Project_NeuralNetwork_implementation-Copy1

July 1, 2025

[1]:
```
%%capture
!pip install tensorflow
!pip install keras_tuner
```

[2]:
```
%%capture

import pandas as pd
import time
import numpy as np
import os
import requests
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.callbacks import EarlyStopping
import keras_tuner as kt
```

```
2025-07-01 17:57:29.613960: I external/local_xla/xla/tsl/cuda/cudart_stub.cc:32]
Could not find cuda drivers on your machine, GPU will not be used.
2025-07-01 17:57:29.620680: I external/local_xla/xla/tsl/cuda/cudart_stub.cc:32]
Could not find cuda drivers on your machine, GPU will not be used.
2025-07-01 17:57:29.636242: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:467] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
WARNING: All log messages before absl::InitializeLog() is called are written to
STDERR
E0000 00:00:1751392649.662243     221 cuda_dnn.cc:8579] Unable to register cuDNN
factory: Attempting to register factory for plugin cuDNN when one has already
been registered
E0000 00:00:1751392649.669664     221 cuda_blas.cc:1407] Unable to register
cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has
already been registered
W0000 00:00:1751392649.689751     221 computation_placer.cc:177] computation
placer already registered. Please check linkage and avoid linking the same
target more than once.
W0000 00:00:1751392649.689785     221 computation_placer.cc:177] computation
placer already registered. Please check linkage and avoid linking the same
```

```
target more than once.
W0000 00:00:1751392649.689788     221 computation_placer.cc:177] computation
placer already registered. Please check linkage and avoid linking the same
target more than once.
W0000 00:00:1751392649.689790     221 computation_placer.cc:177] computation
placer already registered. Please check linkage and avoid linking the same
target more than once.
2025-07-01 17:57:29.696434: I tensorflow/core/platform/cpu_feature_guard.cc:210]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
```

1 Original Data Source

The original data for this project is sourced from https://optiondata.org/. They have graciously provided free sample datasets for option prices and stock prices from January 2013 to June 2013. The data we will be using is from the month of February 2013 located in the `2013-02.zip` file located on their website.

1.1 Data Preprocessing

The dataset we will be using is fairly large, there are 19 trading days in the month of February 2013 and two different csv files for each day, the first being the options data and the second being the stock price data.

To simplify this, we will first join the two datasets through the `symbol` column in the stocks files, and the `underlying` column in the options files. This will allow us to access the data for both in a single file for our features later.

After reviewing the files, we found there were around 500,000 rows for 3800 unique tickers on each trading day, with 19 different days that meant we were looking at close to 9.5 million rows of data. Training machine learning algorithms on this much data will take a very long time, so to minimize this effect, we decided to filter the data to a subset of S&P 100 stocks as of February 1st 2013 and filter the options to call options only, allowing us to train our models within a reasonable time-frame.

Lastly, since our original options data set contained bid and ask prices, we engineered a new feature named `mid_price` which contains the average of the two, and will be evaluated as the price of the option.

Below I've attached a code chunk which displays the filtering we performed. (Please note this won't work on your machine unless you've modified the directories with your own):

```
import numpy as np
import pandas as pd
import os

def combine_options_data(options_data, stock_data):
    options_data['mid_price'] = (options_data['bid'] + options_data['ask'])/2
```

```
    combined_df = options_data.merge(stock_data,
                                     left_on = "underlying", right_on = "symbol",
                                     suffixes=('', '_stock'))
    combined_df = combined_df.drop(columns=['symbol'])
    return combined_df

def combine_new_data(date):
    options = pd.read_csv(f"/home/steve/Downloads/CFRM_521/ProjectData/2013-02/{date}options.cs
    stocks = pd.read_csv(f"/home/steve/Downloads/CFRM_521/ProjectData/2013-02/{date}stocks.csv"
    return combine_options_data(options, stocks)

url = "https://web.archive.org/web/20130201003232/https://en.wikipedia.org/wiki/S%26P_100"
sp100_comp = pd.read_html(url)
sp100_comp = sp100_comp[2]["Symbol"]
sp100_comp = sp100_comp.unique()

dir = "/home/steve/Downloads/CFRM_521/ProjectData/2013-02/"
dates = []
for file in os.listdir(dir):
    #print(file)
    if file.endswith(".csv"):
        if "options" in file:
            dates.append(file.split("options")[0])

dates = sorted(set(dates))
for date in dates:
    print(f"Processing: {date}")
    combined_df = combine_new_data(date)
    combined_df['underlying'] = (
    combined_df['underlying']
    .str.replace('.', '-', regex=False)
    .str.upper()
    .replace({'GOOGL': 'GOOG'})
    )
    sp100_comp = [sym.replace('.', '-').upper() for sym in sp100_comp]
    filt_df = combined_df[combined_df['underlying'].isin(sp100_comp)]
    num_stocks = filt_df['underlying'].unique()
    if len(num_stocks) != len(sp100_comp):
        print(f"Error size mismatch, Filtered Df Size: {len(num_stocks)}, SP100 Size: {len(sp10
        missing = set(sp100_comp) - set(filt_df['underlying'].unique())
        print(f"Missing tickers: {sorted(missing)}")
    else:
        filt_df.to_csv(f"/home/steve/Downloads/CFRM_521/ProjectData/filtered/{date}.csv", inde
```

2 Loading in Data

The preprocessed data was then uploaded to a github repository to allow all team-members to access the new data. The code below is used to import the data into the jupyter notebook and then randomize the order in which the rows appear.

[3]:
```
url = "https://api.github.com/repos/stevedemirev/CFRM521-ProjectData/contents/
↪filtered"
response = requests.get(url)
files = response.json()

csv_files = sorted([file for file in files if file['name'].endswith('.csv')],␣
↪key = lambda x: x['name'])

def get_datasets(files):
    df = pd.DataFrame()
    for file in files:
        temp = pd.read_csv(file['download_url'])
        df = pd.concat([df, temp], ignore_index = True)
    return df

full_df = get_datasets(csv_files)
full_df = full_df.sample(frac=1, random_state=42).reset_index(drop=True)

total_files = len(full_df)
train_size = int(total_files*0.7)
val_size = int(total_files*0.85)

train = full_df[:train_size]
valid = full_df[train_size:val_size]
test = full_df[val_size:]
```

[4]:
```
display(train.head())
```

```
             contract underlying  expiration  type  strike style     bid  \
0     V130921C00110000          V  2013-09-21  call   110.0     A   48.10
1  GOOG150117C00370000       GOOG  2015-01-17  call   370.0     A  415.00
2  AMZN150117P00240000       AMZN  2015-01-17   put   240.0     A   36.15
3  AAPL130720P00630000       AAPL  2013-07-20   put   630.0     A  189.45
4   BAC130322C00013500        BAC  2013-03-22  call    13.5     A    0.00

   bid_size     ask  ask_size  …    theta      vega implied_volatility  \
0       NaN   48.75       NaN  …  -1.5975    6.2814             0.2444
1       NaN  419.40       NaN  …  -4.2084   43.9866             0.2792
2       NaN   37.05       NaN  … -11.5955  130.5729             0.3374
3       NaN  190.40       NaN  … -11.9728   21.8567             0.2993
4       NaN    0.02       NaN  …  -0.5092    0.1719             0.3651
```

```
   mid_price        open        high         low       close  volume_stock  \
0     48.425  156.350006  158.080002  155.740005  157.990005      17884400
1    417.200  778.400030  783.000040  773.750022  782.419993       4331200
2     36.600  260.890015  262.040009  255.729996  259.359985       3348600
3    189.925  453.850014  455.120003  442.569996  442.799988      93144800
4      0.010   11.150000   11.360000   11.100000   11.300000     147145400

   adjust_close
0           NaN
1           NaN
2           NaN
3           NaN
4           NaN

[5 rows x 25 columns]
```

[5]:
```
display(train.tail())
```

```
                   contract underlying  expiration  type  strike style    bid  \
483143  COST140118P00100000       COST  2014-01-18   put  100.00     A   6.50
483144   MCD140118C00085000        MCD  2014-01-18  call   85.00     A  10.60
483145   EMR140118C00025000        EMR  2014-01-18  call   25.00     A  30.70
483146  GILD130817P00036250       GILD  2013-08-17   put   36.25     A   1.33
483147    GD140118C00040000         GD  2014-01-18  call   40.00     A  27.40

        bid_size    ask  ask_size  …   theta     vega implied_volatility  \
483143       NaN   6.70       NaN  … -3.8113  38.6961             0.1933
483144       NaN  10.80       NaN  … -1.0126  27.5612             0.1603
483145       NaN  33.70       NaN  … -0.6678   5.9553             0.6433
483146       NaN   1.38       NaN  … -2.5770   9.1628             0.2868
483147       NaN  29.40       NaN  … -0.4518   2.2104             0.2716

        mid_price        open        high         low       close  \
483143      6.600  102.879997  103.190002  100.940002  101.699997
483144     10.700   94.269997   95.250000   93.849998   95.250000
483145     32.200   57.330002   57.509998   56.840000   57.470001
483146      1.355   40.459999   40.950001   40.299999   40.830002
483147     28.400   68.449997   69.040001   67.959999   67.970001

        volume_stock  adjust_close
483143       3192000           NaN
483144       4421300           NaN
483145       2912700           NaN
483146       9791000           NaN
483147       2338200           NaN

[5 rows x 25 columns]
```

[6]:
```
total_size = len(train) + len(valid) + len(test)
def proportion(df):
    return round(len(df)/total_size,3)

print(f"Length of Training set: {len(train):,} rows, Proportion:␣
 ↪{proportion(train)}")
print(f"Length of Validation set: {len(valid):,} rows, Proportion:␣
 ↪{proportion(valid)}")
print(f"Length of Testing set: {len(test):,} rows, Proportion:␣
 ↪{proportion(test)}")
print(f"Original Dataset size: {total_size:,} rows, Sum Check:␣
 ↪{proportion(train)+proportion(valid)+proportion(test)}")
```

```
Length of Training set: 483,148 rows, Proportion: 0.7
Length of Validation set: 103,532 rows, Proportion: 0.15
Length of Testing set: 103,532 rows, Proportion: 0.15
Original Dataset size: 690,212 rows, Sum Check: 1.0
```

3 Baseline Model

For my implementation, I will be using the `close`, `strike`, `delta`, `gamma`, `vega`, `theta`, and `implied_volatility` columns as features. I also engineered `tte` (time to expiry) as it's an important parameter used in one the most common methods of pricing options, the Black-Scholes formula.

For my model's architecture, I decided to use 4 hidden layers with 50 neurons each with 'ReLU' activation, 'he_normal' initializer, 'Nadam' optimizer, early stopping with a patience of 10, and "MSE" as the primary loss metric, and "MAE" as the secondary loss metric. The choice to use 4 hidden layers with 50 neurons was somewhat arbitrary and not backed by any theoretical justification other than the expectation of it being able to accurately capture any non-linear relationships occuring with options pricing. The output layer would remain the default linear activation function with one neuron as we are expecting a single value, the mid price. I also chose the "ReLU" activation function for each neuron with "he_normal" initializer to better model the non-linear relationship and avoid vanishing gradients during training. Additionally, "ReLU" works well because it encourages positive only outputs, which matches with the intrinsic value of an option is defined as: $V = \max(S_T - K, 0)$. The 'Nadam' optimizer was chosen here as its learning rate is adaptable, allowing it to converge faster and smoother for non-linear functions compared to other optimizers. Early stopping was added as a form of regularization to prevent unnecessary training once the model stops improving on the validation set. Given the time required to train the model, early stopping is an effective way to reduce training time while also helping to avoid overfitting. Since the objective of this project is to predict option prices, Mean Squared Error (MSE) and Mean Absolute Error (MAE) are appropriate evaluation metrics. MSE penalizes larger errors more heavily, making it useful for identifying significant prediction deviations, while MAE provides a more interpretable measure of the average prediction error.

[7]:
```
def define_features(df):
    df['expiration'] = pd.to_datetime(df['expiration'])
```

```
    df['quote_date'] = pd.to_datetime(df['quote_date'])
    df['tte'] = (df['expiration'] - df['quote_date']).dt.days / 252
    X = df[['close', 'strike', 'delta', 'gamma',
            'vega', 'theta', 'implied_volatility', 'tte']]
    y = df['mid_price']
    return X, y

def get_features(opt_type):
    X_feats = []
    y_feats = []
    for df in [train, valid, test]:
        df_temp = df[df['type'] == opt_type].copy()
        X, y = define_features(df_temp)
        X_feats.append(X)
        y_feats.append(y)
    return X_feats, y_feats

# Calls
X_feats, y_feats = get_features("call")
X_train_c, X_valid_c, X_test_c = X_feats
y_train_c, y_valid_c, y_test_c = y_feats

# Puts
X_feats, y_feats = get_features("put")
X_train_p, X_valid_p, X_test_p = X_feats
y_train_p, y_valid_p, y_test_p = y_feats
```

[8]:
```
def scale_data(train, val, test):
    scaler = StandardScaler()
    train_scaled = scaler.fit_transform(train)
    valid_scaled = scaler.transform(val)
    test_scaled = scaler.transform(test)
    return train_scaled, valid_scaled, test_scaled

def reset_session(seed=42):
    tf.keras.backend.clear_session()
    tf.random.set_seed(seed)
    np.random.seed(seed)

def build_model(input_shape):
    reset_session()
    model = tf.keras.Sequential([
        tf.keras.Input(shape = (input_shape,)),
        tf.keras.layers.Dense(50, activation = "relu", kernel_initializer =␣
 ↪"he_normal"),
        tf.keras.layers.Dense(50, activation = "relu", kernel_initializer =␣
 ↪"he_normal"),
```

```
        tf.keras.layers.Dense(50, activation = "relu", kernel_initializer =␣
 ↪"he_normal"),
        tf.keras.layers.Dense(50, activation = "relu", kernel_initializer =␣
 ↪"he_normal"),
        tf.keras.layers.Dense(1)
    ])

    model.compile(
        optimizer = "nadam",
        loss = "mse",
        metrics = ['mae']
    )
    return model

X_train_scaled_c, X_valid_scaled_c, X_test_scaled_c = scale_data(X_train_c,␣
 ↪X_valid_c, X_test_c)
X_train_scaled_p, X_valid_scaled_p, X_test_scaled_p = scale_data(X_train_p,␣
 ↪X_valid_p, X_test_p)

early_stop = EarlyStopping(
    monitor = "val_loss",
    patience = 10,
    mode = "min",
    restore_best_weights = True
)
```

3.1 Call Option Model

[9]:
```
model_calls = build_model(X_train_scaled_c.shape[1])

start = time.time()
history_calls = model_calls.fit(X_train_scaled_c, y_train_c,
             validation_data = (X_valid_scaled_c, y_valid_c),
             epochs = 50, verbose = 1, callbacks=[early_stop])
end = time.time()
baseline_time_c = end-start

mse_val_c, mae_val_c = model_calls.evaluate(X_valid_scaled_c,
                                            y_valid_c, verbose = 0)
test_loss_c, test_mae_c = model_calls.evaluate(X_test_scaled_c,
                                               y_test_c, verbose = 0)

print("\nCall Options:")
print(f"Validation MSE: {mse_val_c}")
print(f"Validation MAE: {mae_val_c}")
```

```python
print(f"\nTest MSE: {test_loss_c}")
print(f"Test MAE: {test_mae_c}")
```

```
Epoch 1/50
```

```
2025-07-01 17:57:51.221715: E
external/local_xla/xla/stream_executor/cuda/cuda_platform.cc:51] failed call to
cuInit: INTERNAL: CUDA error: Failed call to cuInit: UNKNOWN ERROR (303)
```

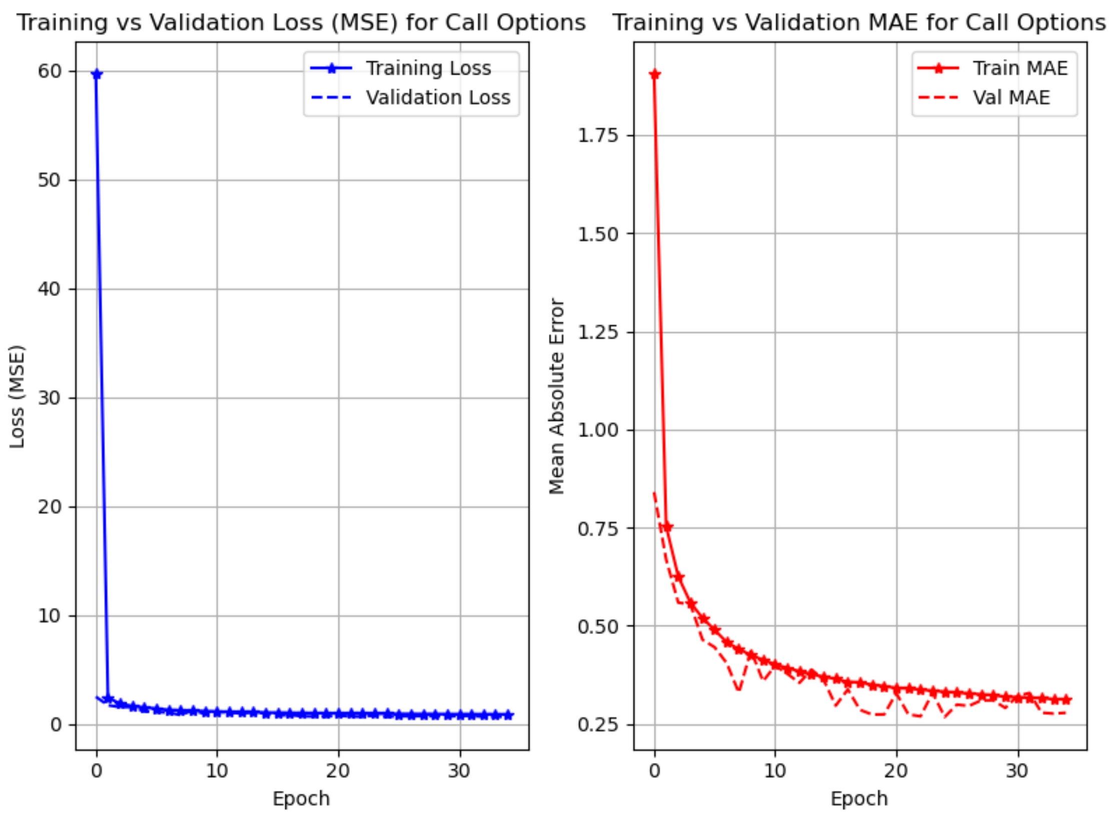
```
7549/7549              22s 3ms/step -
loss: 276.4209 - mae: 4.1048 - val_loss: 2.5094 - val_mae: 0.8404
Epoch 2/50
7549/7549              19s 2ms/step -
loss: 2.6091 - mae: 0.8065 - val_loss: 1.7289 - val_mae: 0.6684
Epoch 3/50
7549/7549              18s 2ms/step -
loss: 1.8883 - mae: 0.6353 - val_loss: 1.5641 - val_mae: 0.5589
Epoch 4/50
7549/7549              19s 2ms/step -
loss: 1.6471 - mae: 0.5632 - val_loss: 1.5665 - val_mae: 0.5549
Epoch 5/50
7549/7549              18s 2ms/step -
loss: 1.5697 - mae: 0.5330 - val_loss: 1.1716 - val_mae: 0.4646
Epoch 6/50
7549/7549              19s 2ms/step -
loss: 1.4368 - mae: 0.4939 - val_loss: 1.1193 - val_mae: 0.4454
Epoch 7/50
7549/7549              19s 3ms/step -
loss: 1.3244 - mae: 0.4629 - val_loss: 1.0621 - val_mae: 0.4049
Epoch 8/50
7549/7549              21s 3ms/step -
loss: 1.3028 - mae: 0.4508 - val_loss: 0.8277 - val_mae: 0.3295
Epoch 9/50
7549/7549              20s 3ms/step -
loss: 1.2889 - mae: 0.4364 - val_loss: 1.2629 - val_mae: 0.4374
Epoch 10/50
7549/7549              22s 3ms/step -
loss: 1.2268 - mae: 0.4206 - val_loss: 0.9297 - val_mae: 0.3587
Epoch 11/50
7549/7549              19s 3ms/step -
loss: 1.1992 - mae: 0.4104 - val_loss: 1.1181 - val_mae: 0.3986
Epoch 12/50
7549/7549              20s 3ms/step -
loss: 1.1733 - mae: 0.3992 - val_loss: 1.0529 - val_mae: 0.3779
Epoch 13/50
7549/7549              19s 3ms/step -
loss: 1.1645 - mae: 0.3946 - val_loss: 0.9847 - val_mae: 0.3546
Epoch 14/50
```

```
7549/7549              18s 2ms/step -
loss: 1.1477 - mae: 0.3882 - val_loss: 1.0925 - val_mae: 0.3883
Epoch 15/50
7549/7549              18s 2ms/step -
loss: 1.1308 - mae: 0.3830 - val_loss: 0.9918 - val_mae: 0.3623
Epoch 16/50
7549/7549              19s 2ms/step -
loss: 1.1146 - mae: 0.3762 - val_loss: 0.7738 - val_mae: 0.2967
Epoch 17/50
7549/7549              19s 2ms/step -
loss: 1.0782 - mae: 0.3666 - val_loss: 0.8522 - val_mae: 0.3380
Epoch 18/50
7549/7549              18s 2ms/step -
loss: 1.0807 - mae: 0.3669 - val_loss: 0.7166 - val_mae: 0.2859
Epoch 19/50
7549/7549              18s 2ms/step -
loss: 1.0669 - mae: 0.3602 - val_loss: 0.6996 - val_mae: 0.2739
Epoch 20/50
7549/7549              18s 2ms/step -
loss: 1.0474 - mae: 0.3564 - val_loss: 0.6927 - val_mae: 0.2743
Epoch 21/50
7549/7549              18s 2ms/step -
loss: 1.0251 - mae: 0.3493 - val_loss: 0.7833 - val_mae: 0.3282
Epoch 22/50
7549/7549              18s 2ms/step -
loss: 1.0163 - mae: 0.3480 - val_loss: 0.6972 - val_mae: 0.2742
Epoch 23/50
7549/7549              18s 2ms/step -
loss: 1.0194 - mae: 0.3477 - val_loss: 0.6800 - val_mae: 0.2694
Epoch 24/50
7549/7549              18s 2ms/step -
loss: 1.0018 - mae: 0.3427 - val_loss: 0.7904 - val_mae: 0.3294
Epoch 25/50
7549/7549              18s 2ms/step -
loss: 0.9894 - mae: 0.3389 - val_loss: 0.6555 - val_mae: 0.2678
Epoch 26/50
7549/7549              19s 3ms/step -
loss: 0.9811 - mae: 0.3374 - val_loss: 0.7301 - val_mae: 0.2992
Epoch 27/50
7549/7549              21s 3ms/step -
loss: 0.9760 - mae: 0.3349 - val_loss: 0.7146 - val_mae: 0.2959
Epoch 28/50
7549/7549              20s 3ms/step -
loss: 0.9647 - mae: 0.3320 - val_loss: 0.7400 - val_mae: 0.3099
Epoch 29/50
7549/7549              19s 3ms/step -
loss: 0.9550 - mae: 0.3287 - val_loss: 0.7331 - val_mae: 0.3103
Epoch 30/50
```

```
7549/7549              19s 3ms/step -
loss: 0.9483 - mae: 0.3252 - val_loss: 0.6975 - val_mae: 0.2908
Epoch 31/50
7549/7549              19s 3ms/step -
loss: 0.9366 - mae: 0.3220 - val_loss: 0.7476 - val_mae: 0.3209
Epoch 32/50
7549/7549              18s 2ms/step -
loss: 0.9357 - mae: 0.3210 - val_loss: 0.7601 - val_mae: 0.3289
Epoch 33/50
7549/7549              20s 3ms/step -
loss: 0.9309 - mae: 0.3204 - val_loss: 0.6711 - val_mae: 0.2786
Epoch 34/50
7549/7549              19s 3ms/step -
loss: 0.9311 - mae: 0.3188 - val_loss: 0.6766 - val_mae: 0.2764
Epoch 35/50
7549/7549              20s 3ms/step -
loss: 0.9290 - mae: 0.3192 - val_loss: 0.6760 - val_mae: 0.2784

Call Options:
Validation MSE: 0.6554962396621704
Validation MAE: 0.2678316831588745

Test MSE: 0.8441799283027649
Test MAE: 0.26972681283950806
```

[10]:
```python
def plot_learning_curves(history, opt_type):
    fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(8, 6))

    ax1.plot(history.epoch, history.history["loss"], '*-', label="Training␣
↪Loss", color='b')
    ax1.plot(history.epoch, history.history["val_loss"], '--',␣
↪label="Validation Loss", color='b')
    ax1.set_title(f"Training vs Validation Loss (MSE) for {opt_type} Options")
    ax1.set_xlabel("Epoch")
    ax1.set_ylabel("Loss (MSE)")
    ax1.grid(True)
    ax1.legend()

    ax2.plot(history.epoch, history.history["mae"], '*-', label="Train MAE",␣
↪color='r')
    ax2.plot(history.epoch, history.history["val_mae"], '--', label="Val MAE",␣
↪color='r')
    ax2.set_title(f"Training vs Validation MAE for {opt_type} Options")
    ax2.set_xlabel("Epoch")
    ax2.set_ylabel("Mean Absolute Error")
    ax2.grid(True)
    ax2.legend()
```

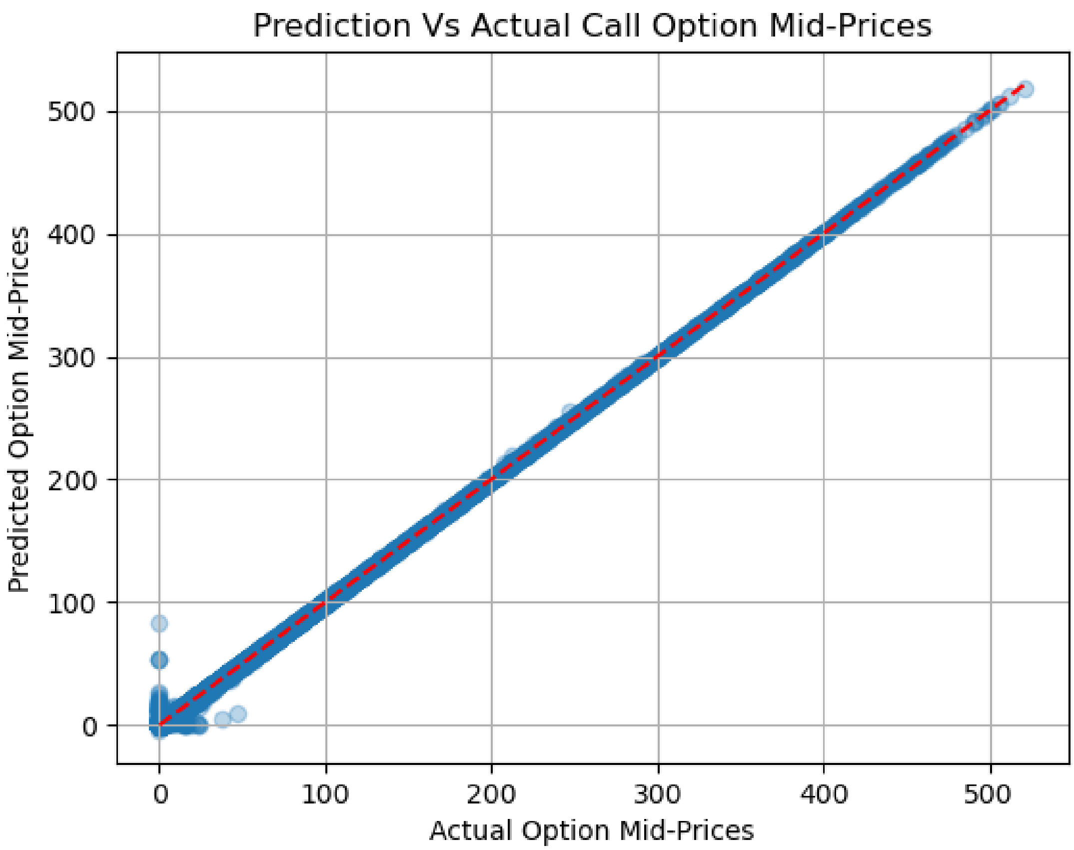
```
    plt.tight_layout()
    plt.show()

plot_learning_curves(history_calls, "Call")
```



[11]:

```
def get_prediction_plot(model, X_test_s, y_test, opt_type):
    y_pred = model.predict(X_test_s)

    plt.scatter(y_test, y_pred, marker = 'o', alpha = 0.3)
    plt.plot([y_test.min(), y_test.max()],
             [y_test.min(), y_test.max()], 'r--')
    plt.xlabel("Actual Option Mid-Prices")
    plt.ylabel("Predicted Option Mid-Prices")
    plt.title(f"Prediction Vs Actual {opt_type} Option Mid-Prices")
    plt.grid(True)
    plt.show()

get_prediction_plot(model_calls, X_test_scaled_c,
                    y_test_c, "Call")
```

```
1617/1617              2s 1ms/step
```



3.2 Put option Model

[12]:
```
model_puts = build_model(X_train_scaled_p.shape[1])

start = time.time()
history_puts = model_puts.fit(X_train_scaled_p, y_train_p,
             validation_data = (X_valid_scaled_p, y_valid_p),
             epochs = 50, verbose = 1, callbacks=[early_stop])
end = time.time()
baseline_time_p = end-start

mse_val_p, mae_val_p = model_puts.evaluate(X_valid_scaled_p,
                                           y_valid_p, verbose = 0)
test_loss_p, test_mae_p = model_puts.evaluate(X_test_scaled_p,
                                              y_test_p, verbose = 0)
print("\nPut Options:")
print(f"Validation MSE: {mse_val_p}")
print(f"Validation MAE: {mae_val_p}")
```

```python
print(f"\nTest MSE: {test_loss_p}")
print(f"Test MAE: {test_mae_p}")
```

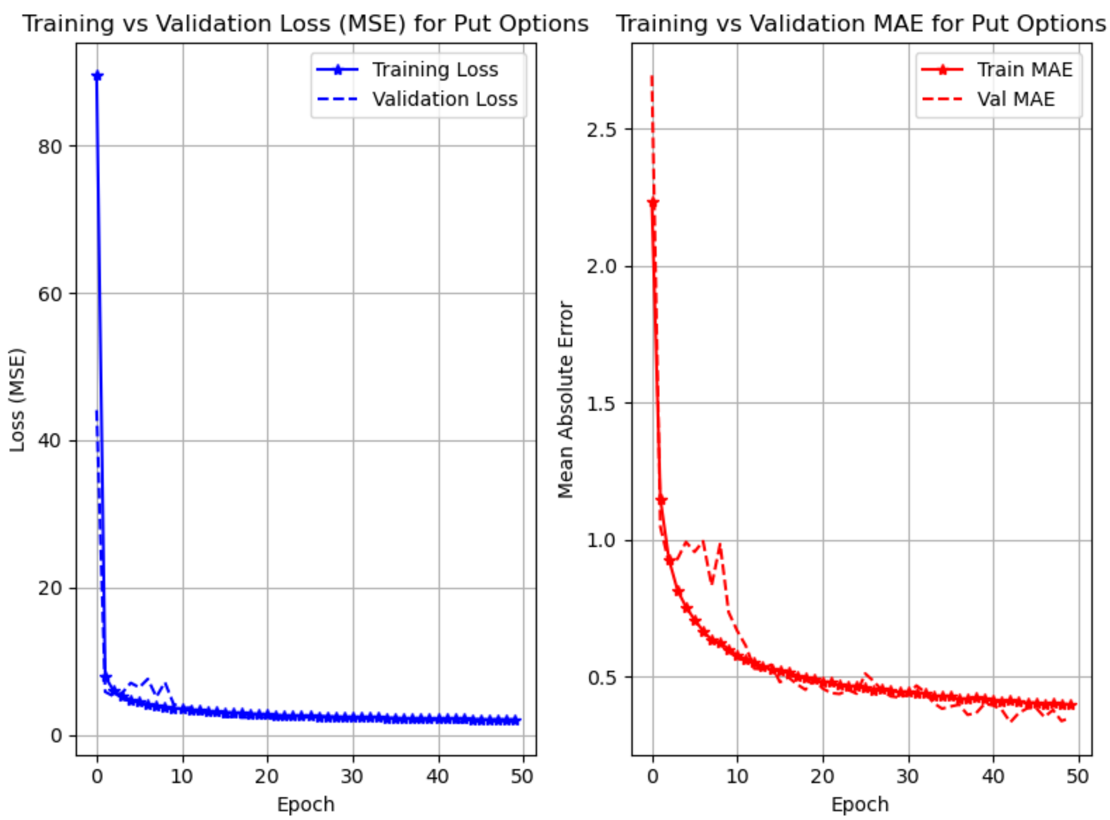
```
Epoch 1/50
7550/7550              22s 3ms/step -
loss: 386.3810 - mae: 4.3048 - val_loss: 44.0751 - val_mae: 2.6941
Epoch 2/50
7550/7550              19s 2ms/step -
loss: 9.3228 - mae: 1.2542 - val_loss: 5.8460 - val_mae: 1.0468
Epoch 3/50
7550/7550              19s 3ms/step -
loss: 6.2965 - mae: 0.9776 - val_loss: 5.2344 - val_mae: 0.9160
Epoch 4/50
7550/7550              19s 2ms/step -
loss: 5.4064 - mae: 0.8472 - val_loss: 5.7089 - val_mae: 0.9302
Epoch 5/50
7550/7550              19s 3ms/step -
loss: 4.7906 - mae: 0.7692 - val_loss: 7.0501 - val_mae: 0.9922
Epoch 6/50
7550/7550              20s 3ms/step -
loss: 4.6544 - mae: 0.7236 - val_loss: 6.4352 - val_mae: 0.9558
Epoch 7/50
7550/7550              18s 2ms/step -
loss: 4.2419 - mae: 0.6785 - val_loss: 7.5897 - val_mae: 0.9951
Epoch 8/50
7550/7550              18s 2ms/step -
loss: 4.0587 - mae: 0.6474 - val_loss: 5.1543 - val_mae: 0.8332
Epoch 9/50
7550/7550              18s 2ms/step -
loss: 3.9256 - mae: 0.6358 - val_loss: 7.1388 - val_mae: 0.9846
Epoch 10/50
7550/7550              18s 2ms/step -
loss: 3.7227 - mae: 0.6076 - val_loss: 4.2217 - val_mae: 0.7353
Epoch 11/50
7550/7550              18s 2ms/step -
loss: 3.6603 - mae: 0.5964 - val_loss: 3.5941 - val_mae: 0.6683
Epoch 12/50
7550/7550              19s 2ms/step -
loss: 3.6692 - mae: 0.5867 - val_loss: 3.6855 - val_mae: 0.6136
Epoch 13/50
7550/7550              20s 3ms/step -
loss: 3.4678 - mae: 0.5648 - val_loss: 2.8578 - val_mae: 0.5301
Epoch 14/50
7550/7550              19s 2ms/step -
loss: 3.1924 - mae: 0.5359 - val_loss: 3.0826 - val_mae: 0.5284
Epoch 15/50
```

```
7550/7550             19s 3ms/step -
loss: 3.2803 - mae: 0.5406 - val_loss: 2.8242 - val_mae: 0.5444
Epoch 16/50
7550/7550             20s 3ms/step -
loss: 3.0732 - mae: 0.5232 - val_loss: 2.4605 - val_mae: 0.4808
Epoch 17/50
7550/7550             20s 3ms/step -
loss: 3.0614 - mae: 0.5224 - val_loss: 2.4961 - val_mae: 0.4985
Epoch 18/50
7550/7550             20s 3ms/step -
loss: 2.8887 - mae: 0.5019 - val_loss: 2.3405 - val_mae: 0.4731
Epoch 19/50
7550/7550             20s 3ms/step -
loss: 2.8594 - mae: 0.4972 - val_loss: 2.3819 - val_mae: 0.4531
Epoch 20/50
7550/7550             20s 3ms/step -
loss: 2.8463 - mae: 0.4900 - val_loss: 2.4343 - val_mae: 0.4932
Epoch 21/50
7550/7550             20s 3ms/step -
loss: 2.7934 - mae: 0.4841 - val_loss: 2.2943 - val_mae: 0.4580
Epoch 22/50
7550/7550             21s 3ms/step -
loss: 2.7122 - mae: 0.4799 - val_loss: 2.2395 - val_mae: 0.4412
Epoch 23/50
7550/7550             21s 3ms/step -
loss: 2.6786 - mae: 0.4718 - val_loss: 2.2519 - val_mae: 0.4378
Epoch 24/50
7550/7550             22s 3ms/step -
loss: 2.7113 - mae: 0.4678 - val_loss: 2.3175 - val_mae: 0.4527
Epoch 25/50
7550/7550             23s 3ms/step -
loss: 2.6975 - mae: 0.4697 - val_loss: 2.1978 - val_mae: 0.4388
Epoch 26/50
7550/7550             21s 3ms/step -
loss: 2.7644 - mae: 0.4762 - val_loss: 2.5314 - val_mae: 0.5120
Epoch 27/50
7550/7550             20s 3ms/step -
loss: 2.6065 - mae: 0.4600 - val_loss: 2.6210 - val_mae: 0.4839
Epoch 28/50
7550/7550             20s 3ms/step -
loss: 2.6119 - mae: 0.4622 - val_loss: 2.2935 - val_mae: 0.4496
Epoch 29/50
7550/7550             19s 3ms/step -
loss: 2.5467 - mae: 0.4572 - val_loss: 2.2713 - val_mae: 0.4291
Epoch 30/50
7550/7550             19s 3ms/step -
loss: 2.5296 - mae: 0.4528 - val_loss: 2.0601 - val_mae: 0.4250
Epoch 31/50
```

```
7550/7550             19s 3ms/step -
loss: 2.5385 - mae: 0.4474 - val_loss: 2.2361 - val_mae: 0.4351
Epoch 32/50
7550/7550             19s 3ms/step -
loss: 2.4830 - mae: 0.4436 - val_loss: 2.2456 - val_mae: 0.4683
Epoch 33/50
7550/7550             19s 3ms/step -
loss: 2.5024 - mae: 0.4475 - val_loss: 2.1952 - val_mae: 0.4444
Epoch 34/50
7550/7550             20s 3ms/step -
loss: 2.3724 - mae: 0.4337 - val_loss: 2.2242 - val_mae: 0.4049
Epoch 35/50
7550/7550             19s 3ms/step -
loss: 2.3864 - mae: 0.4283 - val_loss: 2.0833 - val_mae: 0.3840
Epoch 36/50
7550/7550             19s 2ms/step -
loss: 2.3655 - mae: 0.4325 - val_loss: 1.9880 - val_mae: 0.3919
Epoch 37/50
7550/7550             20s 3ms/step -
loss: 2.2759 - mae: 0.4242 - val_loss: 1.9845 - val_mae: 0.3989
Epoch 38/50
7550/7550             23s 3ms/step -
loss: 2.3150 - mae: 0.4261 - val_loss: 1.8645 - val_mae: 0.3617
Epoch 39/50
7550/7550             20s 3ms/step -
loss: 2.3536 - mae: 0.4332 - val_loss: 1.8498 - val_mae: 0.3678
Epoch 40/50
7550/7550             19s 3ms/step -
loss: 2.2434 - mae: 0.4196 - val_loss: 2.0267 - val_mae: 0.4022
Epoch 41/50
7550/7550             19s 2ms/step -
loss: 2.2185 - mae: 0.4168 - val_loss: 1.9775 - val_mae: 0.3958
Epoch 42/50
7550/7550             19s 3ms/step -
loss: 2.2516 - mae: 0.4143 - val_loss: 1.8752 - val_mae: 0.3801
Epoch 43/50
7550/7550             20s 3ms/step -
loss: 2.2185 - mae: 0.4149 - val_loss: 1.7228 - val_mae: 0.3343
Epoch 44/50
7550/7550             19s 3ms/step -
loss: 2.2425 - mae: 0.4182 - val_loss: 1.7470 - val_mae: 0.3686
Epoch 45/50
7550/7550             19s 3ms/step -
loss: 2.1616 - mae: 0.4100 - val_loss: 1.8580 - val_mae: 0.3856
Epoch 46/50
7550/7550             19s 2ms/step -
loss: 2.2155 - mae: 0.4110 - val_loss: 1.8101 - val_mae: 0.3874
Epoch 47/50
```

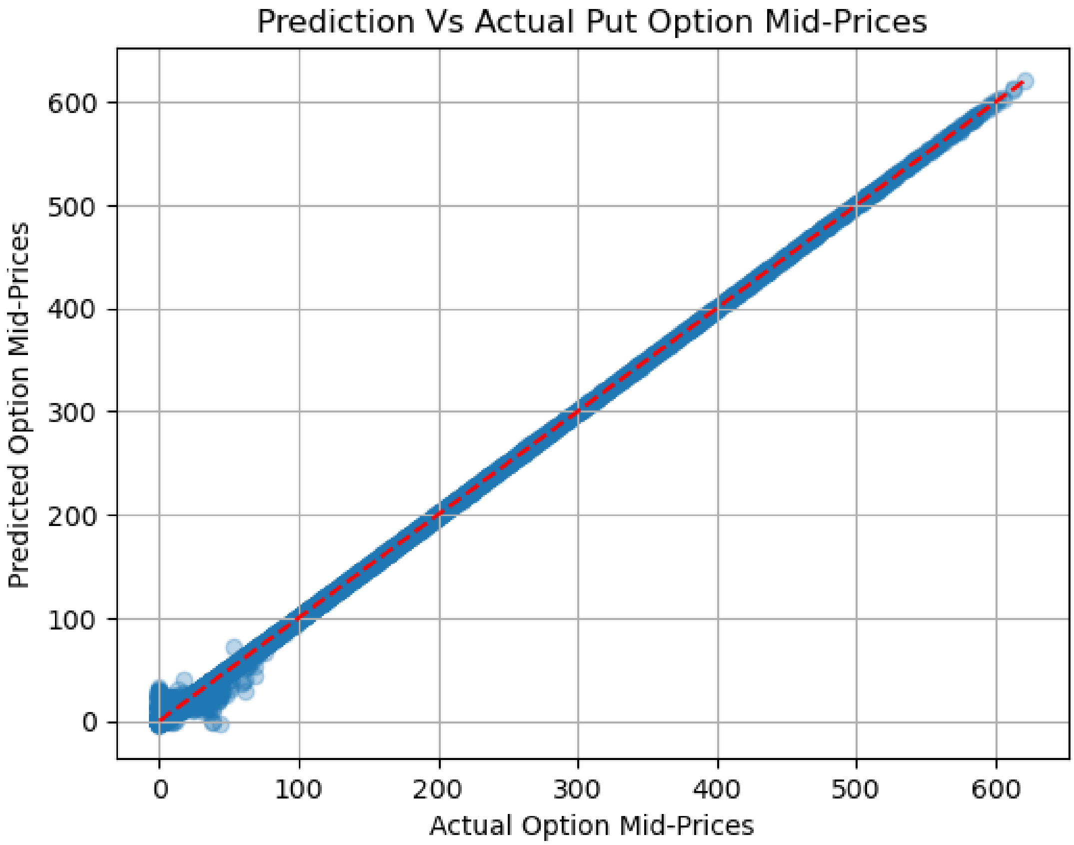
```
7550/7550              19s 3ms/step -
loss: 2.1695 - mae: 0.4067 - val_loss: 1.7064 - val_mae: 0.3533
Epoch 48/50
7550/7550              20s 3ms/step -
loss: 2.1098 - mae: 0.4070 - val_loss: 1.9816 - val_mae: 0.3794
Epoch 49/50
7550/7550              20s 3ms/step -
loss: 2.1072 - mae: 0.4068 - val_loss: 1.7969 - val_mae: 0.3390
Epoch 50/50
7550/7550              18s 2ms/step -
loss: 2.0907 - mae: 0.3986 - val_loss: 1.7812 - val_mae: 0.3507

Put Options:
Validation MSE: 1.7063614130020142
Validation MAE: 0.3533361852169037

Test MSE: 1.6476459503173828
Test MAE: 0.35210031270980835
```

[13]:
```
plot_learning_curves(history_puts, "Put")
```



[14]:
```
get_prediction_plot(model_puts, X_test_scaled_p, y_test_p, "Put")
```

```
1619/1619            2s 1ms/step
```



3.3 Hyperparameter search

To optimize the Neural Network model, we want to identify the best hyperparameter configuration which predicts the option's `mid_price` closest, to do this I will implement a randomized search.

A few of the hyperparameters to optimize are: * Number of Hidden Layers * Number of Neurons per layer * Neuron activation function * Neuron l2 regularization * Learning rate * Optimizer

To find the best configuration, we will select the number of hidden layers between 1 and 5, as adding more may be redundant. For the number of neurons, we will select a value between 10 and 100 neurons. Since deep neural networks often perform better with 'swish' activation rather than with 'ReLU', we will select the activation function between those two at random to assess which performs better. Our l2 regularizer and learning rate will be chosen from a range between 1e-6 and 1e-2 to allow a broader range of values in which to assess performance on. Lastly, the optimizer will be selected at random as either the 'Adam' optimizer or the adam optimizer with nesterov momentum ('Nadam') to assess which fits the model better. Once again we will include early stopping to avoid unneccessary training, but this time with a patience of 5, as we will only be training on 20 epochs instead of 50 due to time constraints. This randomized search will run for 30 iterations in order to assess the optimal parameters, which we will then compare the best model here against the baseline model on their test set performance.

3.3.1 Call Option

[28]:
```
def reset_session(seed=42):
    tf.random.set_seed(seed)
    np.random.seed(seed)
    tf.keras.backend.clear_session()

def build_model(hp):
    reset_session()

    model = tf.keras.Sequential()
    model.add(tf.keras.Input(shape = (X_train_c.shape[1],)))

    n_hidden = hp.Int("n_hidden", 1, 5)
    for i in range(n_hidden):
        model.add(tf.keras.layers.Dense(
            hp.Int(f"n_neurons_{i+1}", 10, 100),
            activation = hp.Choice("activation", ["relu", 'swish']),
            kernel_regularizer = tf.keras.regularizers.l2(
            hp.Float("l2", 1e-6, 1e-4, sampling="log")
            ),
            kernel_initializer = "he_normal"
        ))

    model.add(tf.keras.layers.Dense(1))

    learning_rate = hp.Float("learning_rate", 1e-4, 1e-2, sampling = "log")
    optimizer_name = hp.Choice("optimizer", ["Adam", "Nadam"])

    if optimizer_name == "Adam":
        optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
    else:
        optimizer = tf.keras.optimizers.Nadam(learning_rate=learning_rate)

    model.compile(loss="mse", metrics = ['mae'], optimizer=optimizer)
    return model

early_stop = EarlyStopping(
    monitor = "val_loss",
    patience = 10,
    mode = "min",
    restore_best_weights = True
)
```

[16]:
```
# Train on subsets of training data for speed
X_train_c_sub = X_train_scaled_c[::5]
X_valid_c_sub = X_valid_scaled_c[::5]

y_train_c_sub = y_train_c[::5]
y_valid_c_sub = y_valid_c[::5]

random_search_tuner = kt.RandomSearch(
    build_model,
    objective = "val_loss",
    seed = 42,
    max_trials = 30,
    overwrite = True
)

random_search_tuner.search(X_train_c_sub, y_train_c_sub, epochs = 30,
                    validation_data = (X_valid_c_sub, y_valid_c_sub), verbose =
 ↪1,
                          callbacks=[early_stop])
```

```
Trial 30 Complete [00h 02m 18s]
val_loss: 0.7884814739227295

Best val_loss So Far: 0.77161705493927
Total elapsed time: 00h 49m 21s
```

[17]:
```
best_hyperparams_c = random_search_tuner.get_best_hyperparameters(num_trials =
 ↪1)[0]
best_hyperparams_c.values
```

[17]:
```
{'n_hidden': 2,
 'n_neurons_1': 72,
 'activation': 'relu',
 'l2': 4.938970115853091e-06,
 'learning_rate': 0.002220709089529535,
 'optimizer': 'Nadam',
 'n_neurons_2': 82,
 'n_neurons_3': 20,
 'n_neurons_4': 88,
 'n_neurons_5': 72}
```

[29]:
```
best_model_c = build_model(best_hyperparams_c)

start = time.time()
history_best_c = best_model_c.fit(X_train_scaled_c, y_train_c,
             validation_data = (X_valid_scaled_c, y_valid_c),
             epochs = 50, verbose = 1, callbacks=[early_stop])
```

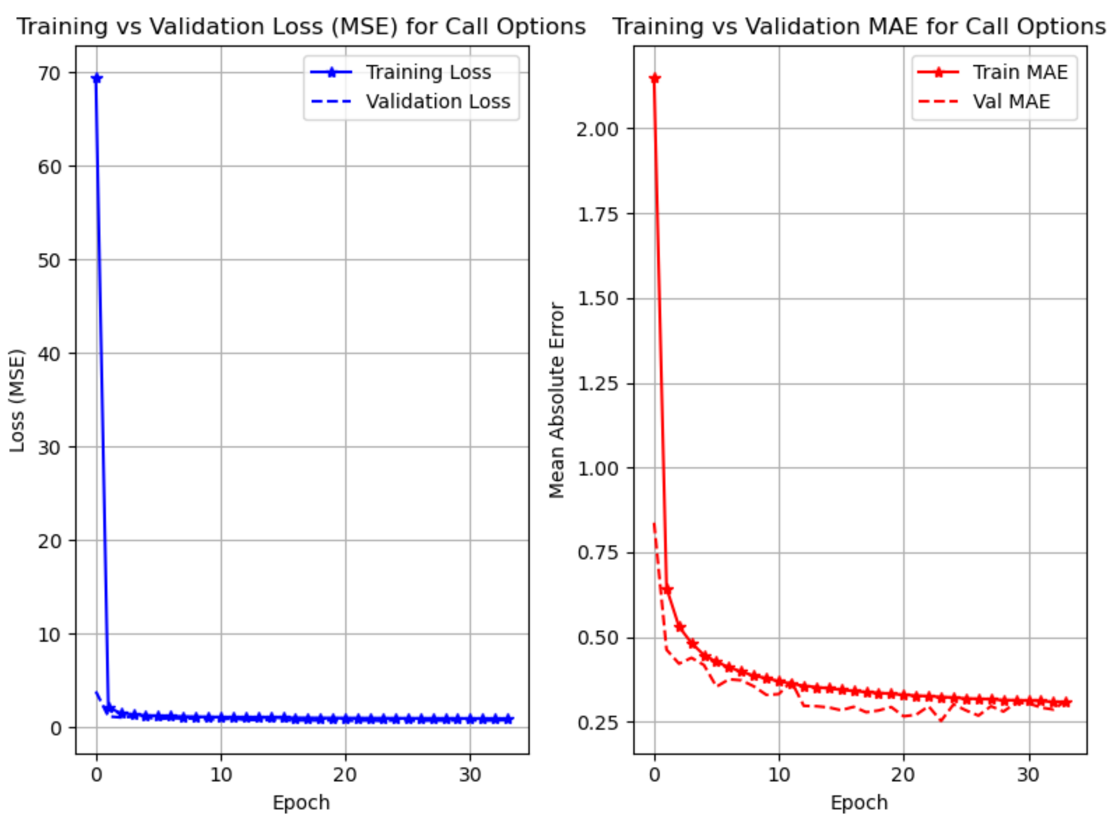
```python
end = time.time()
tuned_model_time_c = end - start

test_loss_best_c, test_mae_best_c = best_model_c.evaluate(X_test_scaled_c,␣
↪y_test_c)
print("\nTest MSE:", test_loss_best_c)
print("Test MAE:", test_mae_best_c)

best_model_c.save("best_model_calls.keras")
```

```
Epoch 1/50
7549/7549               19s 2ms/step -
loss: 297.5094 - mae: 4.8474 - val_loss: 3.8032 - val_mae: 0.8372
Epoch 2/50
7549/7549               17s 2ms/step -
loss: 2.5466 - mae: 0.7101 - val_loss: 1.1366 - val_mae: 0.4629
Epoch 3/50
7549/7549               18s 2ms/step -
loss: 1.5364 - mae: 0.5486 - val_loss: 1.0277 - val_mae: 0.4217
Epoch 4/50
7549/7549               17s 2ms/step -
loss: 1.4010 - mae: 0.4953 - val_loss: 1.1078 - val_mae: 0.4389
Epoch 5/50
7549/7549               17s 2ms/step -
loss: 1.2654 - mae: 0.4526 - val_loss: 1.0340 - val_mae: 0.4177
Epoch 6/50
7549/7549               17s 2ms/step -
loss: 1.2295 - mae: 0.4362 - val_loss: 0.8626 - val_mae: 0.3538
Epoch 7/50
7549/7549               18s 2ms/step -
loss: 1.1839 - mae: 0.4171 - val_loss: 0.9494 - val_mae: 0.3757
Epoch 8/50
7549/7549               17s 2ms/step -
loss: 1.1492 - mae: 0.4038 - val_loss: 0.9188 - val_mae: 0.3727
Epoch 9/50
7549/7549               20s 3ms/step -
loss: 1.1175 - mae: 0.3904 - val_loss: 0.9182 - val_mae: 0.3548
Epoch 10/50
7549/7549               18s 2ms/step -
loss: 1.1023 - mae: 0.3840 - val_loss: 0.8153 - val_mae: 0.3297
Epoch 11/50
7549/7549               18s 2ms/step -
loss: 1.0755 - mae: 0.3736 - val_loss: 0.8254 - val_mae: 0.3321
Epoch 12/50
7549/7549               18s 2ms/step -
loss: 1.0651 - mae: 0.3676 - val_loss: 0.9133 - val_mae: 0.3697
Epoch 13/50
```

```
7549/7549             18s 2ms/step -
loss: 1.0614 - mae: 0.3645 - val_loss: 0.7650 - val_mae: 0.2974
Epoch 14/50
7549/7549             17s 2ms/step -
loss: 1.0629 - mae: 0.3615 - val_loss: 0.7468 - val_mae: 0.2962
Epoch 15/50
7549/7549             17s 2ms/step -
loss: 1.0411 - mae: 0.3559 - val_loss: 0.7396 - val_mae: 0.2928
Epoch 16/50
7549/7549             18s 2ms/step -
loss: 1.0239 - mae: 0.3508 - val_loss: 0.7097 - val_mae: 0.2843
Epoch 17/50
7549/7549             17s 2ms/step -
loss: 1.0186 - mae: 0.3479 - val_loss: 0.7171 - val_mae: 0.2946
Epoch 18/50
7549/7549             17s 2ms/step -
loss: 1.0083 - mae: 0.3450 - val_loss: 0.7078 - val_mae: 0.2786
Epoch 19/50
7549/7549             17s 2ms/step -
loss: 0.9959 - mae: 0.3412 - val_loss: 0.7017 - val_mae: 0.2838
Epoch 20/50
7549/7549             16s 2ms/step -
loss: 1.0022 - mae: 0.3411 - val_loss: 0.7240 - val_mae: 0.2940
Epoch 21/50
7549/7549             17s 2ms/step -
loss: 0.9892 - mae: 0.3373 - val_loss: 0.6679 - val_mae: 0.2665
Epoch 22/50
7549/7549             17s 2ms/step -
loss: 0.9856 - mae: 0.3360 - val_loss: 0.6800 - val_mae: 0.2711
Epoch 23/50
7549/7549             17s 2ms/step -
loss: 0.9844 - mae: 0.3354 - val_loss: 0.7091 - val_mae: 0.2962
Epoch 24/50
7549/7549             17s 2ms/step -
loss: 0.9814 - mae: 0.3342 - val_loss: 0.6491 - val_mae: 0.2529
Epoch 25/50
7549/7549             16s 2ms/step -
loss: 0.9783 - mae: 0.3314 - val_loss: 0.7266 - val_mae: 0.3017
Epoch 26/50
7549/7549             17s 2ms/step -
loss: 0.9724 - mae: 0.3294 - val_loss: 0.6871 - val_mae: 0.2839
Epoch 27/50
7549/7549             17s 2ms/step -
loss: 0.9687 - mae: 0.3283 - val_loss: 0.6631 - val_mae: 0.2686
Epoch 28/50
7549/7549             17s 2ms/step -
loss: 0.9567 - mae: 0.3246 - val_loss: 0.7090 - val_mae: 0.2964
Epoch 29/50
```

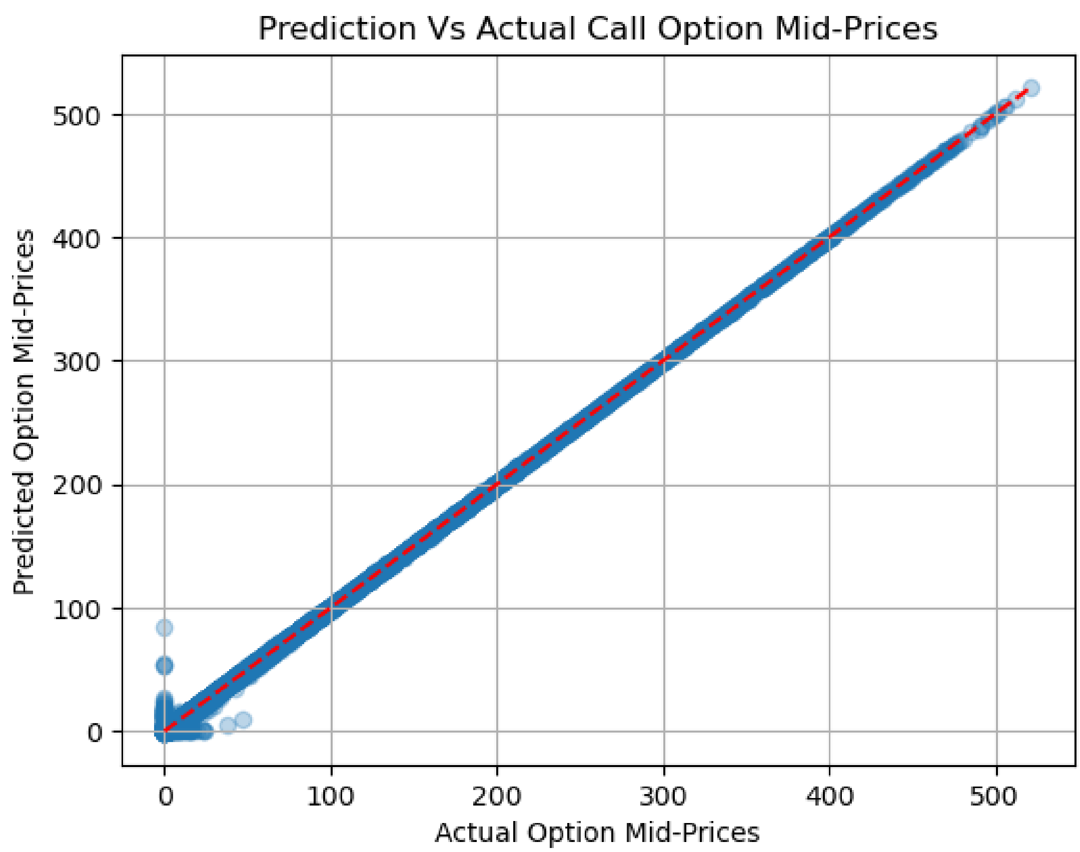
```
7549/7549              16s 2ms/step -
loss: 0.9532 - mae: 0.3231 - val_loss: 0.6870 - val_mae: 0.2806
Epoch 30/50
7549/7549              17s 2ms/step -
loss: 0.9469 - mae: 0.3225 - val_loss: 0.7291 - val_mae: 0.3064
Epoch 31/50
7549/7549              17s 2ms/step -
loss: 0.9513 - mae: 0.3227 - val_loss: 0.7262 - val_mae: 0.3052
Epoch 32/50
7549/7549              18s 2ms/step -
loss: 0.9395 - mae: 0.3198 - val_loss: 0.6934 - val_mae: 0.2924
Epoch 33/50
7549/7549              17s 2ms/step -
loss: 0.9350 - mae: 0.3172 - val_loss: 0.7055 - val_mae: 0.2865
Epoch 34/50
7549/7549              17s 2ms/step -
loss: 0.9326 - mae: 0.3160 - val_loss: 0.7484 - val_mae: 0.3125
1617/1617              3s 2ms/step -
loss: 0.7468 - mae: 0.2536

Test MSE: 0.844775915145874
Test MAE: 0.2551146447658539
```

[30]:
```
plot_learning_curves(history_best_c, "Call")
```



[31]:
```
get_prediction_plot(best_model_c, X_test_scaled_c,
                    y_test_c, "Call")
```

```
1617/1617                    2s 1ms/step
```



3.3.2 Put Option

[21]:
```
X_train_p_sub = X_train_scaled_p[::5]
X_valid_p_sub = X_valid_scaled_p[::5]

y_train_p_sub = y_train_p[::5]
y_valid_p_sub = y_valid_p[::5]

random_search_tuner = kt.RandomSearch(
    build_model,
    objective = "val_loss",
    seed = 42,
```

```
    max_trials = 30,
    overwrite = True
)

random_search_tuner.search(X_train_p_sub, y_train_p_sub, epochs = 30,
                        validation_data = (X_valid_p_sub, y_valid_p_sub), verbose =␣
 ↪1,
                              callbacks=[early_stop])
```

```
Trial 30 Complete [00h 02m 29s]
val_loss: 3.373318910598755

Best val_loss So Far: 2.8543083667755127
Total elapsed time: 00h 55m 25s
```

[22]:
```
best_hyperparams_p = random_search_tuner.get_best_hyperparameters(num_trials =␣
 ↪1)[0]
best_hyperparams_p.values
```

[22]:
```
{'n_hidden': 5,
 'n_neurons_1': 100,
 'activation': 'relu',
 'l2': 1.4513514342725857e-05,
 'learning_rate': 0.0009452767099282696,
 'optimizer': 'Nadam',
 'n_neurons_2': 72,
 'n_neurons_3': 45,
 'n_neurons_4': 21,
 'n_neurons_5': 57}
```

[32]:
```
best_model_p = build_model(best_hyperparams_p)

start = time.time()
history_best_p = best_model_p.fit(X_train_scaled_p, y_train_p,
             validation_data = (X_valid_scaled_p, y_valid_p),
             epochs = 50, verbose = 1, callbacks=[early_stop])
end = time.time()
tuned_model_time_p = end - start

test_loss_best_p, test_mae_best_p = best_model_p.evaluate(X_test_scaled_p,␣
 ↪y_test_p)
print("\nTest MSE:", test_loss_best_p)
print("Test MAE:", test_mae_best_p)

best_model_p.save("best_model_puts.keras")
```

```
Epoch 1/50
```

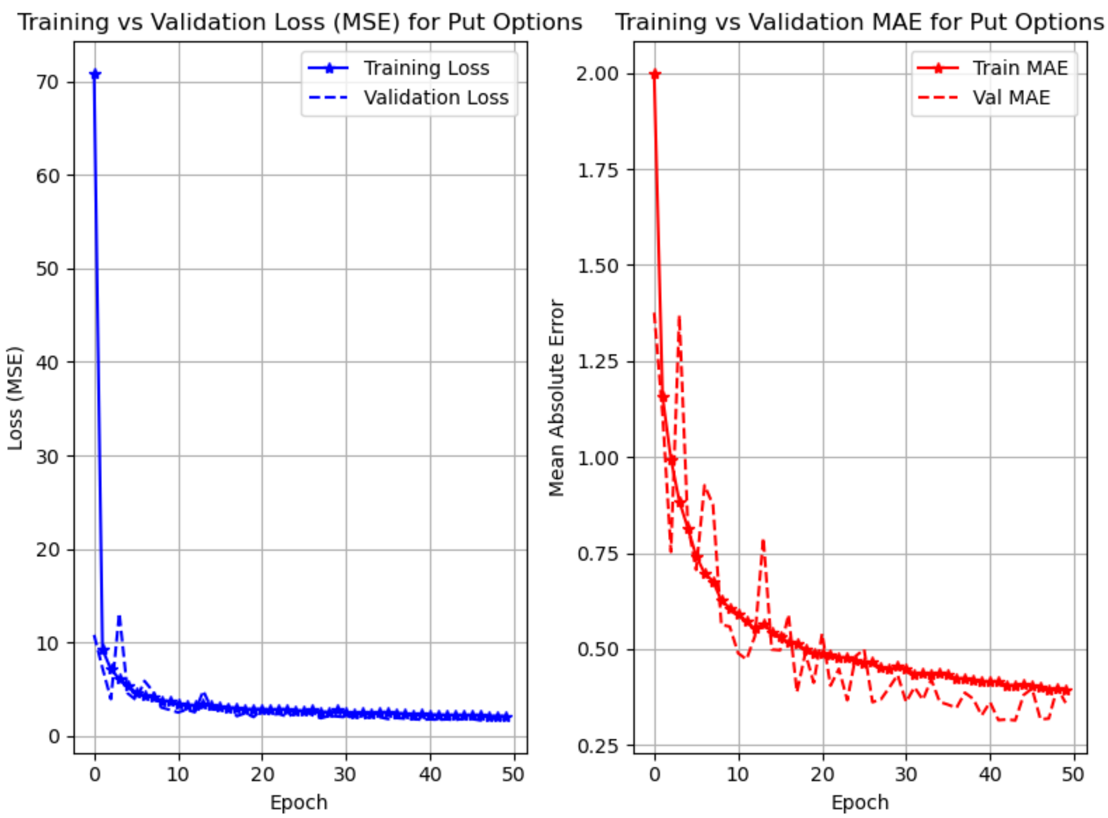
```
7550/7550             22s 2ms/step -
loss: 318.0915 - mae: 3.6989 - val_loss: 10.8060 - val_mae: 1.3758
Epoch 2/50
7550/7550             20s 3ms/step -
loss: 11.0536 - mae: 1.2549 - val_loss: 7.2598 - val_mae: 1.1116
Epoch 3/50
7550/7550             23s 3ms/step -
loss: 7.9689 - mae: 1.0529 - val_loss: 3.9439 - val_mae: 0.7532
Epoch 4/50
7550/7550             20s 3ms/step -
loss: 6.5212 - mae: 0.9298 - val_loss: 13.1403 - val_mae: 1.3690
Epoch 5/50
7550/7550             19s 3ms/step -
loss: 5.5278 - mae: 0.8274 - val_loss: 4.5645 - val_mae: 0.8259
Epoch 6/50
7550/7550             19s 3ms/step -
loss: 4.7953 - mae: 0.7639 - val_loss: 3.8391 - val_mae: 0.7064
Epoch 7/50
7550/7550             20s 3ms/step -
loss: 4.4247 - mae: 0.7106 - val_loss: 5.8779 - val_mae: 0.9263
Epoch 8/50
7550/7550             20s 3ms/step -
loss: 4.1753 - mae: 0.6839 - val_loss: 4.7803 - val_mae: 0.8759
Epoch 9/50
7550/7550             19s 3ms/step -
loss: 4.0149 - mae: 0.6506 - val_loss: 3.0456 - val_mae: 0.5631
Epoch 10/50
7550/7550             19s 2ms/step -
loss: 3.7992 - mae: 0.6221 - val_loss: 2.7964 - val_mae: 0.5587
Epoch 11/50
7550/7550             19s 2ms/step -
loss: 3.5853 - mae: 0.6012 - val_loss: 2.5173 - val_mae: 0.4892
Epoch 12/50
7550/7550             19s 2ms/step -
loss: 3.4402 - mae: 0.5796 - val_loss: 2.8135 - val_mae: 0.4731
Epoch 13/50
7550/7550             20s 3ms/step -
loss: 3.3973 - mae: 0.5604 - val_loss: 2.5415 - val_mae: 0.5338
Epoch 14/50
7550/7550             20s 3ms/step -
loss: 3.4829 - mae: 0.5665 - val_loss: 4.8309 - val_mae: 0.7896
Epoch 15/50
7550/7550             20s 3ms/step -
loss: 3.3990 - mae: 0.5645 - val_loss: 2.8756 - val_mae: 0.4989
Epoch 16/50
7550/7550             19s 2ms/step -
loss: 3.1936 - mae: 0.5405 - val_loss: 2.8150 - val_mae: 0.4970
Epoch 17/50
```

```
7550/7550            18s 2ms/step -
loss: 3.1361 - mae: 0.5273 - val_loss: 3.2202 - val_mae: 0.5897
Epoch 18/50
7550/7550            19s 2ms/step -
loss: 3.1547 - mae: 0.5292 - val_loss: 2.1182 - val_mae: 0.3878
Epoch 19/50
7550/7550            19s 3ms/step -
loss: 2.8985 - mae: 0.5019 - val_loss: 2.4412 - val_mae: 0.4894
Epoch 20/50
7550/7550            19s 3ms/step -
loss: 2.9947 - mae: 0.4952 - val_loss: 2.0268 - val_mae: 0.4126
Epoch 21/50
7550/7550            20s 3ms/step -
loss: 2.9918 - mae: 0.5001 - val_loss: 2.8676 - val_mae: 0.5401
Epoch 22/50
7550/7550            19s 3ms/step -
loss: 2.9596 - mae: 0.4990 - val_loss: 2.2578 - val_mae: 0.4049
Epoch 23/50
7550/7550            19s 3ms/step -
loss: 2.8667 - mae: 0.4750 - val_loss: 2.3602 - val_mae: 0.4485
Epoch 24/50
7550/7550            19s 2ms/step -
loss: 2.8646 - mae: 0.4764 - val_loss: 2.0667 - val_mae: 0.3668
Epoch 25/50
7550/7550            18s 2ms/step -
loss: 2.9535 - mae: 0.4819 - val_loss: 2.4650 - val_mae: 0.4817
Epoch 26/50
7550/7550            18s 2ms/step -
loss: 2.8617 - mae: 0.4824 - val_loss: 2.6603 - val_mae: 0.4992
Epoch 27/50
7550/7550            18s 2ms/step -
loss: 2.9167 - mae: 0.4800 - val_loss: 2.2304 - val_mae: 0.3618
Epoch 28/50
7550/7550            23s 3ms/step -
loss: 2.6285 - mae: 0.4590 - val_loss: 1.9010 - val_mae: 0.3689
Epoch 29/50
7550/7550            19s 2ms/step -
loss: 2.6563 - mae: 0.4557 - val_loss: 2.1501 - val_mae: 0.3998
Epoch 30/50
7550/7550            19s 3ms/step -
loss: 2.9138 - mae: 0.4717 - val_loss: 2.0838 - val_mae: 0.4328
Epoch 31/50
7550/7550            19s 2ms/step -
loss: 2.5626 - mae: 0.4484 - val_loss: 1.9963 - val_mae: 0.3620
Epoch 32/50
7550/7550            20s 3ms/step -
loss: 2.5251 - mae: 0.4394 - val_loss: 2.1065 - val_mae: 0.4008
Epoch 33/50
```

```
7550/7550            20s 3ms/step -
loss: 2.5577 - mae: 0.4456 - val_loss: 1.8781 - val_mae: 0.3679
Epoch 34/50
7550/7550            19s 3ms/step -
loss: 2.4690 - mae: 0.4381 - val_loss: 2.0685 - val_mae: 0.4209
Epoch 35/50
7550/7550            20s 3ms/step -
loss: 2.4860 - mae: 0.4380 - val_loss: 2.0144 - val_mae: 0.3624
Epoch 36/50
7550/7550            20s 3ms/step -
loss: 2.5600 - mae: 0.4327 - val_loss: 1.8152 - val_mae: 0.3553
Epoch 37/50
7550/7550            19s 2ms/step -
loss: 2.4994 - mae: 0.4256 - val_loss: 1.9254 - val_mae: 0.3468
Epoch 38/50
7550/7550            19s 3ms/step -
loss: 2.4538 - mae: 0.4296 - val_loss: 1.8603 - val_mae: 0.3864
Epoch 39/50
7550/7550            19s 2ms/step -
loss: 2.4073 - mae: 0.4283 - val_loss: 1.8291 - val_mae: 0.3703
Epoch 40/50
7550/7550            19s 2ms/step -
loss: 2.4066 - mae: 0.4166 - val_loss: 1.7332 - val_mae: 0.3274
Epoch 41/50
7550/7550            18s 2ms/step -
loss: 2.3233 - mae: 0.4189 - val_loss: 1.7921 - val_mae: 0.3617
Epoch 42/50
7550/7550            18s 2ms/step -
loss: 2.2780 - mae: 0.4164 - val_loss: 1.7321 - val_mae: 0.3151
Epoch 43/50
7550/7550            18s 2ms/step -
loss: 2.3089 - mae: 0.4157 - val_loss: 1.7851 - val_mae: 0.3167
Epoch 44/50
7550/7550            18s 2ms/step -
loss: 2.2808 - mae: 0.4117 - val_loss: 1.7097 - val_mae: 0.3144
Epoch 45/50
7550/7550            18s 2ms/step -
loss: 2.2843 - mae: 0.4067 - val_loss: 1.8567 - val_mae: 0.3805
Epoch 46/50
7550/7550            18s 2ms/step -
loss: 2.1762 - mae: 0.4015 - val_loss: 1.8308 - val_mae: 0.3973
Epoch 47/50
7550/7550            18s 2ms/step -
loss: 2.2376 - mae: 0.4047 - val_loss: 1.6673 - val_mae: 0.3171
Epoch 48/50
7550/7550            18s 2ms/step -
loss: 2.1754 - mae: 0.3979 - val_loss: 1.7649 - val_mae: 0.3192
Epoch 49/50
```

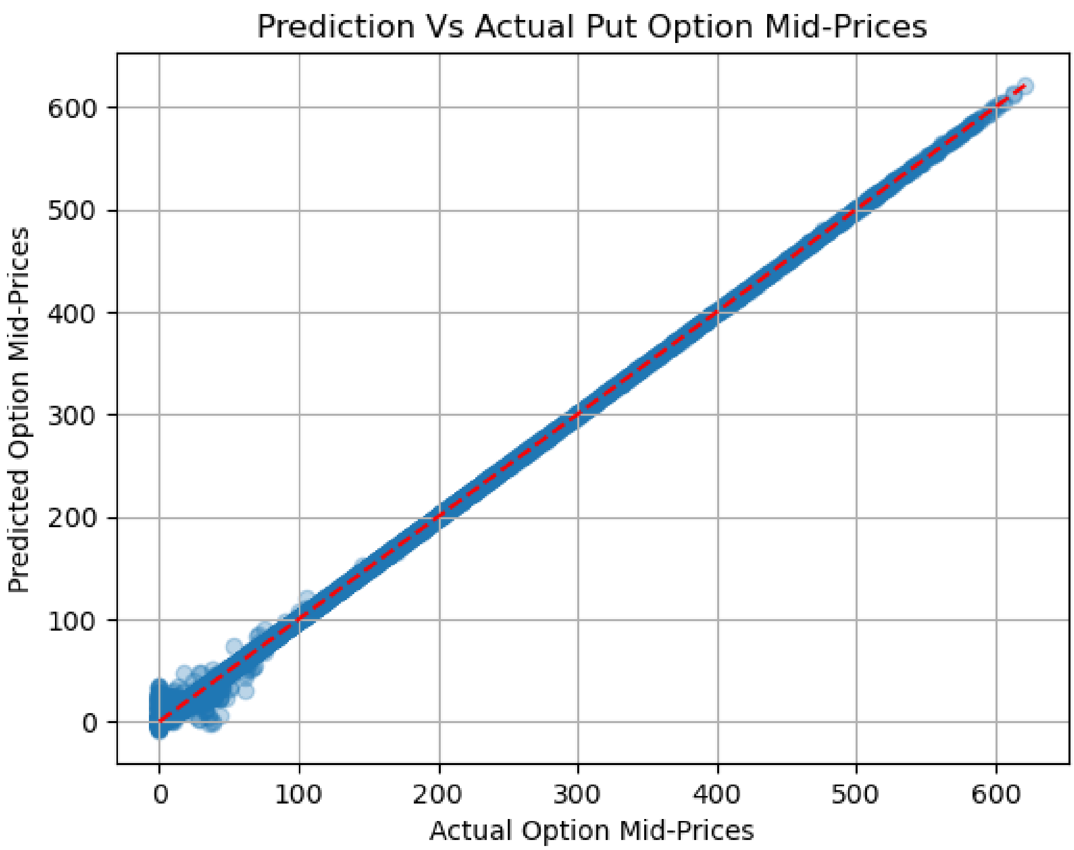
```
7550/7550              18s 2ms/step -
loss: 2.1808 - mae: 0.3970 - val_loss: 1.8743 - val_mae: 0.3997
Epoch 50/50
7550/7550              18s 2ms/step -
loss: 2.2774 - mae: 0.4038 - val_loss: 1.7478 - val_mae: 0.3598
1619/1619              3s 2ms/step -
loss: 1.6900 - mae: 0.3239

Test MSE: 1.654191255569458
Test MAE: 0.3205004334449768
```

[33]:
```
plot_learning_curves(history_best_p, "Put")
```



[34]:
```
get_prediction_plot(best_model_p, X_test_scaled_p,
                    y_test_p, "Put")
```

```
1619/1619              2s 1ms/step
```



[35]:
```
results_df = pd.DataFrame([
    {
        'Model': 'Baseline',
        'Option Type': 'Call Option',
        'Layers': 4,
        'Activation': "ReLU",
        'Optimizer': "Nadam",
        'Epochs': 50,
        'MAE': round(test_mae_c, 4),
        'MSE': round(test_loss_c, 4),
        'Train Time (s)': round(baseline_time_c, 2)
    },
    {
        'Model': 'Tuned',
        'Option Type': 'Call Option',
        'Layers': str(best_hyperparams_c['n_hidden']),
        'Activation': best_hyperparams_c['activation'],
        'Optimizer': best_hyperparams_c['optimizer'],
        'Epochs': max(history_best_c.epoch),
        'MAE': round(test_mae_best_c, 4),
```

```python
        'MSE': round(test_loss_best_c, 4),
        'Train Time (s)': round(tuned_model_time_c, 2)
    },
    {
        'Model': 'Baseline',
        'Option Type': 'Put Option',
        'Layers': 4,
        'Activation': "ReLU",
        'Optimizer': "Nadam",
        'Epochs': 50,
        'MAE': round(test_mae_p, 4),
        'MSE': round(test_loss_p, 4),
        'Train Time (s)': round(baseline_time_p, 2)
    },
    {
    'Model': 'Tuned',
    'Option Type': 'Put Option',
    'Layers': str(best_hyperparams_p['n_hidden']),
    'Activation': best_hyperparams_p['activation'],
    'Optimizer': best_hyperparams_p['optimizer'],
    'Epochs': max(history_best_p.epoch),
    'MAE': round(test_mae_best_p, 4),
    'MSE': round(test_loss_best_p, 4),
    'Train Time (s)': round(tuned_model_time_p, 2)
    }
])

display(results_df)
```

```
      Model  Option Type Layers Activation Optimizer  Epochs     MAE     MSE  \
0  Baseline  Call Option      4       ReLU     Nadam      50  0.2697  0.8442
1     Tuned  Call Option      2       relu     Nadam      33  0.2551  0.8448
2  Baseline   Put Option      4       ReLU     Nadam      50  0.3521  1.6476
3     Tuned   Put Option      5       relu     Nadam      49  0.3205  1.6542

   Train Time (s)
0          667.01
1          587.61
2          981.35
3          955.78
```