

Experiment_1_best_1

November 21, 2024

1 Experiment 1

A BiTCN Implementation that supports synthesized signals of arbitrary length and external datasets.

```
[1]: __author__ = "JUN WEI WANG"
     __email__ = "wjw_03@outlook.com"
```

```
[2]: import torch
     import numpy as np
     import os
     from os import path

     # Check for CUDA!
     device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
     print(f"Currently training on: {device}")

     MODE = "TRAINING"
     # MODE = "INFERENCE"

     print(os.getcwd())
```

Currently training on: cuda
C:\Users\wjw_0\dsp-project

2 Data Generator

```
[3]: import src.generator.QAM as QAM

     SEED = 1
     CONSTELLATION = "QAM"
     QAM_ORDER = 64
     NUMBEROFSYMBOLS = 51200

     cons = QAM.read_constellation_file(QAM_ORDER, CONSTELLATION)

     attenuated_data, original_data, raw_data, QAM_bits, preamble_data, ↵
     ↪insertion_index = QAM.generate_seq(
```

```

        order=QAM_ORDER,
        cons=cons,
        num_symbols=NUMBEROFSYMBOLS,
        seed=SEED
    )

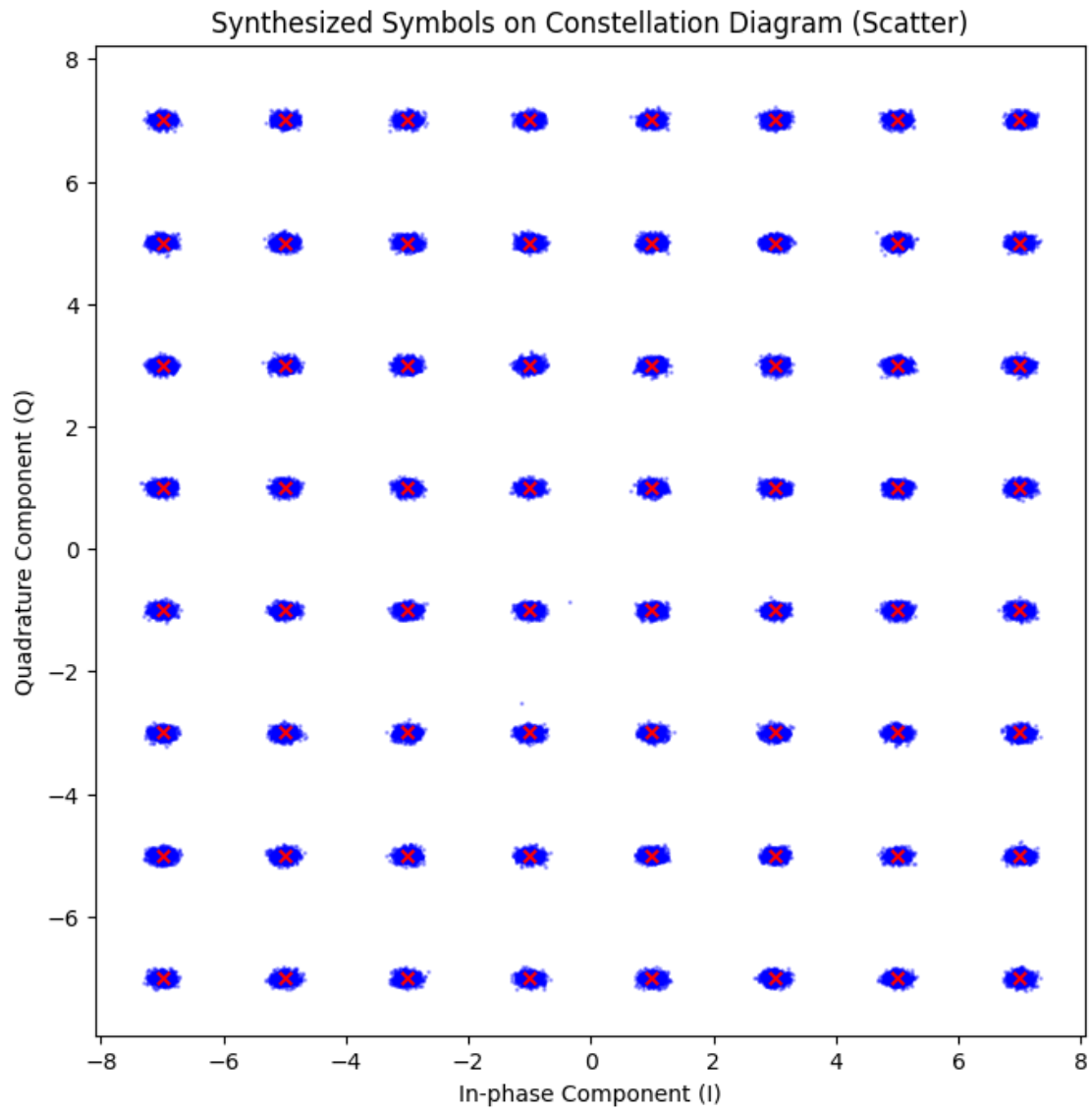
    og_demodulated_symbols, og_recoverdata = QAM.demodulation(
        original_data,
        NUMBEROFSYMBOLS,
        cons,
    )

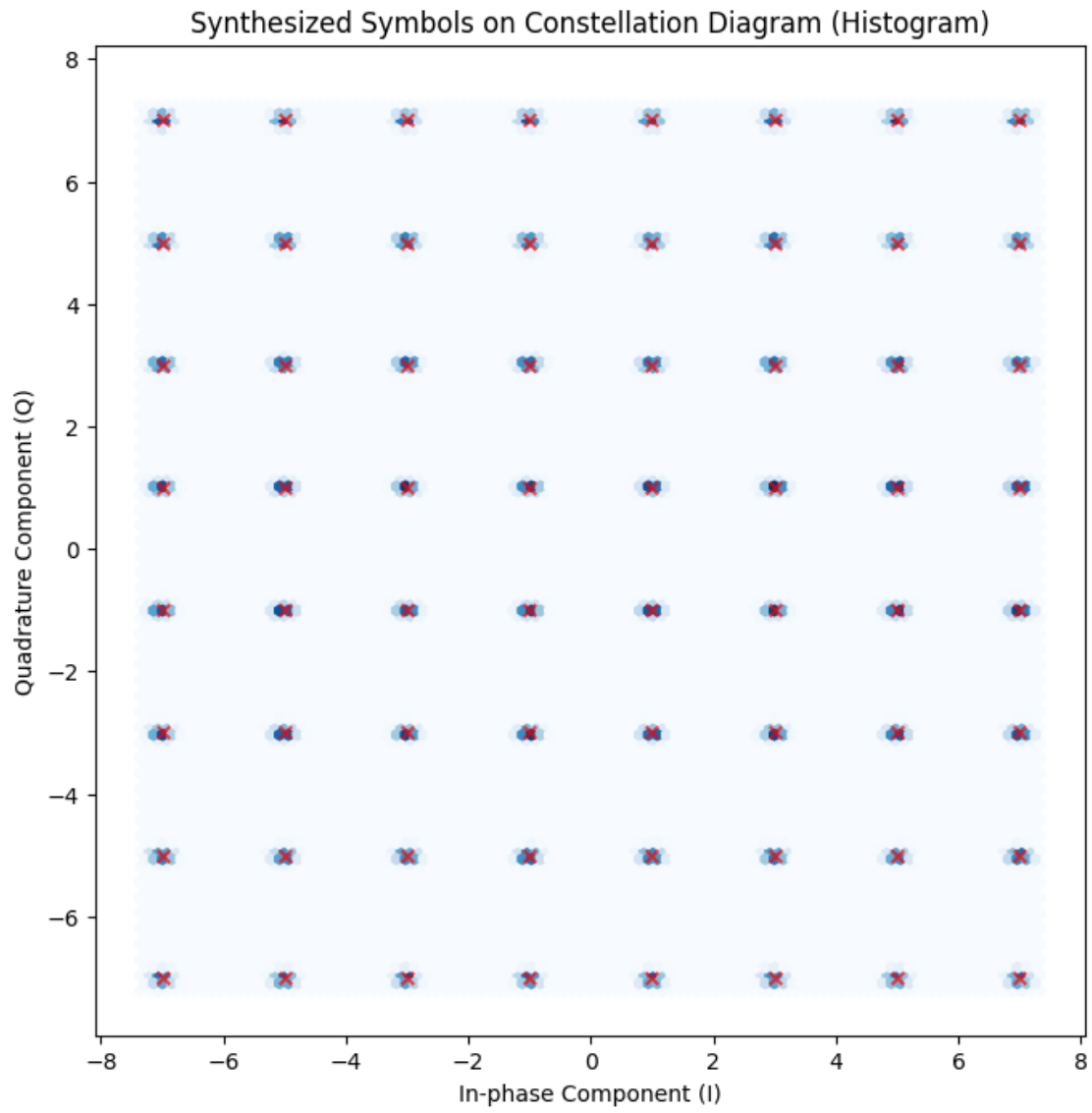
    attenurated_data_demodulated_symbols, attenurated_data_recoverdata = QAM.
    ↪demodulation(
        attenurated_data,
        NUMBEROFSYMBOLS,
        cons,
    )

    QAM.graph_IQ_constellation(
        "Synthesized Symbols",
        og_recoverdata,
        cons
    )

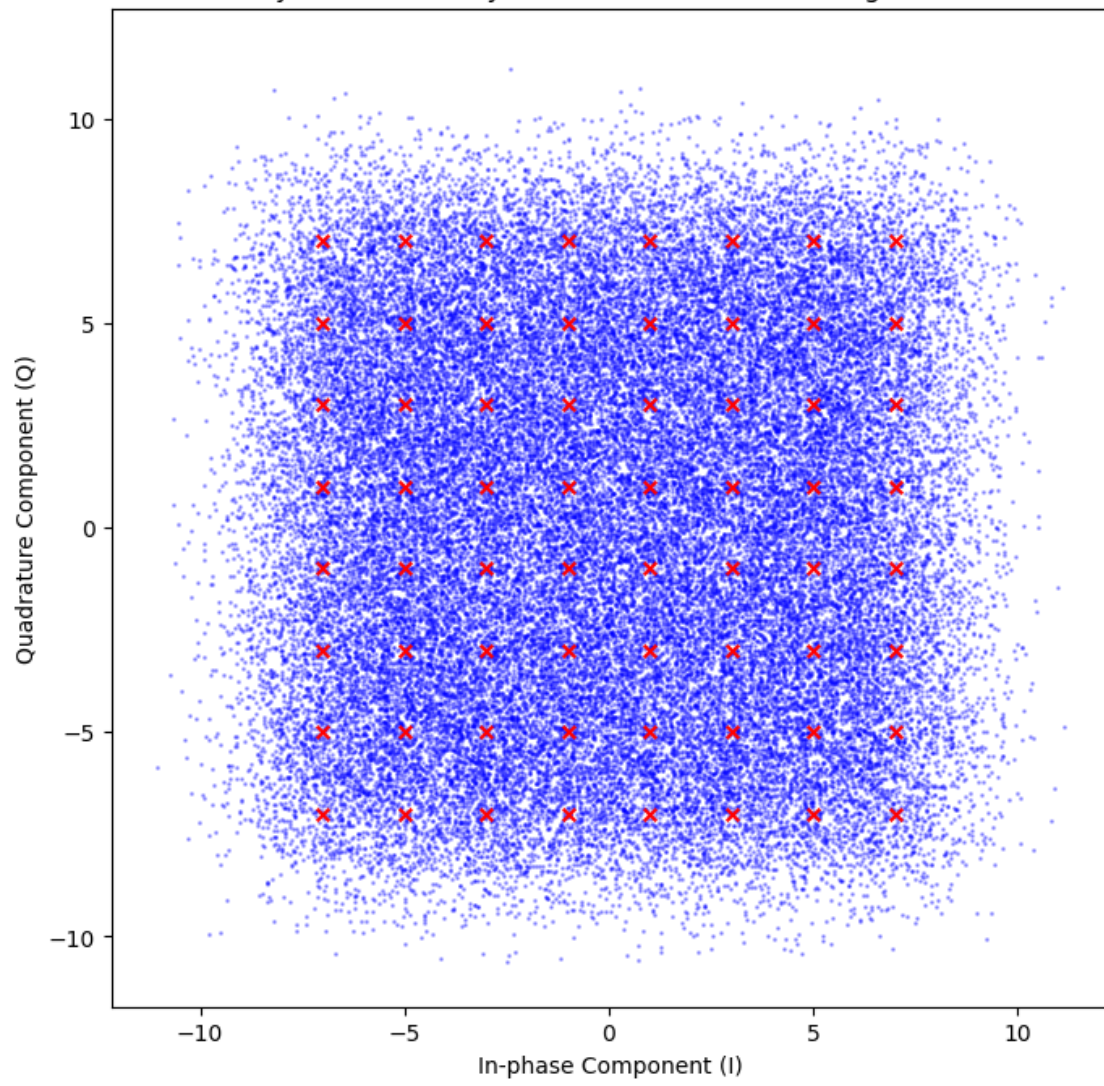
    QAM.graph_IQ_constellation(
        "Artificially Attenurated Symbols",
        attenurated_data_recoverdata,
        cons
    )

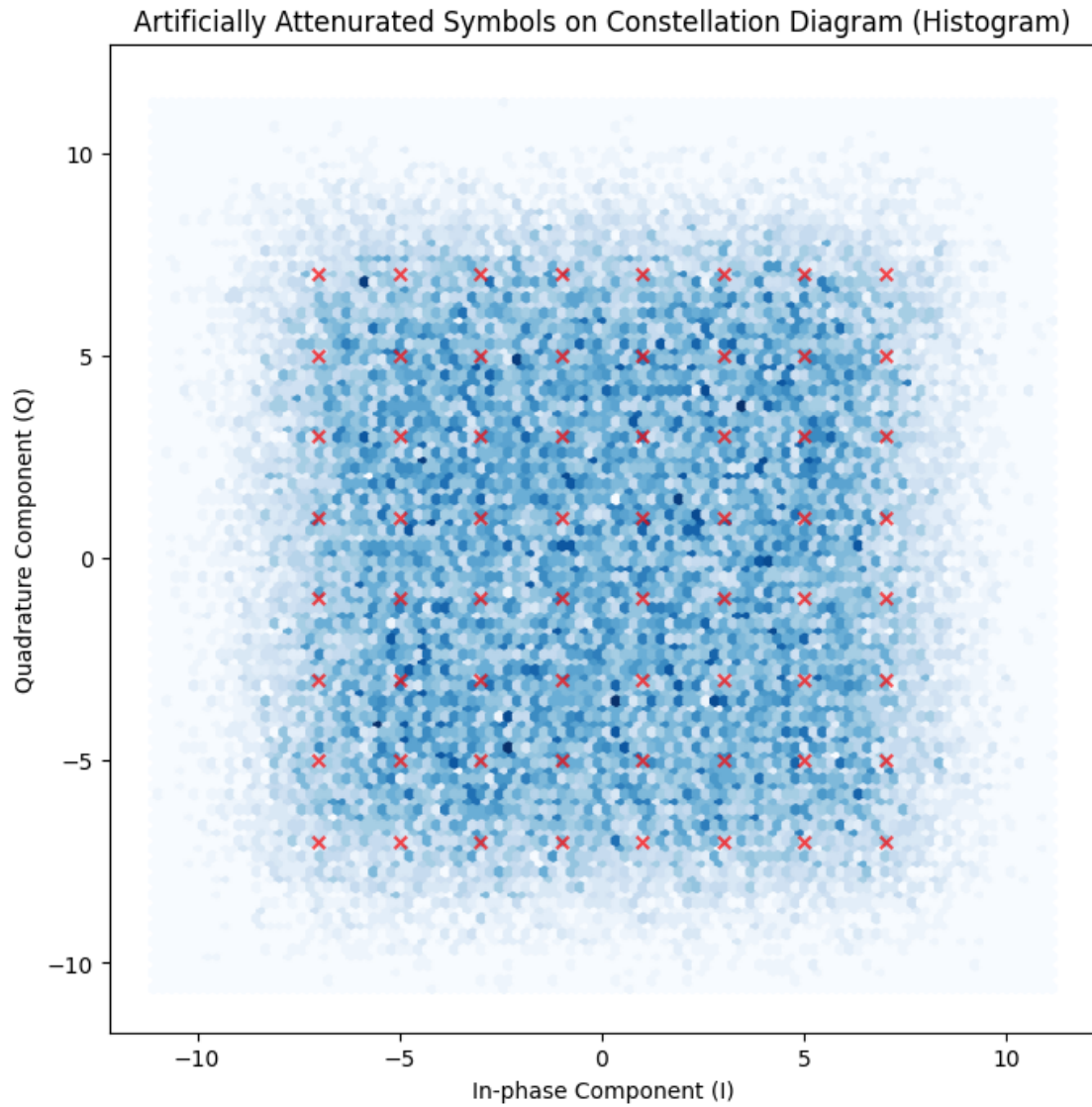
```





Artificially Attenuated Symbols on Constellation Diagram (Scatter)





3 Nerual Network

3.1 Initialize Constants, Dataset, and the NN Model

```
[4]: import torch.nn as nn
from torch.optim.lr_scheduler import StepLR
from tqdm import tqdm

# Neural Network
from src.nn.TCNN import BiTCN

# Dataset Tools
```

```

from torch.utils.data import DataLoader
from src.data.dataset import CustomDataset

```

```

[5]: # CONSTANTS
NN_MODEL = "BiTCN"
TARGET_DATASET = "384" #          SYNTH
# TARGET_DATASET = "SYNTH"
MODEL_EXT = ".pth"
MODEL_FILENAME = f"{NN_MODEL}_best_{TARGET_DATASET}.{MODEL_EXT}"

TRAIN_RATIO = 0.6

#####
#####                                PARAMETERS                                #####
#####

BATCH_SIZE = 512 # used to be 512
WINDOW_SIZE = 256

# Trainning parameters
DROPOUT = 0.01
N_EPOCHS = 20
LEARNING_RATE = 3e-4

# Network parameters
INPUT_SIZE = 1
OUTPUT_SIZE = 1
CHANNEL_SIZES = [32] * 5
KERNEL_SIZE = 16

```

3.2 Load Dataset

```

[6]: DATASET_FOLDER = "dataset"

REAL_DATA_H = lambda x : f"data64QAM.txt" # HOT-ENCODE
REAL_DATA = lambda x : f"OSC_sync_{x}.txt"
SYNTH_DATA = lambda x : f"SYNC_{x}.txt"
# NOTE: But for synthesized data, we can use our data generator implemented
↳ above

dataset: torch.utils.data.Dataset = None

try:
    if TARGET_DATASET == "SYNTH":
        # Using synthesized data, no need to read any files
        try:
            attenuated_data

```

```

        original_data
    except:
        raise "Please run data generation"
    # print(attenurated_data.reshape(-1, 1))
    dataset = CustomDataset(None, win_len=WINDOW_SIZE)
    datas, labels = dataset.split_sequence(
        attenurated_data.reshape(-1, 1),
        original_data.reshape(-1, 1),
        WINDOW_SIZE
    )
    size = len(attenurated_data)
    dataset.dataset, dataset.labels, dataset.size = \
        torch.tensor(datas, dtype=torch.float32), \
        torch.tensor(labels, dtype=torch.float32), \
        size
    pass
elif type(int(TARGET_DATASET)) == int:
    # Read data from dataset
    dataset = CustomDataset(
        DATASET_FOLDER,
        REAL_DATA(int(TARGET_DATASET)),
        REAL_DATA_H(-1),
        WINDOW_SIZE
    )
except ValueError:
    raise("Invalid target dataset")

print(f"{TARGET_DATASET} dataset loaded")
print(f"Dataset length: {len(dataset)} | Size: {dataset.size}")

# Split into train and validation sets

train_size = int(TRAIN_RATIO * len(dataset))
print(f"Traning ratio (train:valid): {TRAIN_RATIO}:{1 - TRAIN_RATIO}")
print(f"Training size: {train_size} | Validation size: {len(dataset) - \
    ↪train_size}")
val_size = len(dataset) - train_size
train_dataset, val_dataset = torch.utils.data.random_split(
    dataset, [train_size, val_size]
)

# Setup the training and validation data loaders
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE, shuffle=False)

```

384 dataset loaded

Dataset length: 204545 | Size: 204800

Traning ratio (train:valid): 0.6:0.4

Training size: 122727 | Validation size: 81818

3.3 Load Model

```
[7]: model: torch.nn.Module = None

if NN_MODEL == "BiTCN":
    model = BiTCN(
        INPUT_SIZE,
        OUTPUT_SIZE,
        CHANNEL_SIZES,
        KERNEL_SIZE,
        seq_len=WINDOW_SIZE,
        dropout=DROPOUT
    )
else:
    raise("Model is not defined!")

print(f"{NN_MODEL} Model successfully Loaded")
model.to(device)
```

BiTCN Model successfully Loaded

```
[7]: BiTCN(
  (tcn): TemporalConvNet(
    (network): Sequential(
      (0): TemporalBlock(
        (conv1): ParametrizedConv1d(
          1, 32, kernel_size=(16,), stride=(1,), padding=(15,)
          (parametrizations): ModuleDict(
            (weight): ParametrizationList(
              (0): _WeightNorm()
            )
          )
        )
      )
      (chomp1): Chomp1D()
      (relu1): PReLU(num_parameters=1)
      (dropout1): Dropout(p=0.01, inplace=False)
      (conv2): ParametrizedConv1d(
        32, 32, kernel_size=(16,), stride=(1,), padding=(15,)
        (parametrizations): ModuleDict(
          (weight): ParametrizationList(
            (0): _WeightNorm()
          )
        )
      )
      (chomp2): Chomp1D()
```

```

(relu2): PReLU(num_parameters=1)
(dropout2): Dropout(p=0.01, inplace=False)
(net): Sequential(
  (0): ParametrizedConv1d(
    1, 32, kernel_size=(16,), stride=(1,), padding=(15,)
    (parametrizations): ModuleDict(
      (weight): ParametrizationList(
        (0): _WeightNorm()
      )
    )
  )
  (1): Chomp1D()
  (2): PReLU(num_parameters=1)
  (3): Dropout(p=0.01, inplace=False)
  (4): ParametrizedConv1d(
    32, 32, kernel_size=(16,), stride=(1,), padding=(15,)
    (parametrizations): ModuleDict(
      (weight): ParametrizationList(
        (0): _WeightNorm()
      )
    )
  )
  (5): Chomp1D()
  (6): PReLU(num_parameters=1)
  (7): Dropout(p=0.01, inplace=False)
)
(downsample): Conv1d(1, 32, kernel_size=(1,), stride=(1,))
(relu): PReLU(num_parameters=1)
)
(1): TemporalBlock(
  (conv1): ParametrizedConv1d(
    32, 32, kernel_size=(16,), stride=(1,), padding=(30,), dilation=(2,)
    (parametrizations): ModuleDict(
      (weight): ParametrizationList(
        (0): _WeightNorm()
      )
    )
  )
  (chomp1): Chomp1D()
  (relu1): PReLU(num_parameters=1)
  (dropout1): Dropout(p=0.01, inplace=False)
  (conv2): ParametrizedConv1d(
    32, 32, kernel_size=(16,), stride=(1,), padding=(30,), dilation=(2,)
    (parametrizations): ModuleDict(
      (weight): ParametrizationList(
        (0): _WeightNorm()
      )
    )
  )
)

```

```

    )
)
(chomp2): Chomp1D()
(rel2): PReLU(num_parameters=1)
(dropout2): Dropout(p=0.01, inplace=False)
(net): Sequential(
  (0): ParametrizedConv1d(
    32, 32, kernel_size=(16,), stride=(1,), padding=(30,), dilation=(2,)
    (parametrizations): ModuleDict(
      (weight): ParametrizationList(
        (0): _WeightNorm()
      )
    )
  )
  (1): Chomp1D()
  (2): PReLU(num_parameters=1)
  (3): Dropout(p=0.01, inplace=False)
  (4): ParametrizedConv1d(
    32, 32, kernel_size=(16,), stride=(1,), padding=(30,), dilation=(2,)
    (parametrizations): ModuleDict(
      (weight): ParametrizationList(
        (0): _WeightNorm()
      )
    )
  )
  (5): Chomp1D()
  (6): PReLU(num_parameters=1)
  (7): Dropout(p=0.01, inplace=False)
)
(relu): PReLU(num_parameters=1)
)
(2): TemporalBlock(
  (conv1): ParametrizedConv1d(
    32, 32, kernel_size=(16,), stride=(1,), padding=(60,), dilation=(4,)
    (parametrizations): ModuleDict(
      (weight): ParametrizationList(
        (0): _WeightNorm()
      )
    )
  )
)
(chomp1): Chomp1D()
(rel1): PReLU(num_parameters=1)
(dropout1): Dropout(p=0.01, inplace=False)
(conv2): ParametrizedConv1d(
  32, 32, kernel_size=(16,), stride=(1,), padding=(60,), dilation=(4,)
  (parametrizations): ModuleDict(
    (weight): ParametrizationList(

```

```

        (0): _WeightNorm()
    )
)
(chomp2): Chomp1D()
(rel2): PReLU(num_parameters=1)
(dropout2): Dropout(p=0.01, inplace=False)
(net): Sequential(
  (0): ParametrizedConv1d(
    32, 32, kernel_size=(16,), stride=(1,), padding=(60,), dilation=(4,)
    (parametrizations): ModuleDict(
      (weight): ParametrizationList(
        (0): _WeightNorm()
      )
    )
  )
  (1): Chomp1D()
  (2): PReLU(num_parameters=1)
  (3): Dropout(p=0.01, inplace=False)
  (4): ParametrizedConv1d(
    32, 32, kernel_size=(16,), stride=(1,), padding=(60,), dilation=(4,)
    (parametrizations): ModuleDict(
      (weight): ParametrizationList(
        (0): _WeightNorm()
      )
    )
  )
  (5): Chomp1D()
  (6): PReLU(num_parameters=1)
  (7): Dropout(p=0.01, inplace=False)
)
(relu): PReLU(num_parameters=1)
)
(3): TemporalBlock(
  (conv1): ParametrizedConv1d(
    32, 32, kernel_size=(16,), stride=(1,), padding=(120,), dilation=(8,)
    (parametrizations): ModuleDict(
      (weight): ParametrizationList(
        (0): _WeightNorm()
      )
    )
  )
  (chomp1): Chomp1D()
  (relu1): PReLU(num_parameters=1)
  (dropout1): Dropout(p=0.01, inplace=False)
  (conv2): ParametrizedConv1d(
    32, 32, kernel_size=(16,), stride=(1,), padding=(120,), dilation=(8,)

```

```

        (parametrizations): ModuleDict(
          (weight): ParametrizationList(
            (0): _WeightNorm()
          )
        )
      )
    )
    (chomp2): Chomp1D()
    (relu2): PReLU(num_parameters=1)
    (dropout2): Dropout(p=0.01, inplace=False)
    (net): Sequential(
      (0): ParametrizedConv1d(
        32, 32, kernel_size=(16,), stride=(1,), padding=(120,),
dilation=(8,)
        (parametrizations): ModuleDict(
          (weight): ParametrizationList(
            (0): _WeightNorm()
          )
        )
      )
    )
    (1): Chomp1D()
    (2): PReLU(num_parameters=1)
    (3): Dropout(p=0.01, inplace=False)
    (4): ParametrizedConv1d(
        32, 32, kernel_size=(16,), stride=(1,), padding=(120,),
dilation=(8,)
        (parametrizations): ModuleDict(
          (weight): ParametrizationList(
            (0): _WeightNorm()
          )
        )
      )
    )
    (5): Chomp1D()
    (6): PReLU(num_parameters=1)
    (7): Dropout(p=0.01, inplace=False)
  )
  (relu): PReLU(num_parameters=1)
)
(4): TemporalBlock(
  (conv1): ParametrizedConv1d(
    32, 32, kernel_size=(16,), stride=(1,), padding=(240,), dilation=(16,)
    (parametrizations): ModuleDict(
      (weight): ParametrizationList(
        (0): _WeightNorm()
      )
    )
  )
)
(chomp1): Chomp1D()

```

```

(relu1): PReLU(num_parameters=1)
(dropout1): Dropout(p=0.01, inplace=False)
(conv2): ParametrizedConv1d(
  32, 32, kernel_size=(16,), stride=(1,), padding=(240,), dilation=(16,)
  (parametrizations): ModuleDict(
    (weight): ParametrizationList(
      (0): _WeightNorm()
    )
  )
)
(chomp2): Chomp1D()
(relu2): PReLU(num_parameters=1)
(dropout2): Dropout(p=0.01, inplace=False)
(net): Sequential(
  (0): ParametrizedConv1d(
    32, 32, kernel_size=(16,), stride=(1,), padding=(240,),
dilation=(16,)
    (parametrizations): ModuleDict(
      (weight): ParametrizationList(
        (0): _WeightNorm()
      )
    )
  )
  (1): Chomp1D()
  (2): PReLU(num_parameters=1)
  (3): Dropout(p=0.01, inplace=False)
  (4): ParametrizedConv1d(
    32, 32, kernel_size=(16,), stride=(1,), padding=(240,),
dilation=(16,)
    (parametrizations): ModuleDict(
      (weight): ParametrizationList(
        (0): _WeightNorm()
      )
    )
  )
  (5): Chomp1D()
  (6): PReLU(num_parameters=1)
  (7): Dropout(p=0.01, inplace=False)
)
(relu): PReLU(num_parameters=1)
)
)
(linear): Linear(in_features=16384, out_features=1, bias=True)
)

```

3.4 Training

```
[8]: def train(model, device, train_loader, optimizer, criterion):
    model.train()
    total_loss = 0
    for X_batch, y_batch in tqdm(train_loader, desc="Training", leave=False):
        optimizer.zero_grad()
        output = model(X_batch.to(device))
        loss = criterion(output, y_batch.to(device))
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
        average_loss = total_loss / len(train_loader)
    return average_loss

def validate(model, device, val_loader, criterion):
    model.eval()
    total_loss = 0
    with torch.no_grad():
        for X_batch, y_batch in tqdm(val_loader, desc="Validation",
    ↪leave=False):
            output = model(X_batch.to(device))
            loss = criterion(output, y_batch.to(device))
            total_loss += loss.item()
        average_loss = total_loss / len(val_loader)
    return average_loss

if MODE == "TRAINING":
    criterion = nn.MSELoss()
    # optimizer = torch.optim.Adam(
    #     model.parameters(),
    #     lr = LEARNING_RATE
    # )
    optimizer = torch.optim.RMSprop(model.parameters(), lr=LEARNING_RATE,
    ↪alpha=0.99, eps=1e-8, weight_decay=0, momentum=0.8)
    scheduler = StepLR(optimizer, step_size=8, gamma=0.1)

    # Train the model
    train_losses = []
    val_losses = []

    best_val_loss = float('inf')

    for epoch in range(N_EPOCHS):
        print(f"Starting epoch {epoch + 1}/{N_EPOCHS}")

        train_loss = train(model, device, train_loader, optimizer, criterion)
```

```

        val_loss = validate(model, device, val_loader, criterion)
        train_losses.append(train_loss)
        val_losses.append(val_loss)
        scheduler.step()

        print(f"Epoch [{epoch + 1}/{N_EPOCHS}], Train Loss: {train_loss:.4f},  

        ↪Validation Loss: {val_loss:.4f}")
        if val_loss < best_val_loss:
            best_val_loss = val_loss
            torch.save(model.state_dict(), MODEL_FILENAME)
            print(f"Saved model with validation loss: {best_val_loss:.4f}")

```

Starting epoch 1/20

Epoch [1/20], Train Loss: 5.2500, Validation Loss: 0.0472

Saved model with validation loss: 0.0472

Starting epoch 2/20

Epoch [2/20], Train Loss: 0.0408, Validation Loss: 0.0296

Saved model with validation loss: 0.0296

Starting epoch 3/20

Epoch [3/20], Train Loss: 0.0310, Validation Loss: 0.0201

Saved model with validation loss: 0.0201

Starting epoch 4/20

Epoch [4/20], Train Loss: 0.0263, Validation Loss: 0.0208

Starting epoch 5/20

Epoch [5/20], Train Loss: 0.0218, Validation Loss: 0.0152

Saved model with validation loss: 0.0152

Starting epoch 6/20

Epoch [6/20], Train Loss: 0.0183, Validation Loss: 0.0173

Starting epoch 7/20

Epoch [7/20], Train Loss: 0.0170, Validation Loss: 0.0111

Saved model with validation loss: 0.0111

Starting epoch 8/20

Epoch [8/20], Train Loss: 0.0142, Validation Loss: 0.0095
Saved model with validation loss: 0.0095
Starting epoch 9/20

Epoch [9/20], Train Loss: 0.0100, Validation Loss: 0.0080
Saved model with validation loss: 0.0080
Starting epoch 10/20

Epoch [10/20], Train Loss: 0.0093, Validation Loss: 0.0074
Saved model with validation loss: 0.0074
Starting epoch 11/20

Epoch [11/20], Train Loss: 0.0087, Validation Loss: 0.0068
Saved model with validation loss: 0.0068
Starting epoch 12/20

Epoch [12/20], Train Loss: 0.0082, Validation Loss: 0.0066
Saved model with validation loss: 0.0066
Starting epoch 13/20

Epoch [13/20], Train Loss: 0.0078, Validation Loss: 0.0063
Saved model with validation loss: 0.0063
Starting epoch 14/20

Epoch [14/20], Train Loss: 0.0075, Validation Loss: 0.0061
Saved model with validation loss: 0.0061
Starting epoch 15/20

Epoch [15/20], Train Loss: 0.0072, Validation Loss: 0.0059
Saved model with validation loss: 0.0059
Starting epoch 16/20

Epoch [16/20], Train Loss: 0.0069, Validation Loss: 0.0060
Starting epoch 17/20

Epoch [17/20], Train Loss: 0.0065, Validation Loss: 0.0055
Saved model with validation loss: 0.0055
Starting epoch 18/20

Epoch [18/20], Train Loss: 0.0065, Validation Loss: 0.0055
Saved model with validation loss: 0.0055
Starting epoch 19/20

Epoch [19/20], Train Loss: 0.0064, Validation Loss: 0.0056
Starting epoch 20/20

Epoch [20/20], Train Loss: 0.0064, Validation Loss: 0.0055
Saved model with validation loss: 0.0055

3.5 Inference

```
[9]: from tqdm import tqdm
      from torch.utils.data import DataLoader
      import math

      from src.data.dataset import CustomDataset
      from src.nn.TCNN import BiTCN

      # Load the model
      model = BiTCN(
          INPUT_SIZE,
          OUTPUT_SIZE,
          CHANNEL_SIZES,
          KERNEL_SIZE,
          seq_len=WINDOW_SIZE,
          dropout=DROPOUT
      )
      model = model.to(device)

      model.load_state_dict(
          torch.load(MODEL_FILENAME, map_location=torch.device(device))
      )

      predictions = []

      def inference(model, device, data_loader):
          model.eval()
          output=[]
          with torch.no_grad():
              import time
              i = 0
```

```

        for X_batch, y_batch in tqdm(data_loader, desc="Validation",
↪leave=False):
            out = model(X_batch.to(device))
            output.append(out.detach().cpu())
            outputs = torch.cat(output, dim=0)
            outputs_np = outputs.numpy()
        return outputs_np.flatten()

data_loader = DataLoader(dataset, batch_size=BATCH_SIZE, shuffle=False)

predictions = inference(model, device, data_loader)
half_window = math.floor((WINDOW_SIZE - 1) // 2)

og_data: np.ndarray = None

if TARGET_DATASET == "SYNTH":
    og_data = original_data
else:
    with open(path.join(os.getcwd(), "dataset", REAL_DATA_H(-1)), 'r') as file:
        og_data = np.array([float(line.strip()) for line in file.readlines()])

diff = dataset.size - (len(predictions) + half_window*2)
prefix = og_data[:half_window + diff]
suffix = og_data[-half_window:]
final_predictions = np.concatenate([prefix, predictions, suffix])
final_predictions = final_predictions.reshape(-1,1)
# final_predictions = final_predictions.astype(np.float16)

np.savetxt("pred.txt", final_predictions, delimiter="\n")

```

C:\Users\wjl_0\AppData\Local\Temp\ipykernel_29384\3700146736.py:20:

FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value), which uses the default pickle module implicitly. It is possible to construct malicious pickle data which will execute arbitrary code during unpickling (See <https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models> for more details). In a future release, the default value for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpickling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowlisted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True` for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any issues related to this experimental feature.

```
torch.load(MODEL_FILENAME, map_location=torch.device(device))
```

Validation: 0%|

| 0/400 [00:00<?, ?it/s]

4 Post Processing

Demodulation & Visualization

```
[10]: cons = QAM.read_constellation_file(QAM_ORDER, CONSTELLATION)

demodulated_symbols, recoverdata = QAM.demodulation(
    final_predictions,
    NUMBEROFSYMBOLS,
    cons
)

QAM.graph_IQ_constellation(
    "Recovered Symbols",
    recoverdata,
    cons
)

if TARGET_DATASET == "SYNTH":
    target = np.zeros(len(QAM_bits), dtype=int)
    for i, sample in enumerate(recoverdata):
        distances = np.abs(sample - cons)
        target[i] = np.argmin(distances) # Find the nearest constellation point

    print(len(demodulated_symbols), len(target))
    print(demodulated_symbols[:10], target[:10])
    res = QAM.calculate_ber(demodulated_symbols, target, 35)
    print(f"BER: {res[1]}")
else:
    from src.data.file_handler import read_file
    from src.data.processing import parse_str

    data = parse_str(read_file(path.join(DATASET_FOLDER, REAL_DATA_H(-1))))
    data = torch.tensor(data, dtype=torch.float32)
    data = data.reshape(1, -1).flatten()

    # data = QAM.downsample(data, 4)

    temp1, temp2 = QAM.demodulation(
        data,
        NUMBEROFSYMBOLS,
        cons
    )

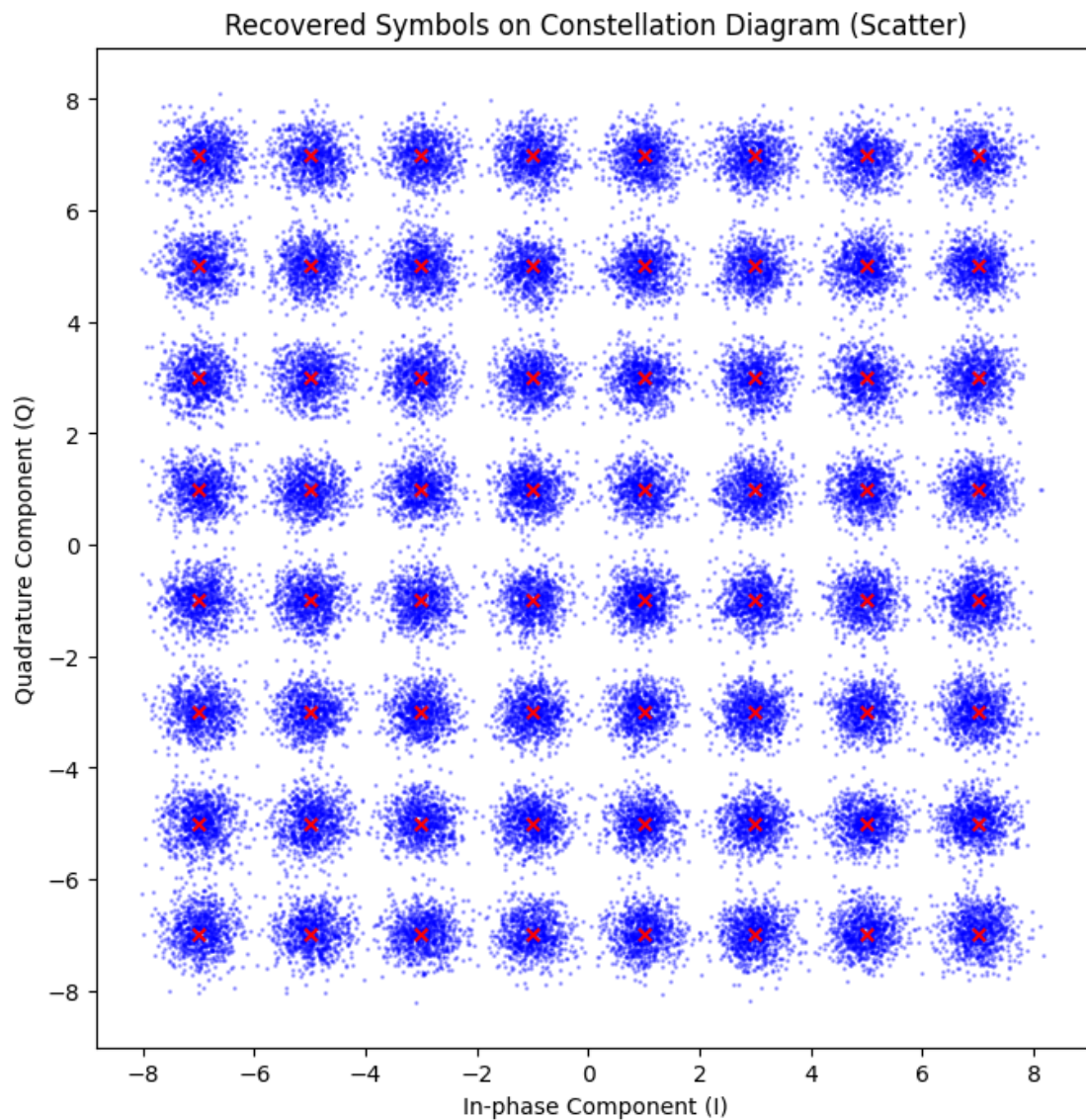
    QAM.graph_IQ_constellation(
        "Recovered Symbols",
        temp2,
```

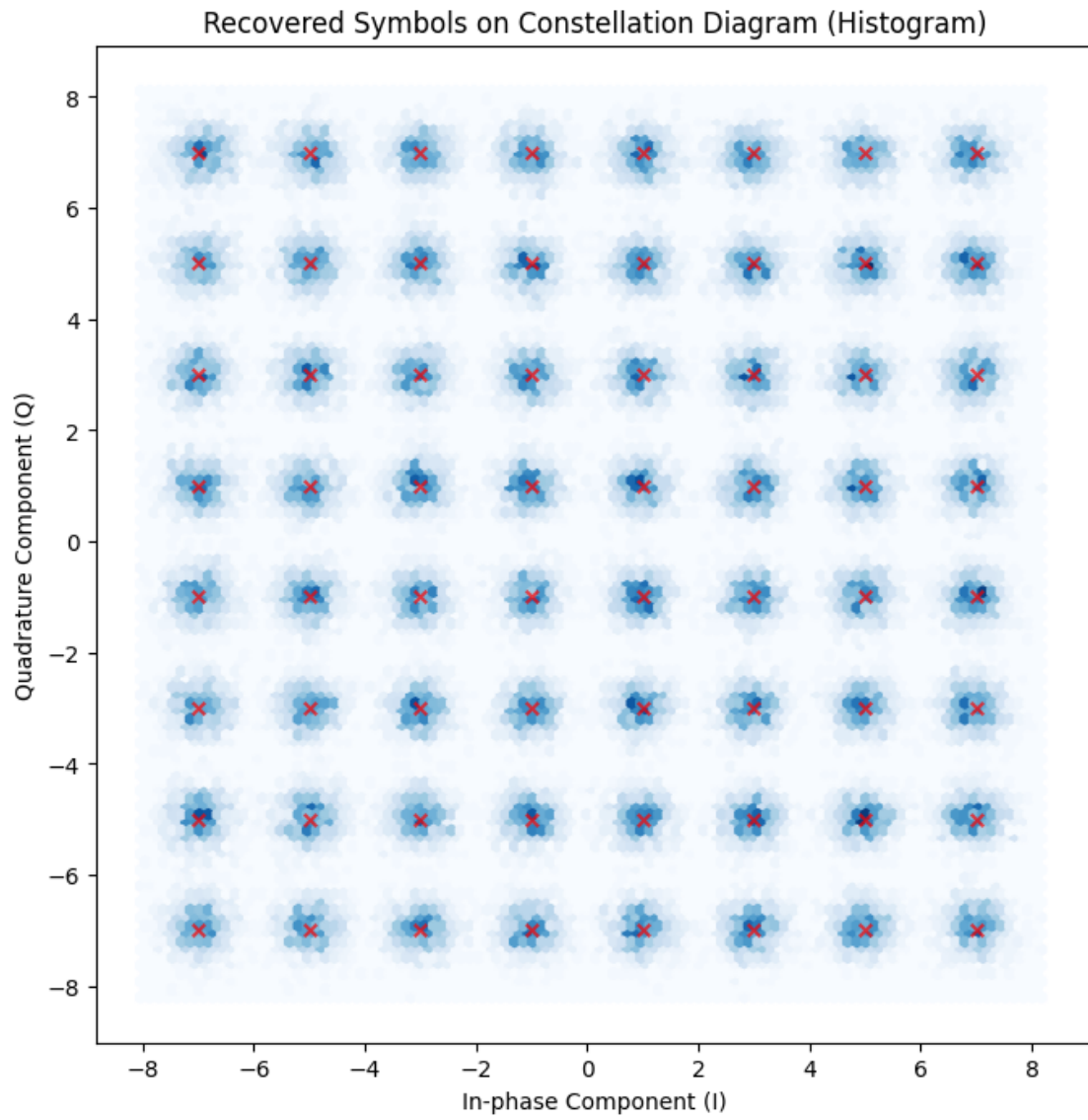
```

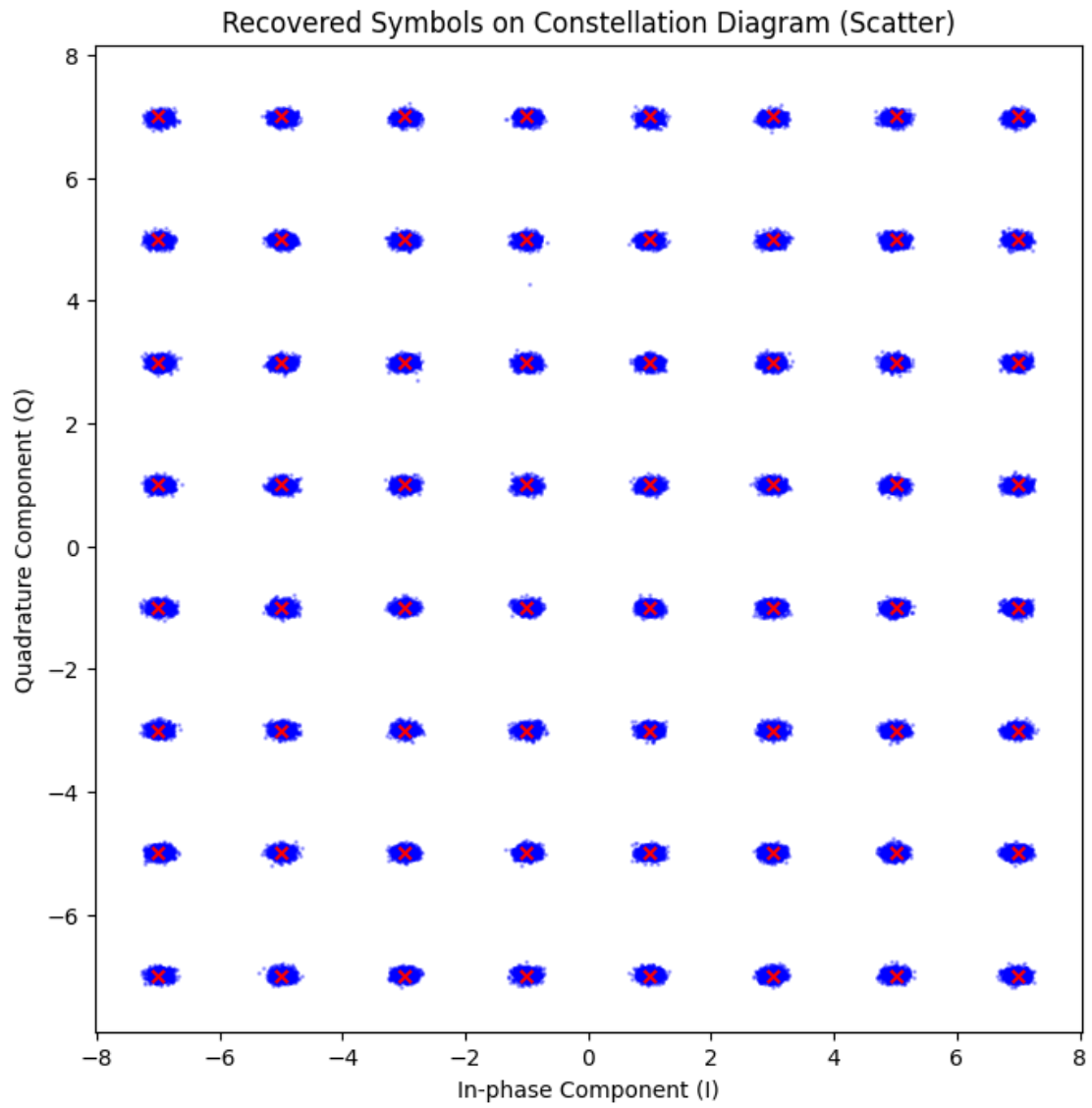
        cons
    )

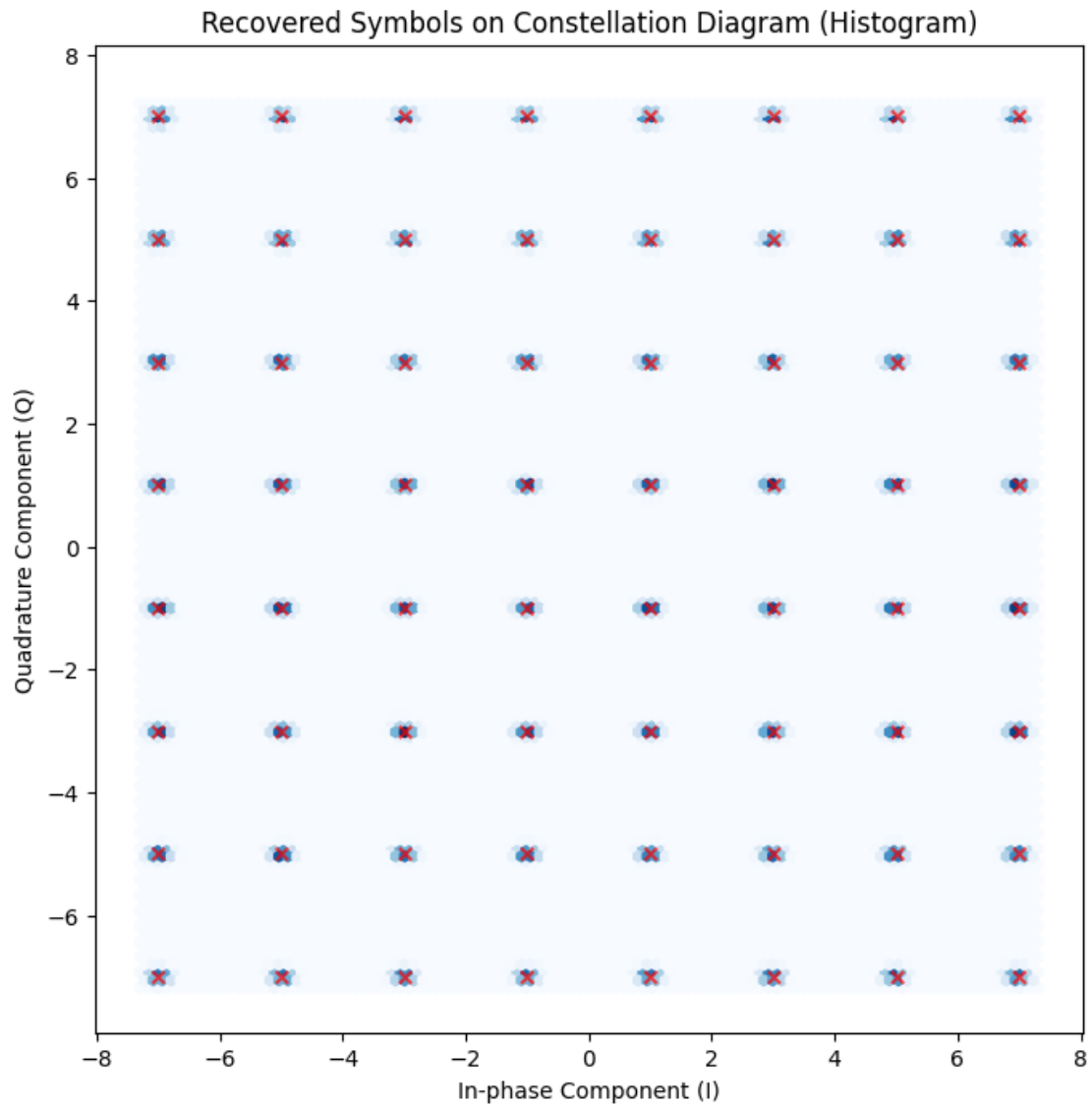
    print(demodulated_symbols[:10], temp1[:10])
    res = QAM.calculate_ber(demodulated_symbols, temp1, 35)
    print(f"BER: {res[1]}")

```









```
[46 17 27 54 0 7 42 52 8 36] [46 17 27 54 0 7 42 52 8 36]  
BER: 0.003422648151769998
```

```
[11]: 0.0060983993
```

```
[11]: 0.0060983993
```