for `plot_r2(gt, pred, title)`

Then create plots for all 3 models.

In [11]:
```
def plot_r2(gt, pred, title):
    plt.figure(figsize=[5,5])

    # YOUR CODE GOES HERE
    plt.scatter(gt, pred, alpha=0.5)
    r2_value = R2(gt, pred)
    plt.text(0.05, 0.95, f"$R^2 = {r2_value:.2f}$", ha="left", va="top", transform=plt.gca().transAxes, fontsiz

    plt.plot([-1000,1000], [-1000,1000],"r--")

    all = np.concatenate([gt, pred])
    plt.xlim(np.min(all), np.max(all))
    plt.ylim(np.min(all), np.max(all))
    plt.xlabel("Ground Truth")
    plt.ylabel("Prediction")
    plt.title(title)
    plt.show()

plot_r2(gt, model1,"Model 1")
plot_r2(gt, model2,"Model 2")
plot_r2(gt, model3,"Model 3")
```







Questions

1. Model 2 has an R^2 of exactly 0. Why?
2. Model 3 has an R^2 less than 0. What does this mean?

Answers

1. An R2 value of 0 means that the model's predictions are simply predicting the mean of the ground truth values for all data points. This suggests that Model 2 is not capturing any useful information or patterns in the data to explain or predict the variability in the ground truth.

2. An R2 value less than 0 indicates that the model's predictions are worse than just using the mean of the ground truth values as the prediction for all points. This typically happens when the model is poorly fit to the data and produces predictions that have a high error compared to the variance of the ground truth.

Problem 1:

Once again consider the plane-strain compression problem shown in "data/plane-strain.png". In this problem you are given node features for 100 parts. These node features have been extracted by processing each part shape using a neural network. You will train a neural network to von Mises stress at each node given its 60 features. Then you will analyze R^2 for the training and testing data, both for the full dataset and for individual shapes within each dataset.

Summary of deliverables

- Neural network model definition
- Training function
- Training loss curve
- Overall R^2 on training and testing data
- Predicted-vs-actual plots for training and testing data
- Histograms of R^2 distributions on training and testing shapes
- Median R^2 values across training and testing shapes

In [28]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score

import torch
from torch import nn, optim

def plot_shape(dataset, index, model=None, lims=None):
    x = dataset["coordinates"][index][:,0]
    y = dataset["coordinates"][index][:,1]

    if model is None:
        c = dataset["stress"][index]
    else:
        c = model(torch.tensor(dataset["features"][index])).detach().numpy().flatten()

    if lims is None:
        lims = [min(c),max(c)]

    plt.scatter(x,y,s=5,c=c,cmap="jet",vmin=lims[0],vmax=lims[1])
    plt.colorbar(orientation="horizontal", shrink=.75, pad=0,ticks=lims)
    plt.axis("off")
    plt.axis("equal")

def plot_shape_comparison(dataset, index, model, title=""):
    plt.figure(figsize=[6,3.2], dpi=120)
    plt.subplot(1,2,1)
    plot_shape(dataset,index)
    plt.title("Ground Truth",fontsize=9,y=.96)
    plt.subplot(1,2,2)
    c = dataset["stress"][index]
    plot_shape(dataset, index, model, lims = [min(c), max(c)])
    plt.title("Prediction",fontsize=9,y=.96)
    plt.suptitle(title)
    plt.show()

def load_dataset(path):
    dataset = np.load(path)
    coordinates = []
    features = []
    stress = []
    N = np.max(dataset[:,0].astype(int)) + 1
    split = int(N*.8)
    for i in range(N):
        idx = dataset[:,0].astype(int) == i
        data = dataset[idx,:]
        coordinates.append(data[:,1:3])
        features.append(data[:,3:-1])
        stress.append(data[:,-1])
    dataset_train = dict(coordinates=coordinates[:split], features=features[:split], stress=stress[:split])
    dataset_test = dict(coordinates=coordinates[split:], features=features[split:], stress=stress[split:])
    X_train, X_test = np.concatenate(features[:split], axis=0), np.concatenate(features[split:], axis=0)
    y_train, y_test = np.concatenate(stress[:split], axis=0), np.concatenate(stress[split:], axis=0)
    return dataset_train, dataset_test, X_train, X_test, y_train, y_test

def get_shape(dataset,index):
    X = torch.tensor(dataset["features"][index])
    Y = torch.tensor(dataset["stress"][index].reshape(-1,1))
    return X, Y
```

```
def plot_r2_distribution(r2s, title=""):
    plt.figure(dpi=120,figsize=(6,4))
    plt.hist(r2s, bins=10)
    plt.xlabel("$R^2$")
    plt.ylabel("Number of shapes")
    plt.title(title)
    plt.show()
```

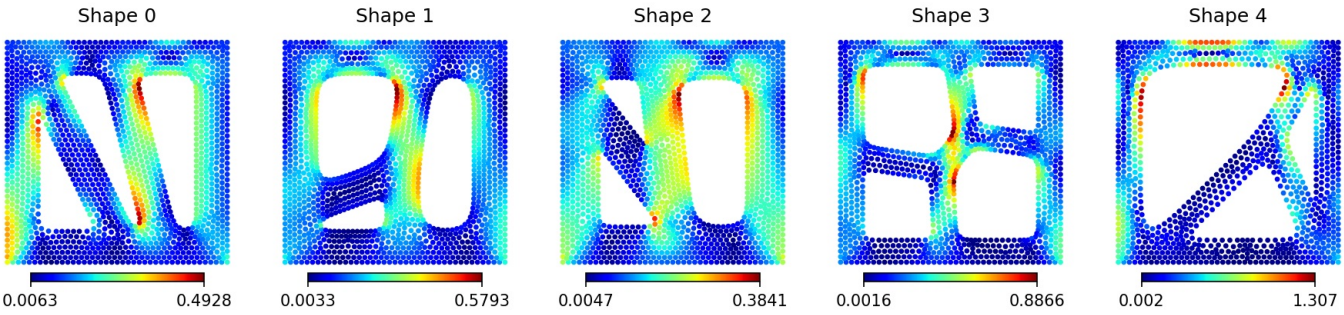
Loading the data

First, complete the code below to load the data and plot the von Mises stress fields for a few shapes.
You'll need to input the path of the data file, the rest is done for you.

All training node features and outputs are in `X_train` and `y_train`, respectively. Testing nodes are in `X_test`, `y_test`.

`dataset_train` and `dataset_test` contain more detailed information such as node coordinates, and they are separated by shape.
Get features and outputs for a shape by calling `get_shape(dataset,index)`. `N_train` and `N_test` are the number of training and testing shapes in each of these datasets.

In [29]:
```
data_path = "data/stress_nodal_features.npy"
dataset_train, dataset_test, X_train, X_test, y_train, y_test = load_dataset(data_path)
N_train = len(dataset_train["stress"])
N_test = len(dataset_test["stress"])

plt.figure(figsize=[15,3.2], dpi=150)
for i in range(5):
    plt.subplot(1,5,i+1)
    plot_shape(dataset_train,i)
    plt.title(f"Shape {i}")
plt.show()
```



Neural network to predict stress

Create a PyTorch neural network class `StressPredictor` below. This should be an MLP with 60 inputs (the given features) and 1 output (stress). The hidden layer sizes and activations are up to you.

In [30]:
```
import torch.nn.functional as F

class StressPredictor(nn.Module):
    # YOUR CODE GOES HERE
    def __init__(self):
        super(StressPredictor, self).__init__()

        self.fc1 = nn.Linear(60, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 32)
        self.output = nn.Linear(32, 1)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = self.output(x)
        return x
```

Training function

Below, you should define a function `train(model, dataset, lr, epochs)` that will train `model` on the data in `dataset` with the Adam optimizer for `epochs` epochs with a learning rate of `lr`.

Because there are so many total nodes, you should treat each shape as a batch of nodes -- each epoch of training will require you to

loop through each shape in the dataset in a random order, performing a step of gradient descent for each shape encountered. Your function should automatically generate a plot of the loss curve on training data.

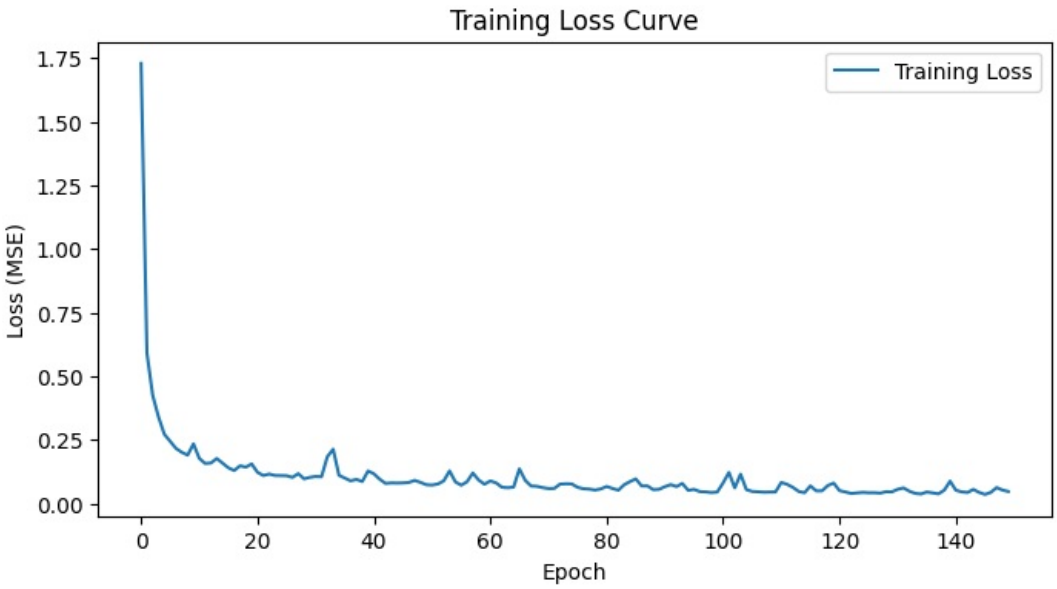
- You can use the provided `get_shape` to access feature and output tensors for each shape.
- Use MSE as a your loss function.
- Look into `np.random.permutation()` for generating a random index order

In [31]:
```
def train(model, dataset, lr, epochs):
    # YOUR CODE GOES HERE
    optimizer = optim.Adam(model.parameters(), lr = lr)
    criterion = nn.MSELoss()

    loss_values = []

    for epoch in range(epochs):
        epoch_loss = 0.0
        indices = np.random.permutation(len(dataset["features"]))

        for i in indices:
            X, y = get_shape(dataset, i)
            # X = torch.tensor(X, dtype=torch.float32)
            # y = torch.tensor(y, dtype=torch.float32)
            predictions = model(X)
            loss = criterion(predictions, y)
            loss.backward()
            optimizer.step()
            optimizer.zero_grad()
            epoch_loss += loss.item()

        loss_values.append(epoch_loss)

    plt.figure(figsize=(8, 4))
    plt.plot(loss_values, label="Training Loss")
    plt.xlabel("Epoch")
    plt.ylabel("Loss (MSE)")
    plt.title("Training Loss Curve")
    plt.legend()
    plt.show()

    return model
```

Training your Neural Network

Now, create your neural network model and run your train function on the training dataset `dataset_train`.
Determining the right number of epochs and learning rate are up to you. The training loss curve should be shown.

In [32]:
```
# YOUR CODE GOES HERE
hidden_dim = [100, 100]
activations = [nn.ReLU(), nn.ReLU()]

# model = StressPredictor(hidden_dim, activations)
model = StressPredictor()

learning_rate = 0.001
num_epochs = 150

model = train(model, dataset_train, learning_rate, num_epochs)
```



R^2 Score

Compute the R^2 Score on the training dataset. You will have to convert between tensors and arrays versions to use sklearn functions, or you can write your own function.

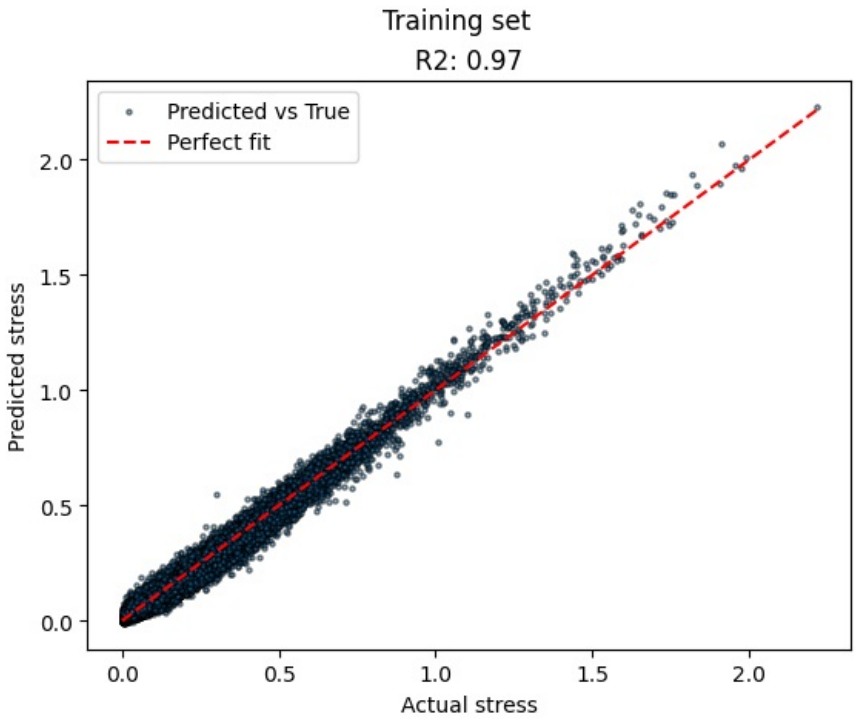
In [33]:
```
# YOUR CODE GOES HERE
from sklearn.metrics import r2_score

def compute_r2(model, X_train, y_train):
    X_train = torch.tensor(X_train)
    y_train = torch.tensor(y_train)
    y_pred = model(X_train)

    r2 = r2_score(y_train.detach().numpy(), y_pred.detach().numpy())

    return r2

r2_train = compute_r2(model, X_train, y_train)
r2_test = compute_r2(model, X_test, y_test)

print(f"R^2 score on training dataset: {r2_train:.4f}")
print(f"R^2 score on test dataset: {r2_test:.4f}")
```

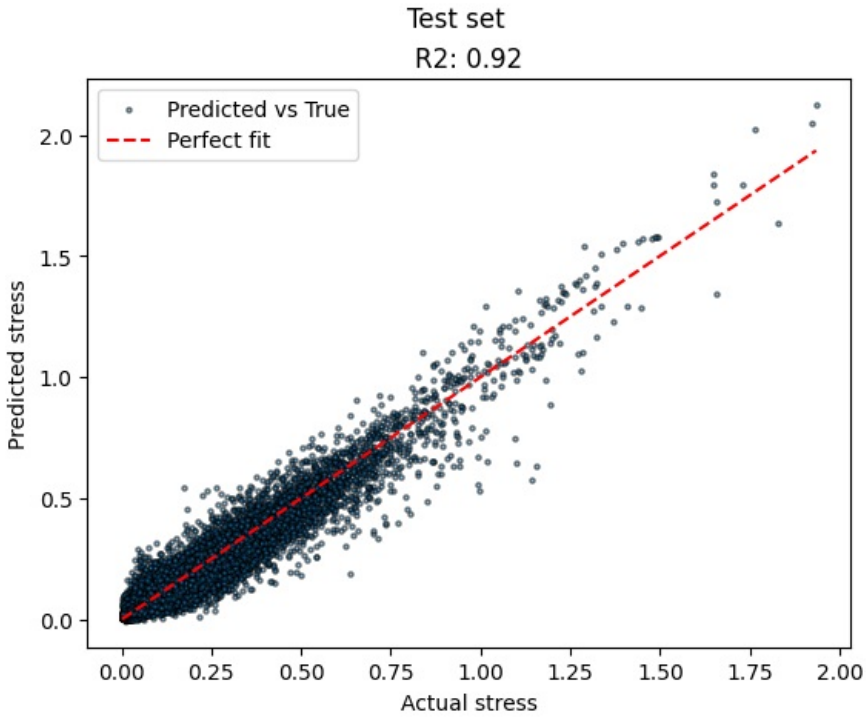
```
R^2 score on training dataset: 0.9721
R^2 score on test dataset: 0.9190
```

R^2 Plots

Now, generate predicted-vs-actual plots that display both data and a theoretical best fit line. Make 2 such plots - one for training data and one for testing.

In [34]:
```
# YOUR CODE GOES HERE
def plot_r2(model, X, y, title=""):
    y_pred = model(torch.tensor(X))
    r2 = r2_score(y, y_pred.detach().numpy())

    plt.scatter(y, y_pred.detach().numpy(), s=5, alpha=.5, edgecolors="k", label="Predicted vs True")
    plt.plot([min(y),max(y)],[min(y),max(y)],'r--', label="Perfect fit")
    plt.xlabel("Actual stress")
    plt.ylabel("Predicted stress")
    plt.title(f"R2: {r2:.2f}")
    plt.legend()
    plt.suptitle(title)
    plt.show()

plot_r2(model, X_train, y_train, title="Training set")
plot_r2(model, X_test, y_test, title="Test set")
```





Individual Shape R^2

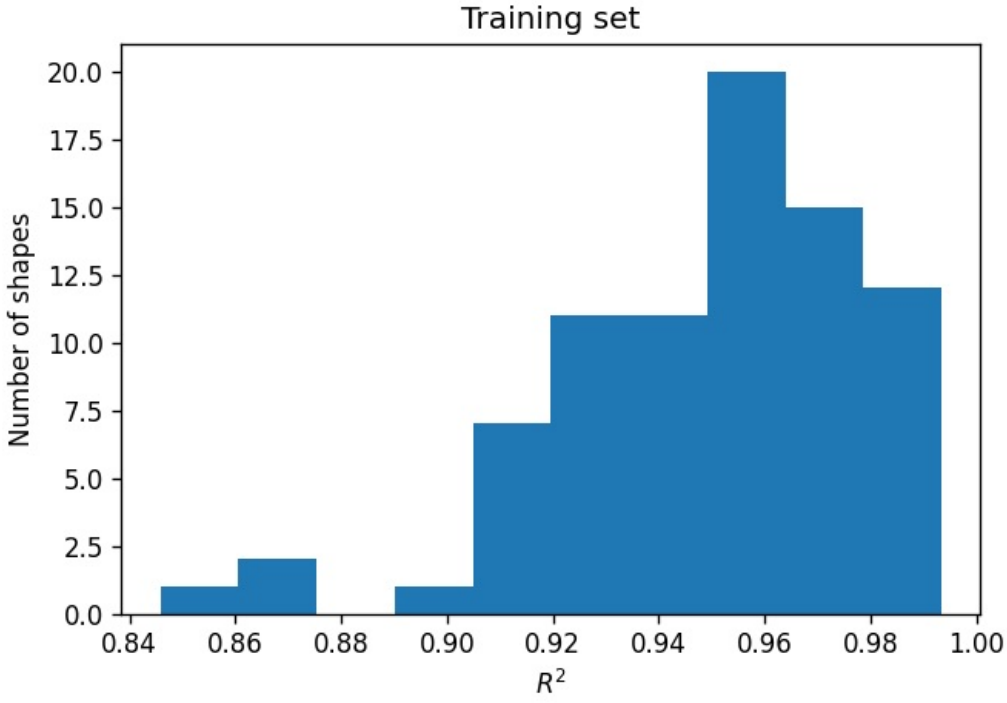
Because we have a unique problem where groups of nodes in a dataset form a single shape, we can compute an R^2 score for an individual shape. For each shape in the training set, compute an R^2 score. Then create a histogram of the values with the function `plot_r2_hist(r2s)`. Repeat for the testing set.

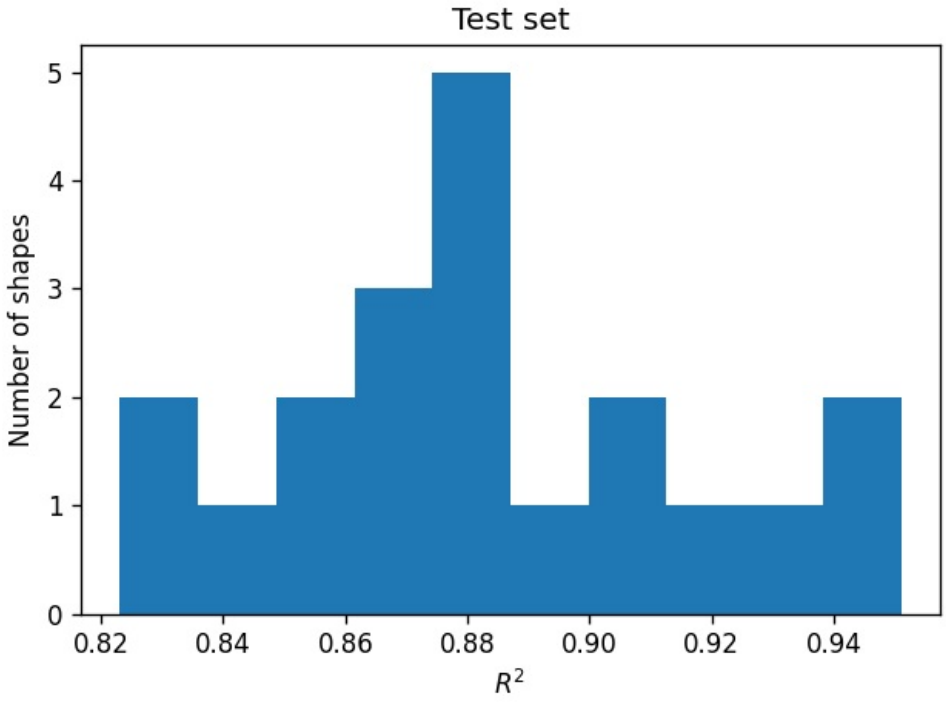
Report the median R^2 score across all training shapes, and the median across all testing shapes.

If your test median is below 0.85, try and tune your network size/training hyperparameters until it reaches this threshold.

In [35]:
```python
# YOUR CODE GOES HERE
def get_r2_distribution(model, dataset):
    r2s = []

    for i in range(len(dataset["features"])):
        X, Y = get_shape(dataset, i)
        y_pred = model(X)
        r2 = r2_score(Y.detach().numpy(), y_pred.detach().numpy())
        r2s.append(r2)

    return r2s

r2s_train = get_r2_distribution(model, dataset_train)
```

```
r2s_test = get_r2_distribution(model, dataset_test)

plot_r2_distribution(r2s_train, title="Training set")
plot_r2_distribution(r2s_test, title="Test set")

print(f"Median R2 score on training set: {np.median(r2s_train)}")
print(f"Median R2 score on test set: {np.median(r2s_test)}")
```






```
Median R2 score on training set: 0.9512203931808472
Median R2 score on test set: 0.8791888654232025
```