newfile

June 22, 2025

[ ]:
```
import os
import glob
import pandas as pd

base_dir = 'advanceaimethodsallfilestobeextracted22222222'
# check
if not os.path.isdir(base_dir):
    raise FileNotFoundError(f"Base directory not found: {base_dir}")

topic_paths = glob.glob(os.path.join(base_dir, '**', 'topics_combined.csv'),␣
 ↪recursive=True)
print(f"Discovered {len(topic_paths)} topics_combined.csv files")

dfs = []
for csv_path in topic_paths:
    df = pd.read_csv(csv_path)
    df['subset']     = os.path.basename(os.path.dirname(csv_path))
    df['source_csv'] = csv_path
    dfs.append(df)

# Concatenate into one DataFrame
combined = pd.concat(dfs, ignore_index=True)
print(f"Total rows in combined DataFrame: {len(combined)}")

# peek
display(combined.head())
display(combined[['subset','filename','timestamp_x']].describe())
```

```
Discovered 8 topics_combined.csv files
Total rows in combined DataFrame: 2644
```

```
    timestamp_x       cog       sog   timestamp_y  waterdepth     timestamp  \
0  1.561660e+18  1.102289  1.064741  1.561660e+18    4.719632  1.561660e+18
1  1.561660e+18  1.107757  1.070156  1.561660e+18    4.719303  1.561660e+18
2  1.561660e+18  1.113364  1.075707  1.561660e+18    4.718966  1.561660e+18
3  1.561660e+18  1.118834  1.081123  1.561660e+18    4.718637  1.561660e+18
4  1.561660e+18  1.124384  1.086618  1.561660e+18    4.718303  1.561660e+18
```

```
   height  width                                           filename  \
0   500.0  530.0  _camera_crop_image_rect_color_compressed_00013…
1   500.0  530.0  _camera_crop_image_rect_color_compressed_00013…
2   500.0  530.0  _camera_crop_image_rect_color_compressed_00013…
3   500.0  530.0  _camera_crop_image_rect_color_compressed_00013…
4   500.0  530.0  _camera_crop_image_rect_color_compressed_00013…

                                          filepath  \
0  camera_images/_camera_crop_image_rect_color_co…
1  camera_images/_camera_crop_image_rect_color_co…
2  camera_images/_camera_crop_image_rect_color_co…
3  camera_images/_camera_crop_image_rect_color_co…
4  camera_images/_camera_crop_image_rect_color_co…

                              subset  \
0  subset_2019-06-27-21-20-46_extract
1  subset_2019-06-27-21-20-46_extract
2  subset_2019-06-27-21-20-46_extract
3  subset_2019-06-27-21-20-46_extract
4  subset_2019-06-27-21-20-46_extract

                                        source_csv
0  advanceaimethodsallfilestobeextracted22222222/…
1  advanceaimethodsallfilestobeextracted22222222/…
2  advanceaimethodsallfilestobeextracted22222222/…
3  advanceaimethodsallfilestobeextracted22222222/…
4  advanceaimethodsallfilestobeextracted22222222/…

        timestamp_x
count  2.644000e+03
mean   1.561657e+18
std    1.769171e+12
min    1.561655e+18
25%    1.561655e+18
50%    1.561657e+18
75%    1.561659e+18
max    1.561660e+18
```

[ ]:
```
import glob
import pandas as pd
import os

# Load speed data
speed_paths = glob.glob(os.path.join(base_dir, '**', '_speed.csv'),␣
 ↪recursive=True)
speed_dfs = []
for sp in speed_paths:
    df_sp = pd.read_csv(sp, parse_dates=['datetime'])
```

```python
    df_sp['subset'] = os.path.basename(os.path.dirname(sp))
    speed_dfs.append(df_sp)
speeds = pd.concat(speed_dfs, ignore_index=True)

# timestamp to int64
combined['ts_int'] = combined['timestamp_x'].astype(float).astype('int64')
speeds  ['ts_int'] = speeds  ['timestamp'].astype(float).astype('int64')

# 4) Sort and as-of-merge
combined = combined.sort_values('ts_int')
speeds   = speeds.sort_values('ts_int')
merged = pd.merge_asof(
    combined,
    speeds[['ts_int','sog','cog','datetime']],
    on='ts_int',
    direction='nearest',
    tolerance=1_000_000_000
)
# clean data
print("Columns in merged DataFrame:")
print(merged.columns.tolist(), "\n")


y_cols = [c for c in merged.columns if c.endswith('_y')]
na_count = merged[y_cols].isna().any(axis=1).sum()
print(f"Rows with missing speed data (in {y_cols}): {na_count}\n")

# Rename columns to avoid conflicts
rename_map = {col: col.rstrip('_y') + '_speed' for col in y_cols}
merged = merged.rename(columns=rename_map)

print("Renamed columns:")
print(merged.columns.tolist())
```

```
Columns in merged DataFrame:
['timestamp_x', 'cog_x', 'sog_x', 'timestamp_y', 'waterdepth', 'timestamp',
'height', 'width', 'filename', 'filepath', 'subset', 'source_csv', 'ts_int',
'sog_y', 'cog_y', 'datetime']

Rows with missing speed data (in ['timestamp_y', 'sog_y', 'cog_y']): 0

Renamed columns:
['timestamp_x', 'cog_x', 'sog_x', 'timestamp_speed', 'waterdepth', 'timestamp',
'height', 'width', 'filename', 'filepath', 'subset', 'source_csv', 'ts_int',
'sog_speed', 'cog_speed', 'datetime']
```

[ ]:
```python
import numpy as np

# Sort timestamp
merged = merged.sort_values('ts_int').reset_index(drop=True)
# split
n = len(merged)
train_end = int(0.6 * n)
val_end   = int(0.8 * n)

train_df = merged.iloc[:train_end].copy()
val_df   = merged.iloc[train_end:val_end].copy()
test_df  = merged.iloc[val_end:].copy()

print(f"Total samples: {n}")
print(f"  Train:      {len(train_df)} ({len(train_df)/n:.1%})")
print(f"  Validation: {len(val_df)} ({len(val_df)/n:.1%})")
print(f"  Test:       {len(test_df)} ({len(test_df)/n:.1%})")

# saveing splits
train_df.to_csv('train_split.csv', index=False)
val_df.to_csv('val_split.csv',   index=False)
test_df.to_csv('test_split.csv',  index=False)
```

```
Total samples: 2644
  Train:      1586 (60.0%)
  Validation: 529 (20.0%)
  Test:       529 (20.0%)
```

[ ]:
```python
import os
import pandas as pd
from torch.utils.data import Dataset
from PIL import Image
import torch

class SonarSequenceDataset(Dataset):
    """
    Returns a sequence of T sonar images plus
    the corresponding speed, cog, and depth values.
    """
    def __init__(self, csv_path, base_dir, seq_len=10, transform=None):
        self.df = pd.read_csv(csv_path)
        self.base_dir = base_dir
        self.seq_len = seq_len
        self.transform = transform

        self.starts = list(range(len(self.df) - seq_len + 1))
```

```
    def __len__(self):
        return len(self.starts)

    def __getitem__(self, idx):
        start = self.starts[idx]
        end   = start + self.seq_len
        records = self.df.iloc[start:end]

        imgs = []
        for _, row in records.iterrows():
            img_path = os.path.join(self.base_dir, row['subset'],␣
↪row['filepath']) #include subsett
            img = Image.open(img_path).convert('RGB')
            if self.transform:
                img = self.transform(img)
            imgs.append(img)
        imgs = torch.stack(imgs, dim=0)

        last = records.iloc[-1]
        return {
            'images': imgs,
            'speed':  torch.tensor(last['sog_speed'], dtype=torch.float32),
            'cog':    torch.tensor(last['cog_speed'], dtype=torch.float32),
            'depth':  torch.tensor(last['waterdepth'], dtype=torch.float32)
        }
```

[5]:
```
from torchvision import transforms
base_dir = 'advanceaimethodsallfilestobeextracted22222222'

transform = transforms.Compose([
    transforms.Resize((224,224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5,0.5,0.5], std=[0.5,0.5,0.5])
])

train_ds = SonarSequenceDataset('train_split.csv', base_dir, seq_len=10,␣
↪transform=transform)
val_ds   = SonarSequenceDataset('val_split.csv',   base_dir, seq_len=10,␣
↪transform=transform)
test_ds  = SonarSequenceDataset('test_split.csv',  base_dir, seq_len=10,␣
↪transform=transform)

print(f"Train sequences: {len(train_ds)}")
sample = train_ds[0]
print("Sample image sequence shape:", sample['images'].shape)
print("Sample speed, cog, depth:", sample['speed'], sample['cog'],␣
↪sample['depth'])
```

```
Train sequences: 1577
Sample image sequence shape: torch.Size([10, 3, 224, 224])
Sample speed, cog, depth: tensor(0.1600) tensor(3.8227) tensor(0.5500)
```

[ ]:
```python
import torch
import torch.nn as nn
import torchvision.models as models

class SonarCNNLSTM(nn.Module):
    def __init__(
        self,
        cnn_backbone='resnet18',
        cnn_out_dim=128,
        lstm_hidden=64,
        lstm_layers=1,
        dropout=0.2,
        bidirectional=False,
        num_targets=3
    ):
        super().__init__()
        # feature extractor
        backbone = getattr(models, cnn_backbone)(pretrained=False)
        # replace final FC with our projection
        in_features = backbone.fc.in_features
        backbone.fc = nn.Linear(in_features, cnn_out_dim)
        self.cnn = backbone

        # LSTM over the CNN features
        self.lstm = nn.LSTM(
            input_size=cnn_out_dim,
            hidden_size=lstm_hidden,
            num_layers=lstm_layers,
            batch_first=True,
            dropout=dropout if lstm_layers > 1 else 0.0,
            bidirectional=bidirectional
        )

        # Final head
        lstm_output_dim = lstm_hidden * (2 if bidirectional else 1)
        self.regressor = nn.Sequential(
            nn.Dropout(dropout),
            nn.Linear(lstm_output_dim, lstm_output_dim//2),
            nn.ReLU(inplace=True),
            nn.Linear(lstm_output_dim//2, num_targets)
        )

    def forward(self, x):
```

```
        """
        x: tensor of shape (B, T, C, H, W)
        returns: tensor of shape (B, num_targets)
        """
        B, T, C, H, W = x.shape
        # collapse time and batch to run through CNN
        x = x.view(B * T, C, H, W)
        feats = self.cnn(x)
        feats = feats.view(B, T, -1)

        # run LSTM
        lstm_out, _ = self.lstm(feats)      # (B, T, lstm_hidden *␣
↪num_directions)
        last_step = lstm_out[:, -1, :]      # (B, lstm_hidden * num_directions)

        # final regression
        out = self.regressor(last_step)     # (B, num_targets)
        return out
```

[ ]:
```
if __name__ == '__main__':
    model = SonarCNNLSTM(
        cnn_backbone='resnet18',
        cnn_out_dim=128,
        lstm_hidden=64,
        lstm_layers=1,
        dropout=0.2,
        bidirectional=False,
        num_targets=3
    )
# dumy data
    dummy = torch.randn(2, 10, 3, 224, 224)
    output = model(dummy)
    print("Output shape:", output.shape)  # expect (2,3)
```

```
Output shape: torch.Size([2, 3])
```

[ ]:
```
from torch.utils.data import DataLoader

# Hyperparameters
batch_size = 8
seq_len    = 10

# reuse from before
transform = transforms.Compose([
    transforms.Resize((224,224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5]*3, std=[0.5]*3)
```

```
]) #dataset and transforms
train_ds = SonarSequenceDataset('train_split.csv', base_dir, seq_len, transform)
val_ds   = SonarSequenceDataset('val_split.csv',   base_dir, seq_len, transform)

# DataLoaders
train_loader = DataLoader(train_ds, batch_size=batch_size, shuffle=True, ␣
 ↪num_workers=4)
val_loader   = DataLoader(val_ds,   batch_size=batch_size, shuffle=False,␣
 ↪num_workers=4)
```

[ ]:
```
import torch.optim as optim
import torch.nn as nn

# Instantiate model, loss, optimizer
model     = SonarCNNLSTM()
criterion = nn.MSELoss()  # since speed, cog, depth are continuous
optimizer = optim.Adam(model.parameters(), lr=1e-3)

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")
model.to(device)

# Training & Validation
num_epochs = 100
best_val_loss = float('inf')
train_losses, val_losses = [], []

for epoch in range(1, num_epochs+1):
    # - Training -
    model.train()
    running_train_loss = 0.0
    for batch in train_loader:
        imgs   = batch['images'].to(device)     # (B, T, C, H, W)
        targets = torch.stack([batch['speed'],
                               batch['cog'],
                               batch['depth']], dim=1).to(device)  # (B,3)

        preds = model(imgs)                     # (B,3)
        loss  = criterion(preds, targets)

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

        running_train_loss += loss.item() * imgs.size(0)

    epoch_train_loss = running_train_loss / len(train_ds)
```

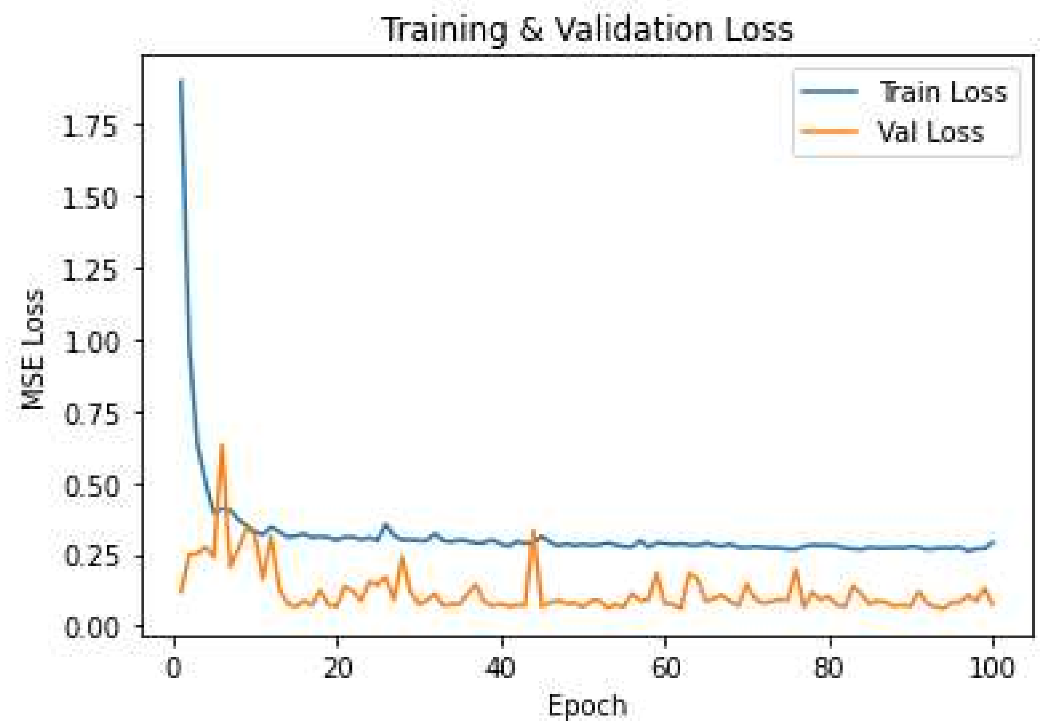
```python
    train_losses.append(epoch_train_loss)

    # - Validation -
    model.eval()
    running_val_loss = 0.0
    with torch.no_grad():
        for batch in val_loader:
            imgs    = batch['images'].to(device)
            targets = torch.stack([batch['speed'],
                                   batch['cog'],
                                   batch['depth']], dim=1).to(device)
            preds   = model(imgs)
            loss    = criterion(preds, targets)
            running_val_loss += loss.item() * imgs.size(0)

    epoch_val_loss = running_val_loss / len(val_ds)
    val_losses.append(epoch_val_loss)

    print(f"Epoch {epoch}/{num_epochs} - "
          f"Train Loss: {epoch_train_loss:.4f}  "
          f"Val Loss: {epoch_val_loss:.4f}")

    # Save best model
    if epoch_val_loss < best_val_loss:
        best_val_loss = epoch_val_loss
        torch.save(model.state_dict(), 'best_model001newfile.pth')
```

```
Using device: cuda
Epoch 1/100 - Train Loss: 1.9004  Val Loss: 0.1175
Epoch 2/100 - Train Loss: 0.9845  Val Loss: 0.2491
Epoch 3/100 - Train Loss: 0.6328  Val Loss: 0.2520
Epoch 4/100 - Train Loss: 0.5000  Val Loss: 0.2760
Epoch 5/100 - Train Loss: 0.3951  Val Loss: 0.2401
Epoch 6/100 - Train Loss: 0.4086  Val Loss: 0.6317
Epoch 7/100 - Train Loss: 0.4040  Val Loss: 0.2050
Epoch 8/100 - Train Loss: 0.3715  Val Loss: 0.2693
Epoch 9/100 - Train Loss: 0.3495  Val Loss: 0.3424
Epoch 10/100 - Train Loss: 0.3300  Val Loss: 0.3190
Epoch 11/100 - Train Loss: 0.3180  Val Loss: 0.1648
Epoch 12/100 - Train Loss: 0.3451  Val Loss: 0.3128
Epoch 13/100 - Train Loss: 0.3283  Val Loss: 0.1288
Epoch 14/100 - Train Loss: 0.3095  Val Loss: 0.0781
Epoch 15/100 - Train Loss: 0.3134  Val Loss: 0.0667
Epoch 16/100 - Train Loss: 0.3214  Val Loss: 0.0867
Epoch 17/100 - Train Loss: 0.3086  Val Loss: 0.0714
Epoch 18/100 - Train Loss: 0.3107  Val Loss: 0.1244
Epoch 19/100 - Train Loss: 0.3093  Val Loss: 0.0750
```

```
Epoch 20/100 - Train Loss: 0.2982  Val Loss: 0.0671
Epoch 21/100 - Train Loss: 0.3107  Val Loss: 0.1379
Epoch 22/100 - Train Loss: 0.3098  Val Loss: 0.1215
Epoch 23/100 - Train Loss: 0.3007  Val Loss: 0.0882
Epoch 24/100 - Train Loss: 0.3079  Val Loss: 0.1541
Epoch 25/100 - Train Loss: 0.2986  Val Loss: 0.1438
Epoch 26/100 - Train Loss: 0.3554  Val Loss: 0.1711
Epoch 27/100 - Train Loss: 0.3147  Val Loss: 0.0895
Epoch 28/100 - Train Loss: 0.2998  Val Loss: 0.2414
Epoch 29/100 - Train Loss: 0.3005  Val Loss: 0.1191
Epoch 30/100 - Train Loss: 0.2978  Val Loss: 0.0761
Epoch 31/100 - Train Loss: 0.2980  Val Loss: 0.0869
Epoch 32/100 - Train Loss: 0.3222  Val Loss: 0.1129
Epoch 33/100 - Train Loss: 0.2970  Val Loss: 0.0702
Epoch 34/100 - Train Loss: 0.2938  Val Loss: 0.0768
Epoch 35/100 - Train Loss: 0.3000  Val Loss: 0.0751
Epoch 36/100 - Train Loss: 0.2958  Val Loss: 0.1115
Epoch 37/100 - Train Loss: 0.2890  Val Loss: 0.1437
Epoch 38/100 - Train Loss: 0.2905  Val Loss: 0.0849
Epoch 39/100 - Train Loss: 0.3003  Val Loss: 0.0704
Epoch 40/100 - Train Loss: 0.2867  Val Loss: 0.0773
Epoch 41/100 - Train Loss: 0.2778  Val Loss: 0.0675
Epoch 42/100 - Train Loss: 0.2944  Val Loss: 0.0736
Epoch 43/100 - Train Loss: 0.2882  Val Loss: 0.0720
Epoch 44/100 - Train Loss: 0.2890  Val Loss: 0.3353
Epoch 45/100 - Train Loss: 0.3130  Val Loss: 0.0681
Epoch 46/100 - Train Loss: 0.2928  Val Loss: 0.0822
Epoch 47/100 - Train Loss: 0.2796  Val Loss: 0.0867
Epoch 48/100 - Train Loss: 0.2860  Val Loss: 0.0789
Epoch 49/100 - Train Loss: 0.2809  Val Loss: 0.0798
Epoch 50/100 - Train Loss: 0.2836  Val Loss: 0.0661
Epoch 51/100 - Train Loss: 0.2811  Val Loss: 0.0871
Epoch 52/100 - Train Loss: 0.2826  Val Loss: 0.0899
Epoch 53/100 - Train Loss: 0.2901  Val Loss: 0.0627
Epoch 54/100 - Train Loss: 0.2828  Val Loss: 0.0755
Epoch 55/100 - Train Loss: 0.2775  Val Loss: 0.0660
Epoch 56/100 - Train Loss: 0.2757  Val Loss: 0.1138
Epoch 57/100 - Train Loss: 0.2981  Val Loss: 0.0885
Epoch 58/100 - Train Loss: 0.2748  Val Loss: 0.0902
Epoch 59/100 - Train Loss: 0.2888  Val Loss: 0.1857
Epoch 60/100 - Train Loss: 0.2882  Val Loss: 0.0786
Epoch 61/100 - Train Loss: 0.2843  Val Loss: 0.0761
Epoch 62/100 - Train Loss: 0.2869  Val Loss: 0.0602
Epoch 63/100 - Train Loss: 0.2823  Val Loss: 0.1822
Epoch 64/100 - Train Loss: 0.2802  Val Loss: 0.1651
Epoch 65/100 - Train Loss: 0.2896  Val Loss: 0.0857
Epoch 66/100 - Train Loss: 0.2817  Val Loss: 0.0943
Epoch 67/100 - Train Loss: 0.2780  Val Loss: 0.1074
```

```
Epoch 68/100 - Train Loss: 0.2857  Val Loss: 0.0847
Epoch 69/100 - Train Loss: 0.2740  Val Loss: 0.0699
Epoch 70/100 - Train Loss: 0.2733  Val Loss: 0.1448
Epoch 71/100 - Train Loss: 0.2760  Val Loss: 0.0963
Epoch 72/100 - Train Loss: 0.2742  Val Loss: 0.0794
Epoch 73/100 - Train Loss: 0.2716  Val Loss: 0.0839
Epoch 74/100 - Train Loss: 0.2725  Val Loss: 0.0925
Epoch 75/100 - Train Loss: 0.2680  Val Loss: 0.0890
Epoch 76/100 - Train Loss: 0.2673  Val Loss: 0.1958
Epoch 77/100 - Train Loss: 0.2770  Val Loss: 0.0638
Epoch 78/100 - Train Loss: 0.2822  Val Loss: 0.1173
Epoch 79/100 - Train Loss: 0.2809  Val Loss: 0.0919
Epoch 80/100 - Train Loss: 0.2816  Val Loss: 0.1032
Epoch 81/100 - Train Loss: 0.2782  Val Loss: 0.0738
Epoch 82/100 - Train Loss: 0.2714  Val Loss: 0.0664
Epoch 83/100 - Train Loss: 0.2695  Val Loss: 0.1390
Epoch 84/100 - Train Loss: 0.2667  Val Loss: 0.1116
Epoch 85/100 - Train Loss: 0.2745  Val Loss: 0.0758
Epoch 86/100 - Train Loss: 0.2715  Val Loss: 0.0868
Epoch 87/100 - Train Loss: 0.2708  Val Loss: 0.0828
Epoch 88/100 - Train Loss: 0.2737  Val Loss: 0.0682
Epoch 89/100 - Train Loss: 0.2708  Val Loss: 0.0735
Epoch 90/100 - Train Loss: 0.2770  Val Loss: 0.0671
Epoch 91/100 - Train Loss: 0.2740  Val Loss: 0.1201
Epoch 92/100 - Train Loss: 0.2673  Val Loss: 0.0792
Epoch 93/100 - Train Loss: 0.2697  Val Loss: 0.0673
Epoch 94/100 - Train Loss: 0.2724  Val Loss: 0.0608
Epoch 95/100 - Train Loss: 0.2707  Val Loss: 0.0813
Epoch 96/100 - Train Loss: 0.2750  Val Loss: 0.0825
Epoch 97/100 - Train Loss: 0.2615  Val Loss: 0.1074
Epoch 98/100 - Train Loss: 0.2692  Val Loss: 0.0856
Epoch 99/100 - Train Loss: 0.2697  Val Loss: 0.1317
Epoch 100/100 - Train Loss: 0.2911  Val Loss: 0.0754
```

[10]:
```python
import matplotlib.pyplot as plt

epochs = range(1, num_epochs+1)
plt.plot(epochs, train_losses, label='Train Loss')
plt.plot(epochs, val_losses,   label='Val Loss')
plt.xlabel('Epoch')
plt.ylabel('MSE Loss')
plt.legend()
plt.title('Training & Validation Loss')
plt.show()
```



[ ]:
```
import torch
import torch.nn as nn
from torch.utils.data import DataLoader
from torchvision import transforms
from PIL import Image
import os
import pandas as pd

class SonarSequenceDataset(torch.utils.data.Dataset):
    def __init__(self, csv_path, base_dir, seq_len=10, transform=None):
        self.df = pd.read_csv(csv_path)
        self.base_dir = base_dir
        self.seq_len = seq_len
        self.transform = transform
        self.starts = list(range(len(self.df) - seq_len + 1))

    def __len__(self):
        return len(self.starts)

    def __getitem__(self, idx):
        start = self.starts[idx]
        records = self.df.iloc[start:start+self.seq_len]
        imgs = []
```

```python
        for _, row in records.iterrows():
            path = os.path.join(self.base_dir, row['subset'], row['filepath'])
            img = Image.open(path).convert('RGB')
            if self.transform:
                img = self.transform(img)
            imgs.append(img)
        imgs = torch.stack(imgs, dim=0)
        last = records.iloc[-1]
        targets = torch.tensor([last['sog_speed'], last['cog_speed'],␣
↪last['waterdepth']], dtype=torch.float32)
        return {'images': imgs, 'targets': targets}

class SonarCNNLSTM(nn.Module):
    def __init__(self, cnn_out_dim=128, lstm_hidden=64, lstm_layers=1,␣
↪dropout=0.2, bidirectional=False, num_targets=3):
        super().__init__()
        from torchvision.models import resnet18
        backbone = resnet18(pretrained=False)
        in_features = backbone.fc.in_features
        backbone.fc = nn.Linear(in_features, cnn_out_dim)
        self.cnn = backbone
        self.lstm = nn.LSTM(
            input_size=cnn_out_dim,
            hidden_size=lstm_hidden,
            num_layers=lstm_layers,
            batch_first=True,
            dropout=dropout if lstm_layers > 1 else 0.0,
            bidirectional=bidirectional
        )
        lstm_dim = lstm_hidden * (2 if bidirectional else 1)
        self.regressor = nn.Sequential(
            nn.Dropout(dropout),
            nn.Linear(lstm_dim, lstm_dim // 2),
            nn.ReLU(inplace=True),
            nn.Linear(lstm_dim // 2, num_targets)
        )

    def forward(self, x):
        B, T, C, H, W = x.shape
        x = x.view(B * T, C, H, W)
        feats = self.cnn(x).view(B, T, -1)
        lstm_out, _ = self.lstm(feats)
        return self.regressor(lstm_out[:, -1, :])

# Load Test Set
transform = transforms.Compose([
    transforms.Resize((224, 224)),
```

```
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5]*3, std=[0.5]*3)
])
base_dir = '/home/user/persistent/advanceaimethodsallfilestobeextracted22222222'
test_ds = SonarSequenceDataset('test_split.csv', base_dir, seq_len=10,␣
 ↪transform=transform)
test_loader = DataLoader(test_ds, batch_size=8, shuffle=False)

#   Evaluate
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = SonarCNNLSTM()
model.load_state_dict(torch.load('best_model001newfile.pth',␣
 ↪map_location=device))
model.to(device).eval()

criterion = nn.MSELoss(reduction='none')
all_losses, all_targets, all_preds = [], [], []

with torch.no_grad():
    for batch in test_loader:
        imgs = batch['images'].to(device)
        targets = batch['targets'].to(device)
        preds = model(imgs)
        loss = criterion(preds, targets)
        all_losses.append(loss.cpu())
        all_targets.append(targets.cpu())
        all_preds.append(preds.cpu())

losses = torch.cat(all_losses, dim=0).numpy()
targets = torch.cat(all_targets, dim=0).numpy()
preds = torch.cat(all_preds, dim=0).numpy()

mse_per_target = losses.mean(axis=0)
overall_mse = mse_per_target.mean()

df_metrics = pd.DataFrame({
    'Target': ['Speed', 'COG', 'Depth', 'Overall'],
    'MSE': [mse_per_target[0], mse_per_target[1], mse_per_target[2],␣
 ↪overall_mse]
})

print("\nTest Evaluation Results (Mean Squared Error per target):")
print(df_metrics)
```

```
Test Evaluation Results (Mean Squared Error per target):
    Target        MSE
```

```
0    Speed   0.040594
1      COG  12.258199
2    Depth   4.138172
3  Overall   5.478988
```

[12]:
```
def train_model(run_id, model, train_loader, val_loader, criterion, optimizer,␣
↪num_epochs=100, device='cpu'):
    train_losses, val_losses = [], []
    best_model_wts = None
    best_val_loss = float('inf')

    for epoch in range(num_epochs):
        # Training
        model.train()
        train_loss = 0.0
        for batch in train_loader:
            imgs = batch['images'].to(device)
            targets = torch.stack([batch['speed'], batch['cog'],␣
↪batch['depth']], dim=1).to(device)

            preds = model(imgs)
            loss = criterion(preds, targets)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            train_loss += loss.item() * imgs.size(0)

        epoch_train_loss = train_loss / len(train_loader.dataset)
        train_losses.append(epoch_train_loss)

        # Validation
        model.eval()
        val_loss = 0.0
        with torch.no_grad():
            for batch in val_loader:
                imgs = batch['images'].to(device)
                targets = torch.stack([batch['speed'], batch['cog'],␣
↪batch['depth']], dim=1).to(device)

                preds = model(imgs)
                loss = criterion(preds, targets)
                val_loss += loss.item() * imgs.size(0)

        epoch_val_loss = val_loss / len(val_loader.dataset)
        val_losses.append(epoch_val_loss)
```

```
        print(f"Run {run_id} | Epoch {epoch+1}/{num_epochs} - Train Loss:␣
↪{epoch_train_loss:.4f} | Val Loss: {epoch_val_loss:.4f}")

        # Save best
        if epoch_val_loss < best_val_loss:
            best_val_loss = epoch_val_loss
            best_model_wts = model.state_dict()

    # Save best weights
    torch.save(best_model_wts, f'model_run{run_id}.pt')
    return train_losses, val_losses
```

[13]:
```
import torch.nn as nn
import torch.optim as optim

all_losses = []

for run in range(1, 4):
    print(f"\n=== Training Run {run}/3 ===\n")
    model = SonarCNNLSTM().to(device)
    criterion = nn.MSELoss()
    optimizer = optim.Adam(model.parameters(), lr=1e-4)

    train_losses, val_losses = train_model(
        run_id=run,
        model=model,
        train_loader=train_loader,
        val_loader=val_loader,
        criterion=criterion,
        optimizer=optimizer,
        num_epochs=100,
        device=device
    )
    all_losses.append((train_losses, val_losses))
```

```
=== Training Run 1/3 ===

Run 1 | Epoch 1/100 - Train Loss: 4.5290 | Val Loss: 0.7208
Run 1 | Epoch 2/100 - Train Loss: 1.1570 | Val Loss: 0.2657
Run 1 | Epoch 3/100 - Train Loss: 0.5527 | Val Loss: 0.1624
Run 1 | Epoch 4/100 - Train Loss: 0.4156 | Val Loss: 0.1445
Run 1 | Epoch 5/100 - Train Loss: 0.3340 | Val Loss: 0.1006
Run 1 | Epoch 6/100 - Train Loss: 0.2752 | Val Loss: 0.0864
Run 1 | Epoch 7/100 - Train Loss: 0.2622 | Val Loss: 0.0751
Run 1 | Epoch 8/100 - Train Loss: 0.2728 | Val Loss: 0.1092
Run 1 | Epoch 9/100 - Train Loss: 0.2343 | Val Loss: 0.0731
```

```
Run 1 | Epoch 10/100 - Train Loss: 0.2028 | Val Loss: 0.0816
Run 1 | Epoch 11/100 - Train Loss: 0.2092 | Val Loss: 0.0692
Run 1 | Epoch 12/100 - Train Loss: 0.2015 | Val Loss: 0.0739
Run 1 | Epoch 13/100 - Train Loss: 0.2004 | Val Loss: 0.0805
Run 1 | Epoch 14/100 - Train Loss: 0.1975 | Val Loss: 0.0676
Run 1 | Epoch 15/100 - Train Loss: 0.1565 | Val Loss: 0.0692
Run 1 | Epoch 16/100 - Train Loss: 0.1331 | Val Loss: 0.0692
Run 1 | Epoch 17/100 - Train Loss: 0.1202 | Val Loss: 0.0693
Run 1 | Epoch 18/100 - Train Loss: 0.1174 | Val Loss: 0.0751
Run 1 | Epoch 19/100 - Train Loss: 0.1123 | Val Loss: 0.0864
Run 1 | Epoch 20/100 - Train Loss: 0.1164 | Val Loss: 0.0720
Run 1 | Epoch 21/100 - Train Loss: 0.1092 | Val Loss: 0.0708
Run 1 | Epoch 22/100 - Train Loss: 0.1142 | Val Loss: 0.0791
Run 1 | Epoch 23/100 - Train Loss: 0.1091 | Val Loss: 0.0751
Run 1 | Epoch 24/100 - Train Loss: 0.1110 | Val Loss: 0.0712
Run 1 | Epoch 25/100 - Train Loss: 0.1092 | Val Loss: 0.0784
Run 1 | Epoch 26/100 - Train Loss: 0.1014 | Val Loss: 0.0759
Run 1 | Epoch 27/100 - Train Loss: 0.1263 | Val Loss: 0.0709
Run 1 | Epoch 28/100 - Train Loss: 0.0972 | Val Loss: 0.0713
Run 1 | Epoch 29/100 - Train Loss: 0.0984 | Val Loss: 0.0797
Run 1 | Epoch 30/100 - Train Loss: 0.1117 | Val Loss: 0.0720
Run 1 | Epoch 31/100 - Train Loss: 0.0909 | Val Loss: 0.0928
Run 1 | Epoch 32/100 - Train Loss: 0.0843 | Val Loss: 0.0934
Run 1 | Epoch 33/100 - Train Loss: 0.0868 | Val Loss: 0.1001
Run 1 | Epoch 34/100 - Train Loss: 0.0805 | Val Loss: 0.1073
Run 1 | Epoch 35/100 - Train Loss: 0.0947 | Val Loss: 0.0805
Run 1 | Epoch 36/100 - Train Loss: 0.0925 | Val Loss: 0.0707
Run 1 | Epoch 37/100 - Train Loss: 0.0843 | Val Loss: 0.0810
Run 1 | Epoch 38/100 - Train Loss: 0.0889 | Val Loss: 0.0771
Run 1 | Epoch 39/100 - Train Loss: 0.0839 | Val Loss: 0.0833
Run 1 | Epoch 40/100 - Train Loss: 0.0890 | Val Loss: 0.1025
Run 1 | Epoch 41/100 - Train Loss: 0.0844 | Val Loss: 0.0745
Run 1 | Epoch 42/100 - Train Loss: 0.0798 | Val Loss: 0.0856
Run 1 | Epoch 43/100 - Train Loss: 0.0764 | Val Loss: 0.0738
Run 1 | Epoch 44/100 - Train Loss: 0.0735 | Val Loss: 0.0733
Run 1 | Epoch 45/100 - Train Loss: 0.0732 | Val Loss: 0.0851
Run 1 | Epoch 46/100 - Train Loss: 0.0724 | Val Loss: 0.0815
Run 1 | Epoch 47/100 - Train Loss: 0.0847 | Val Loss: 0.0676
Run 1 | Epoch 48/100 - Train Loss: 0.0724 | Val Loss: 0.0728
Run 1 | Epoch 49/100 - Train Loss: 0.0664 | Val Loss: 0.0925
Run 1 | Epoch 50/100 - Train Loss: 0.0769 | Val Loss: 0.0720
Run 1 | Epoch 51/100 - Train Loss: 0.0720 | Val Loss: 0.0997
Run 1 | Epoch 52/100 - Train Loss: 0.0721 | Val Loss: 0.0760
Run 1 | Epoch 53/100 - Train Loss: 0.0658 | Val Loss: 0.0813
Run 1 | Epoch 54/100 - Train Loss: 0.0721 | Val Loss: 0.0942
Run 1 | Epoch 55/100 - Train Loss: 0.0750 | Val Loss: 0.0852
Run 1 | Epoch 56/100 - Train Loss: 0.0671 | Val Loss: 0.0750
Run 1 | Epoch 57/100 - Train Loss: 0.0695 | Val Loss: 0.0698
```

```
Run 1 | Epoch 58/100 - Train Loss: 0.0666 | Val Loss: 0.0747
Run 1 | Epoch 59/100 - Train Loss: 0.0622 | Val Loss: 0.0738
Run 1 | Epoch 60/100 - Train Loss: 0.0735 | Val Loss: 0.0794
Run 1 | Epoch 61/100 - Train Loss: 0.0820 | Val Loss: 0.0681
Run 1 | Epoch 62/100 - Train Loss: 0.0696 | Val Loss: 0.0943
Run 1 | Epoch 63/100 - Train Loss: 0.0620 | Val Loss: 0.0911
Run 1 | Epoch 64/100 - Train Loss: 0.0592 | Val Loss: 0.0736
Run 1 | Epoch 65/100 - Train Loss: 0.0610 | Val Loss: 0.0734
Run 1 | Epoch 66/100 - Train Loss: 0.0629 | Val Loss: 0.0832
Run 1 | Epoch 67/100 - Train Loss: 0.0719 | Val Loss: 0.0664
Run 1 | Epoch 68/100 - Train Loss: 0.0666 | Val Loss: 0.0762
Run 1 | Epoch 69/100 - Train Loss: 0.0611 | Val Loss: 0.0766
Run 1 | Epoch 70/100 - Train Loss: 0.0593 | Val Loss: 0.0719
Run 1 | Epoch 71/100 - Train Loss: 0.0579 | Val Loss: 0.0801
Run 1 | Epoch 72/100 - Train Loss: 0.0618 | Val Loss: 0.0769
Run 1 | Epoch 73/100 - Train Loss: 0.0584 | Val Loss: 0.0751
Run 1 | Epoch 74/100 - Train Loss: 0.0549 | Val Loss: 0.0845
Run 1 | Epoch 75/100 - Train Loss: 0.0565 | Val Loss: 0.0842
Run 1 | Epoch 76/100 - Train Loss: 0.0567 | Val Loss: 0.0884
Run 1 | Epoch 77/100 - Train Loss: 0.0555 | Val Loss: 0.0836
Run 1 | Epoch 78/100 - Train Loss: 0.0539 | Val Loss: 0.0698
Run 1 | Epoch 79/100 - Train Loss: 0.0600 | Val Loss: 0.0698
Run 1 | Epoch 80/100 - Train Loss: 0.0523 | Val Loss: 0.0748
Run 1 | Epoch 81/100 - Train Loss: 0.0582 | Val Loss: 0.1009
Run 1 | Epoch 82/100 - Train Loss: 0.0529 | Val Loss: 0.0691
Run 1 | Epoch 83/100 - Train Loss: 0.0522 | Val Loss: 0.0728
Run 1 | Epoch 84/100 - Train Loss: 0.0550 | Val Loss: 0.0731
Run 1 | Epoch 85/100 - Train Loss: 0.0523 | Val Loss: 0.0840
Run 1 | Epoch 86/100 - Train Loss: 0.0538 | Val Loss: 0.0910
Run 1 | Epoch 87/100 - Train Loss: 0.0516 | Val Loss: 0.0827
Run 1 | Epoch 88/100 - Train Loss: 0.0921 | Val Loss: 0.0874
Run 1 | Epoch 89/100 - Train Loss: 0.0722 | Val Loss: 0.0857
Run 1 | Epoch 90/100 - Train Loss: 0.0570 | Val Loss: 0.0660
Run 1 | Epoch 91/100 - Train Loss: 0.0665 | Val Loss: 0.0643
Run 1 | Epoch 92/100 - Train Loss: 0.0501 | Val Loss: 0.0727
Run 1 | Epoch 93/100 - Train Loss: 0.0522 | Val Loss: 0.0742
Run 1 | Epoch 94/100 - Train Loss: 0.0543 | Val Loss: 0.0705
Run 1 | Epoch 95/100 - Train Loss: 0.0504 | Val Loss: 0.0864
Run 1 | Epoch 96/100 - Train Loss: 0.0479 | Val Loss: 0.0675
Run 1 | Epoch 97/100 - Train Loss: 0.0452 | Val Loss: 0.0845
Run 1 | Epoch 98/100 - Train Loss: 0.0479 | Val Loss: 0.0754
Run 1 | Epoch 99/100 - Train Loss: 0.0525 | Val Loss: 0.0727
Run 1 | Epoch 100/100 - Train Loss: 0.0447 | Val Loss: 0.0900

=== Training Run 2/3 ===

Run 2 | Epoch 1/100 - Train Loss: 5.6752 | Val Loss: 1.0592
Run 2 | Epoch 2/100 - Train Loss: 1.6040 | Val Loss: 0.0655
```

```
Run 2 | Epoch 3/100 - Train Loss: 0.6318 | Val Loss: 0.1318
Run 2 | Epoch 4/100 - Train Loss: 0.3678 | Val Loss: 0.1818
Run 2 | Epoch 5/100 - Train Loss: 0.2955 | Val Loss: 0.1105
Run 2 | Epoch 6/100 - Train Loss: 0.3134 | Val Loss: 0.1310
Run 2 | Epoch 7/100 - Train Loss: 0.2570 | Val Loss: 0.1079
Run 2 | Epoch 8/100 - Train Loss: 0.2203 | Val Loss: 0.0575
Run 2 | Epoch 9/100 - Train Loss: 0.2052 | Val Loss: 0.0656
Run 2 | Epoch 10/100 - Train Loss: 0.1575 | Val Loss: 0.0637
Run 2 | Epoch 11/100 - Train Loss: 0.1686 | Val Loss: 0.0743
Run 2 | Epoch 12/100 - Train Loss: 0.1860 | Val Loss: 0.0741
Run 2 | Epoch 13/100 - Train Loss: 0.1257 | Val Loss: 0.0778
Run 2 | Epoch 14/100 - Train Loss: 0.1226 | Val Loss: 0.0814
Run 2 | Epoch 15/100 - Train Loss: 0.1239 | Val Loss: 0.0751
Run 2 | Epoch 16/100 - Train Loss: 0.1293 | Val Loss: 0.0956
Run 2 | Epoch 17/100 - Train Loss: 0.1399 | Val Loss: 0.2234
Run 2 | Epoch 18/100 - Train Loss: 0.1437 | Val Loss: 0.0727
Run 2 | Epoch 19/100 - Train Loss: 0.1084 | Val Loss: 0.0733
Run 2 | Epoch 20/100 - Train Loss: 0.1084 | Val Loss: 0.0768
Run 2 | Epoch 21/100 - Train Loss: 0.1085 | Val Loss: 0.0748
Run 2 | Epoch 22/100 - Train Loss: 0.1040 | Val Loss: 0.0730
Run 2 | Epoch 23/100 - Train Loss: 0.1149 | Val Loss: 0.0863
Run 2 | Epoch 24/100 - Train Loss: 0.0981 | Val Loss: 0.0771
Run 2 | Epoch 25/100 - Train Loss: 0.0888 | Val Loss: 0.0668
Run 2 | Epoch 26/100 - Train Loss: 0.0899 | Val Loss: 0.0725
Run 2 | Epoch 27/100 - Train Loss: 0.0876 | Val Loss: 0.0725
Run 2 | Epoch 28/100 - Train Loss: 0.0927 | Val Loss: 0.0681
Run 2 | Epoch 29/100 - Train Loss: 0.0928 | Val Loss: 0.0812
Run 2 | Epoch 30/100 - Train Loss: 0.0996 | Val Loss: 0.0788
Run 2 | Epoch 31/100 - Train Loss: 0.0916 | Val Loss: 0.0754
Run 2 | Epoch 32/100 - Train Loss: 0.0824 | Val Loss: 0.0806
Run 2 | Epoch 33/100 - Train Loss: 0.0742 | Val Loss: 0.0804
Run 2 | Epoch 34/100 - Train Loss: 0.0837 | Val Loss: 0.0772
Run 2 | Epoch 35/100 - Train Loss: 0.0816 | Val Loss: 0.0760
Run 2 | Epoch 36/100 - Train Loss: 0.0805 | Val Loss: 0.0786
Run 2 | Epoch 37/100 - Train Loss: 0.0883 | Val Loss: 0.0766
Run 2 | Epoch 38/100 - Train Loss: 0.0772 | Val Loss: 0.0772
Run 2 | Epoch 39/100 - Train Loss: 0.0808 | Val Loss: 0.0768
Run 2 | Epoch 40/100 - Train Loss: 0.0764 | Val Loss: 0.0834
Run 2 | Epoch 41/100 - Train Loss: 0.0725 | Val Loss: 0.0833
Run 2 | Epoch 42/100 - Train Loss: 0.1070 | Val Loss: 0.0822
Run 2 | Epoch 43/100 - Train Loss: 0.0910 | Val Loss: 0.0846
Run 2 | Epoch 44/100 - Train Loss: 0.0833 | Val Loss: 0.0879
Run 2 | Epoch 45/100 - Train Loss: 0.0773 | Val Loss: 0.0730
Run 2 | Epoch 46/100 - Train Loss: 0.0775 | Val Loss: 0.0800
Run 2 | Epoch 47/100 - Train Loss: 0.0808 | Val Loss: 0.0813
Run 2 | Epoch 48/100 - Train Loss: 0.0680 | Val Loss: 0.0915
Run 2 | Epoch 49/100 - Train Loss: 0.0742 | Val Loss: 0.0741
Run 2 | Epoch 50/100 - Train Loss: 0.0704 | Val Loss: 0.0973
```

```
Run 2 | Epoch 51/100 - Train Loss: 0.0740 | Val Loss: 0.0747
Run 2 | Epoch 52/100 - Train Loss: 0.0665 | Val Loss: 0.0769
Run 2 | Epoch 53/100 - Train Loss: 0.0666 | Val Loss: 0.0748
Run 2 | Epoch 54/100 - Train Loss: 0.0668 | Val Loss: 0.0745
Run 2 | Epoch 55/100 - Train Loss: 0.0759 | Val Loss: 0.0806
Run 2 | Epoch 56/100 - Train Loss: 0.0696 | Val Loss: 0.0964
Run 2 | Epoch 57/100 - Train Loss: 0.0679 | Val Loss: 0.0771
Run 2 | Epoch 58/100 - Train Loss: 0.0716 | Val Loss: 0.1074
Run 2 | Epoch 59/100 - Train Loss: 0.0907 | Val Loss: 0.0910
Run 2 | Epoch 60/100 - Train Loss: 0.0719 | Val Loss: 0.0707
Run 2 | Epoch 61/100 - Train Loss: 0.0639 | Val Loss: 0.0730
Run 2 | Epoch 62/100 - Train Loss: 0.0694 | Val Loss: 0.0730
Run 2 | Epoch 63/100 - Train Loss: 0.0816 | Val Loss: 0.0918
Run 2 | Epoch 64/100 - Train Loss: 0.0694 | Val Loss: 0.0864
Run 2 | Epoch 65/100 - Train Loss: 0.0637 | Val Loss: 0.0810
Run 2 | Epoch 66/100 - Train Loss: 0.0612 | Val Loss: 0.0695
Run 2 | Epoch 67/100 - Train Loss: 0.0621 | Val Loss: 0.0772
Run 2 | Epoch 68/100 - Train Loss: 0.0604 | Val Loss: 0.0762
Run 2 | Epoch 69/100 - Train Loss: 0.0591 | Val Loss: 0.0875
Run 2 | Epoch 70/100 - Train Loss: 0.0634 | Val Loss: 0.0712
Run 2 | Epoch 71/100 - Train Loss: 0.0618 | Val Loss: 0.0742
Run 2 | Epoch 72/100 - Train Loss: 0.0591 | Val Loss: 0.0727
Run 2 | Epoch 73/100 - Train Loss: 0.0581 | Val Loss: 0.0733
Run 2 | Epoch 74/100 - Train Loss: 0.0553 | Val Loss: 0.0708
Run 2 | Epoch 75/100 - Train Loss: 0.0563 | Val Loss: 0.0877
Run 2 | Epoch 76/100 - Train Loss: 0.0593 | Val Loss: 0.0729
Run 2 | Epoch 77/100 - Train Loss: 0.0557 | Val Loss: 0.0709
Run 2 | Epoch 78/100 - Train Loss: 0.0703 | Val Loss: 0.0686
Run 2 | Epoch 79/100 - Train Loss: 0.0712 | Val Loss: 0.0808
Run 2 | Epoch 80/100 - Train Loss: 0.0592 | Val Loss: 0.0722
Run 2 | Epoch 81/100 - Train Loss: 0.0576 | Val Loss: 0.0743
Run 2 | Epoch 82/100 - Train Loss: 0.0558 | Val Loss: 0.0722
Run 2 | Epoch 83/100 - Train Loss: 0.0524 | Val Loss: 0.0739
Run 2 | Epoch 84/100 - Train Loss: 0.0526 | Val Loss: 0.0742
Run 2 | Epoch 85/100 - Train Loss: 0.0541 | Val Loss: 0.0713
Run 2 | Epoch 86/100 - Train Loss: 0.0547 | Val Loss: 0.0718
Run 2 | Epoch 87/100 - Train Loss: 0.0496 | Val Loss: 0.0693
Run 2 | Epoch 88/100 - Train Loss: 0.0531 | Val Loss: 0.0719
Run 2 | Epoch 89/100 - Train Loss: 0.0671 | Val Loss: 0.0809
Run 2 | Epoch 90/100 - Train Loss: 0.0541 | Val Loss: 0.0702
Run 2 | Epoch 91/100 - Train Loss: 0.0507 | Val Loss: 0.0672
Run 2 | Epoch 92/100 - Train Loss: 0.0486 | Val Loss: 0.0718
Run 2 | Epoch 93/100 - Train Loss: 0.0486 | Val Loss: 0.0692
Run 2 | Epoch 94/100 - Train Loss: 0.0528 | Val Loss: 0.0653
Run 2 | Epoch 95/100 - Train Loss: 0.0546 | Val Loss: 0.0701
Run 2 | Epoch 96/100 - Train Loss: 0.0490 | Val Loss: 0.0788
Run 2 | Epoch 97/100 - Train Loss: 0.0472 | Val Loss: 0.0707
Run 2 | Epoch 98/100 - Train Loss: 0.0452 | Val Loss: 0.0672
```

```
Run 2 | Epoch 99/100 - Train Loss: 0.0461 | Val Loss: 0.0717
Run 2 | Epoch 100/100 - Train Loss: 0.0471 | Val Loss: 0.0688

=== Training Run 3/3 ===

Run 3 | Epoch 1/100 - Train Loss: 3.7300 | Val Loss: 0.1372
Run 3 | Epoch 2/100 - Train Loss: 0.9523 | Val Loss: 0.0724
Run 3 | Epoch 3/100 - Train Loss: 0.5008 | Val Loss: 0.0772
Run 3 | Epoch 4/100 - Train Loss: 0.3912 | Val Loss: 0.1128
Run 3 | Epoch 5/100 - Train Loss: 0.3035 | Val Loss: 0.1292
Run 3 | Epoch 6/100 - Train Loss: 0.2657 | Val Loss: 0.1489
Run 3 | Epoch 7/100 - Train Loss: 0.2312 | Val Loss: 0.1432
Run 3 | Epoch 8/100 - Train Loss: 0.2310 | Val Loss: 0.0910
Run 3 | Epoch 9/100 - Train Loss: 0.2187 | Val Loss: 0.1068
Run 3 | Epoch 10/100 - Train Loss: 0.2217 | Val Loss: 0.0678
Run 3 | Epoch 11/100 - Train Loss: 0.2153 | Val Loss: 0.0958
Run 3 | Epoch 12/100 - Train Loss: 0.1896 | Val Loss: 0.0976
Run 3 | Epoch 13/100 - Train Loss: 0.1450 | Val Loss: 0.0884
Run 3 | Epoch 14/100 - Train Loss: 0.1364 | Val Loss: 0.0811
Run 3 | Epoch 15/100 - Train Loss: 0.1257 | Val Loss: 0.0733
Run 3 | Epoch 16/100 - Train Loss: 0.1146 | Val Loss: 0.1054
Run 3 | Epoch 17/100 - Train Loss: 0.1109 | Val Loss: 0.0934
Run 3 | Epoch 18/100 - Train Loss: 0.1107 | Val Loss: 0.0774
Run 3 | Epoch 19/100 - Train Loss: 0.1150 | Val Loss: 0.0856
Run 3 | Epoch 20/100 - Train Loss: 0.1294 | Val Loss: 0.0920
Run 3 | Epoch 21/100 - Train Loss: 0.1050 | Val Loss: 0.0801
Run 3 | Epoch 22/100 - Train Loss: 0.0918 | Val Loss: 0.0906
Run 3 | Epoch 23/100 - Train Loss: 0.0936 | Val Loss: 0.0725
Run 3 | Epoch 24/100 - Train Loss: 0.1182 | Val Loss: 0.0778
Run 3 | Epoch 25/100 - Train Loss: 0.1109 | Val Loss: 0.0897
Run 3 | Epoch 26/100 - Train Loss: 0.0984 | Val Loss: 0.0854
Run 3 | Epoch 27/100 - Train Loss: 0.0893 | Val Loss: 0.0944
Run 3 | Epoch 28/100 - Train Loss: 0.0886 | Val Loss: 0.0775
Run 3 | Epoch 29/100 - Train Loss: 0.0987 | Val Loss: 0.0757
Run 3 | Epoch 30/100 - Train Loss: 0.0864 | Val Loss: 0.0820
Run 3 | Epoch 31/100 - Train Loss: 0.0800 | Val Loss: 0.0923
Run 3 | Epoch 32/100 - Train Loss: 0.0855 | Val Loss: 0.0878
Run 3 | Epoch 33/100 - Train Loss: 0.1021 | Val Loss: 0.0746
Run 3 | Epoch 34/100 - Train Loss: 0.0868 | Val Loss: 0.0764
Run 3 | Epoch 35/100 - Train Loss: 0.0789 | Val Loss: 0.0679
Run 3 | Epoch 36/100 - Train Loss: 0.0784 | Val Loss: 0.0686
Run 3 | Epoch 37/100 - Train Loss: 0.0829 | Val Loss: 0.0743
Run 3 | Epoch 38/100 - Train Loss: 0.0967 | Val Loss: 0.0715
Run 3 | Epoch 39/100 - Train Loss: 0.1130 | Val Loss: 0.0685
Run 3 | Epoch 40/100 - Train Loss: 0.0866 | Val Loss: 0.0721
Run 3 | Epoch 41/100 - Train Loss: 0.0869 | Val Loss: 0.0687
Run 3 | Epoch 42/100 - Train Loss: 0.0823 | Val Loss: 0.0680
Run 3 | Epoch 43/100 - Train Loss: 0.0743 | Val Loss: 0.0820
```

```
Run 3 | Epoch 44/100 - Train Loss: 0.0748 | Val Loss: 0.0718
Run 3 | Epoch 45/100 - Train Loss: 0.0773 | Val Loss: 0.0755
Run 3 | Epoch 46/100 - Train Loss: 0.0691 | Val Loss: 0.0702
Run 3 | Epoch 47/100 - Train Loss: 0.0692 | Val Loss: 0.0747
Run 3 | Epoch 48/100 - Train Loss: 0.0721 | Val Loss: 0.0714
Run 3 | Epoch 49/100 - Train Loss: 0.1200 | Val Loss: 0.0839
Run 3 | Epoch 50/100 - Train Loss: 0.0740 | Val Loss: 0.0770
Run 3 | Epoch 51/100 - Train Loss: 0.0653 | Val Loss: 0.0704
Run 3 | Epoch 52/100 - Train Loss: 0.0761 | Val Loss: 0.0810
Run 3 | Epoch 53/100 - Train Loss: 0.0797 | Val Loss: 0.0741
Run 3 | Epoch 54/100 - Train Loss: 0.0827 | Val Loss: 0.0721
Run 3 | Epoch 55/100 - Train Loss: 0.0651 | Val Loss: 0.0751
Run 3 | Epoch 56/100 - Train Loss: 0.0680 | Val Loss: 0.0862
Run 3 | Epoch 57/100 - Train Loss: 0.0633 | Val Loss: 0.0711
Run 3 | Epoch 58/100 - Train Loss: 0.0624 | Val Loss: 0.0746
Run 3 | Epoch 59/100 - Train Loss: 0.0688 | Val Loss: 0.0693
Run 3 | Epoch 60/100 - Train Loss: 0.0671 | Val Loss: 0.0773
Run 3 | Epoch 61/100 - Train Loss: 0.0813 | Val Loss: 0.0826
Run 3 | Epoch 62/100 - Train Loss: 0.0728 | Val Loss: 0.0775
Run 3 | Epoch 63/100 - Train Loss: 0.0660 | Val Loss: 0.0651
Run 3 | Epoch 64/100 - Train Loss: 0.0615 | Val Loss: 0.0725
Run 3 | Epoch 65/100 - Train Loss: 0.0631 | Val Loss: 0.0686
Run 3 | Epoch 66/100 - Train Loss: 0.0567 | Val Loss: 0.0987
Run 3 | Epoch 67/100 - Train Loss: 0.0564 | Val Loss: 0.0813
Run 3 | Epoch 68/100 - Train Loss: 0.0583 | Val Loss: 0.0721
Run 3 | Epoch 69/100 - Train Loss: 0.0542 | Val Loss: 0.0785
Run 3 | Epoch 70/100 - Train Loss: 0.0577 | Val Loss: 0.0801
Run 3 | Epoch 71/100 - Train Loss: 0.0534 | Val Loss: 0.0678
Run 3 | Epoch 72/100 - Train Loss: 0.0538 | Val Loss: 0.0802
Run 3 | Epoch 73/100 - Train Loss: 0.0632 | Val Loss: 0.0764
Run 3 | Epoch 74/100 - Train Loss: 0.0655 | Val Loss: 0.0959
Run 3 | Epoch 75/100 - Train Loss: 0.0603 | Val Loss: 0.0648
Run 3 | Epoch 76/100 - Train Loss: 0.0588 | Val Loss: 0.0659
Run 3 | Epoch 77/100 - Train Loss: 0.0577 | Val Loss: 0.0990
Run 3 | Epoch 78/100 - Train Loss: 0.0542 | Val Loss: 0.0706
Run 3 | Epoch 79/100 - Train Loss: 0.0525 | Val Loss: 0.0929
Run 3 | Epoch 80/100 - Train Loss: 0.0536 | Val Loss: 0.0723
Run 3 | Epoch 81/100 - Train Loss: 0.0507 | Val Loss: 0.0677
Run 3 | Epoch 82/100 - Train Loss: 0.0505 | Val Loss: 0.0912
Run 3 | Epoch 83/100 - Train Loss: 0.0494 | Val Loss: 0.0733
Run 3 | Epoch 84/100 - Train Loss: 0.0529 | Val Loss: 0.0770
Run 3 | Epoch 85/100 - Train Loss: 0.0497 | Val Loss: 0.0921
Run 3 | Epoch 86/100 - Train Loss: 0.0528 | Val Loss: 0.0712
Run 3 | Epoch 87/100 - Train Loss: 0.0565 | Val Loss: 0.0691
Run 3 | Epoch 88/100 - Train Loss: 0.0633 | Val Loss: 0.0709
Run 3 | Epoch 89/100 - Train Loss: 0.0497 | Val Loss: 0.0835
Run 3 | Epoch 90/100 - Train Loss: 0.0498 | Val Loss: 0.0800
Run 3 | Epoch 91/100 - Train Loss: 0.0495 | Val Loss: 0.0797
```

```
Run 3 | Epoch 92/100 - Train Loss: 0.0493 | Val Loss: 0.0707
Run 3 | Epoch 93/100 - Train Loss: 0.0470 | Val Loss: 0.0710
Run 3 | Epoch 94/100 - Train Loss: 0.0459 | Val Loss: 0.0919
Run 3 | Epoch 95/100 - Train Loss: 0.0561 | Val Loss: 0.0686
Run 3 | Epoch 96/100 - Train Loss: 0.0480 | Val Loss: 0.0709
Run 3 | Epoch 97/100 - Train Loss: 0.0480 | Val Loss: 0.0643
Run 3 | Epoch 98/100 - Train Loss: 0.0475 | Val Loss: 0.0775
Run 3 | Epoch 99/100 - Train Loss: 0.0468 | Val Loss: 0.0745
Run 3 | Epoch 100/100 - Train Loss: 0.0451 | Val Loss: 0.0741
```

[16]:
```python
def evaluate_model(model_path, test_loader):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model = SonarCNNLSTM()
    model.load_state_dict(torch.load(model_path, map_location=device))
    model.to(device).eval()

    criterion = nn.MSELoss(reduction='none')
    all_losses = []

    with torch.no_grad():
        for batch in test_loader:
            imgs = batch['images'].to(device)
            targets = batch['targets'].to(device)  # ← fix here
            outputs = model(imgs)
            loss = criterion(outputs, targets)
            all_losses.append(loss.cpu())

    losses = torch.cat(all_losses, dim=0).numpy()
    mse_per_target = losses.mean(axis=0)
    overall_mse = mse_per_target.mean()
    return mse_per_target, overall_mse
```

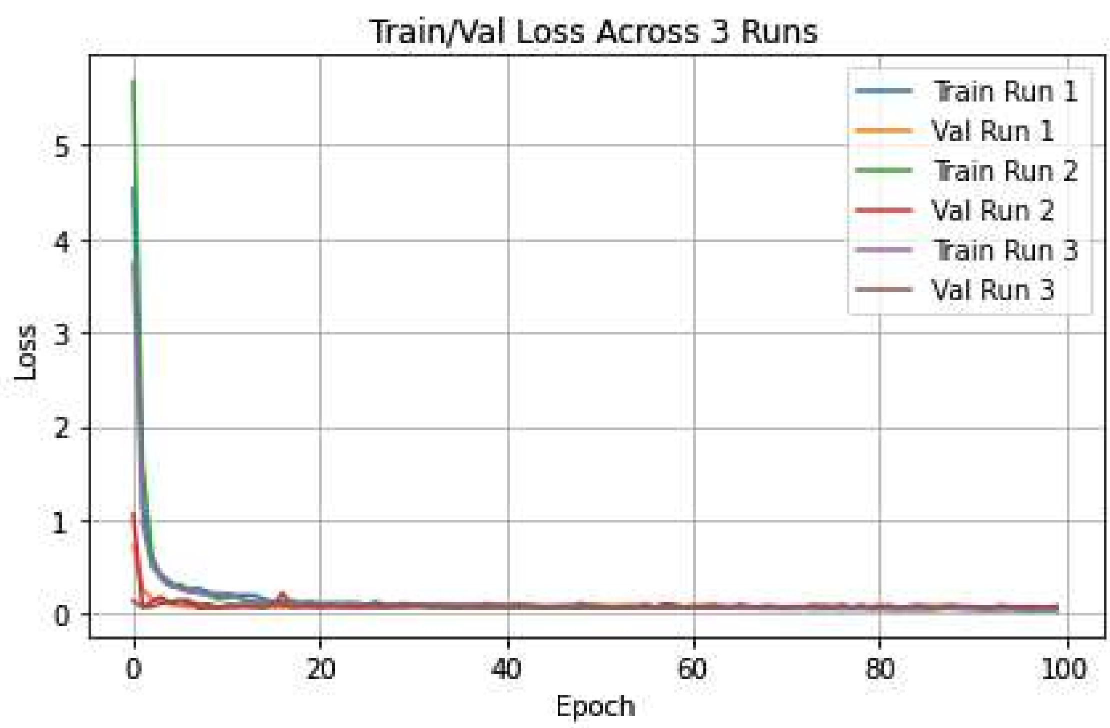
[17]:
```python
for i in range(1, 4):
    print(f"\nEvaluation of Run {i}")
    mse_vals, mse_overall = evaluate_model(f'model_run{i}.pt', test_loader)
    print(f"  Speed MSE: {mse_vals[0]:.4f}")
    print(f"  COG MSE:   {mse_vals[1]:.4f}")
    print(f"  Depth MSE: {mse_vals[2]:.4f}")
    print(f"  Overall MSE: {mse_overall:.4f}")
```

```
Evaluation of Run 1
  Speed MSE: 0.0969
  COG MSE:   7.3301
  Depth MSE: 6.3422
  Overall MSE: 4.5898

Evaluation of Run 2
```

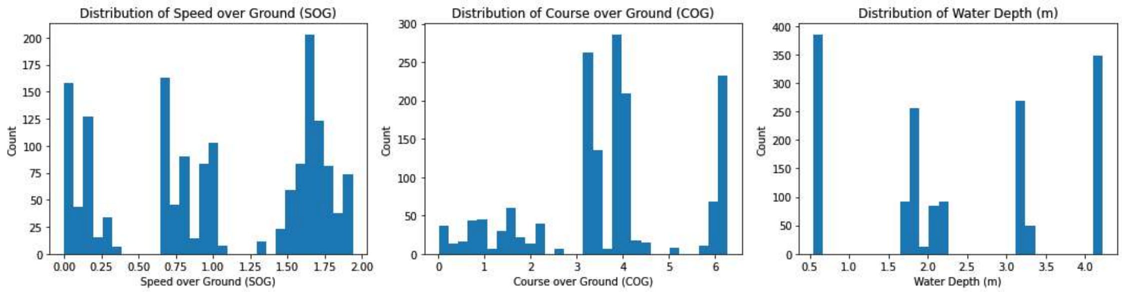
```
  Speed MSE: 0.0920
  COG MSE:   6.5768
  Depth MSE: 6.6812
  Overall MSE: 4.4500

Evaluation of Run 3
  Speed MSE: 0.1059
  COG MSE:   6.4471
  Depth MSE: 6.5358
  Overall MSE: 4.3629
```

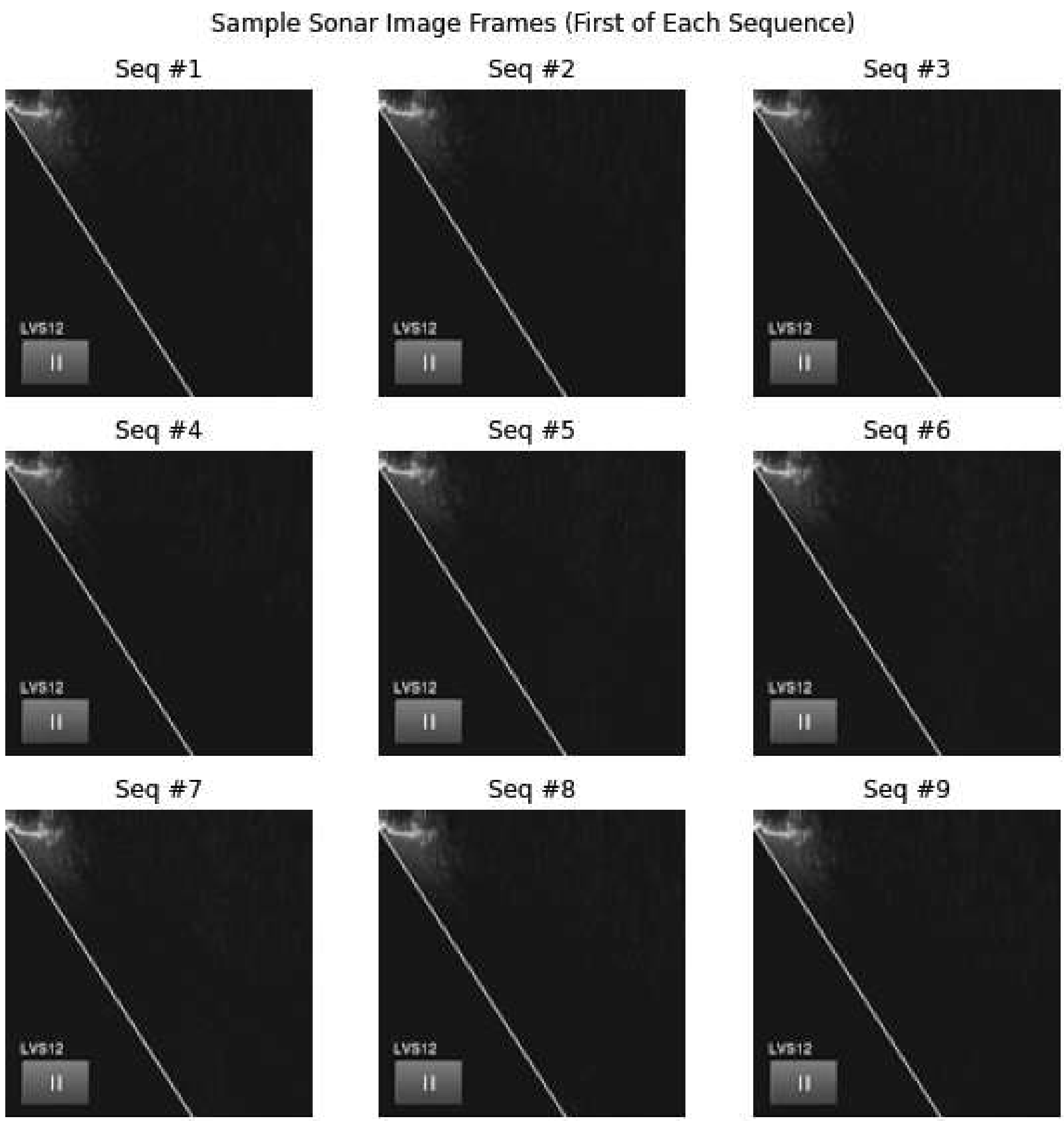
[18]:

```python
import matplotlib.pyplot as plt

for i, (train_loss, val_loss) in enumerate(all_losses, 1):
    plt.plot(train_loss, label=f'Train Run {i}')
    plt.plot(val_loss, label=f'Val Run {i}')

plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Train/Val Loss Across 3 Runs')
plt.legend()
plt.grid()
plt.tight_layout()
plt.show()
```



[ ]:
```
# added this from a different file# to visualize the data distribution and␣
↪sample images (meant to be at the start of the notebook)

import pandas as pd
import matplotlib.pyplot as plt
from PIL import Image
import os

train_df = pd.read_csv('train_split.csv')

fig, axes = plt.subplots(1, 3, figsize=(15, 4))
targets = ['sog_speed', 'cog_speed', 'waterdepth']
titles = ['Speed over Ground (SOG)', 'Course over Ground (COG)', 'Water Depth␣
↪(m)']

for ax, col, title in zip(axes, targets, titles):
    ax.hist(train_df[col].dropna(), bins=30)
    ax.set_title(f'Distribution of {title}')
    ax.set_xlabel(title)
    ax.set_ylabel('Count')

plt.tight_layout()
plt.show()

# Display the first frame of the first 9 sequences
fig, axes = plt.subplots(3, 3, figsize=(8, 8))
for i in range(9):
    row = train_df.iloc[i]
    img_path = os.path.join(
        'advanceaimethodsallfilestobeextracted22222222',
        row['subset'],
        row['filepath']
    )
    img = Image.open(img_path).convert('L').resize((128,128))
    ax = axes[i//3, i%3]
    ax.imshow(img, cmap='gray')
    ax.axis('off')
    ax.set_title(f"Seq #{i+1}")

plt.suptitle('Sample Sonar Image Frames (First of Each Sequence)')
plt.tight_layout()
plt.show()
```


Distribution of Speed over Ground (SOG)
Count
Speed over Ground (SOG)
Distribution of Course over Ground (COG)
Count
Course over Ground (COG)
Distribution of Water Depth (m)
Count
Water Depth (m)


Sample Sonar Image Frames (First of Each Sequence)
Seq #1
Seq #2
Seq #3
Seq #4
Seq #5
Seq #6
Seq #7
Seq #8
Seq #9