Test & Evaluation Notebook

This notebook contains the evaluation pipeline for the YedroudjNet512 steganalysis model. I removed duplicated cells and added short explanations before each code block to clarify purpose and usage.

1) Imports and utilities

This cell imports core libraries required for dataset handling, model code, metrics and plotting.

```
import os
import random
import numpy as np
from PIL import Image
import torch
from torch import amp
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader, random_split
from torchvision import transforms
import argparse
import math
from sklearn.metrics import (
    confusion_matrix, roc_curve, auc,
    precision_recall_curve, f1_score,
    precision_score, recall_score, accuracy_score
)
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm
```

2) Configuration

Set dataset paths, training/evaluation hyperparameters, random seeds and device selection.

```
# Configuration
COVER_DIR = "c"                        # 512x512 covers
STEGO_DIR = "stego_suniward"           # 512x512 stegos
IMG_SIZE = 512
BATCH_SIZE = 8
NUM_EPOCHS = 60
LR = 1e-3
WEIGHT_DECAY = 1e-5
VAL_SPLIT = 0.15
TEST_SPLIT = 0.15
SEED = 42
```

```
PRINT_EVERY = 20       # tqdm updates
DEVICE = "cuda" if torch.cuda.is_available() else "cpu"

# Setting same seed for reproducibility
random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
if DEVICE.startswith("cuda"):
    torch.cuda.manual_seed_all(SEED)

print("Using device:", DEVICE)
print("Covers dir:", COVER_DIR)
print("Stego dir:", STEGO_DIR)
print("Image size:", IMG_SIZE, "Batch size:", BATCH_SIZE)
```

3) Model definitions (SRM + Yedroudj-Net)

Defines the SRM high-pass filters and the YedroudjNet512 architecture used for steganalysis.

```
# SRM Filter (3 kernels)
class SRMFilter(nn.Module):
    def __init__(self):
        super().__init__()
        HPF = np.zeros((3, 1, 5, 5), dtype=np.float32)

        HPF[0, 0] = np.array([
            [0, 0, 0, 0, 0],
            [0, -1, 2, -1, 0],
            [0,  2, -4,  2, 0],
            [0, -1, 2, -1, 0],
            [0, 0, 0, 0, 0],
        ], dtype=np.float32) / 4.0

        HPF[1, 0] = np.array([
            [-1, 2, -2, 2, -1],
            [2, -6, 8, -6, 2],
            [-2, 8, -12, 8, -2],
            [2, -6, 8, -6, 2],
            [-1, 2, -2, 2, -1],
        ], dtype=np.float32) / 12.0

        HPF[2, 0] = np.array([
            [0, 0, 0, 0, 0],
            [0, 1, -2, 1, 0],
            [0, -2, 4, -2, 0],
            [0, 1, -2, 1, 0],
            [0, 0, 0, 0, 0],
        ], dtype=np.float32) / 4.0
        self.register_buffer("weight", torch.from_numpy(HPF))

    def forward(self, x):
        # x: (N,1,H,W) in [0,255]
        return F.conv2d(x, self.weight, padding=2)


# Yedroudj-Net (practical variant)
def conv_block(in_c, out_c, pool='avg', drop=0.0):
    layers = []
    layers.append(nn.Conv2d(in_c, out_c, kernel_size=3, padding=1,
bias=False))
    layers.append(nn.BatchNorm2d(out_c))
    layers.append(nn.ReLU(inplace=True))
    layers.append(nn.Conv2d(out_c, out_c, kernel_size=3, padding=1,
bias=False))
    layers.append(nn.BatchNorm2d(out_c))
    layers.append(nn.ReLU(inplace=True))
    if pool == 'avg':
        layers.append(nn.AvgPool2d(2))
    elif pool == 'max':
        layers.append(nn.MaxPool2d(2))
    if drop > 0:
        layers.append(nn.Dropout(drop))
    return nn.Sequential(*layers)

class YedroudjNet512(nn.Module):
    def __init__(self, tlu=3.0):
        super().__init__()
        self.srm = SRMFilter()
        self.tlu = float(tlu)

        # Blocks tuned for 512 input
        self.block1 = conv_block(3, 32, pool='avg', drop=0.2)   # 512
-> 256
        self.block2 = conv_block(32, 64, pool='max', drop=0.25) # 256
-> 128
        self.block3 = conv_block(64, 128, pool='max', drop=0.35)# 128
-> 64
        self.block4 = conv_block(128, 256, pool='max', drop=0.45)# 64
-> 32
        self.block5 = conv_block(256, 512, pool='avg', drop=0.5) # 32
-> 16

        self.pool = nn.AdaptiveAvgPool2d((1, 1))
        self.head = nn.Sequential(
            nn.Flatten(),
            nn.Linear(512, 512),
            nn.ReLU(inplace=True),
            nn.Dropout(0.5),
            nn.Linear(512, 1)
        )

    def forward(self, x):
        # x in [0,255], shape (N,1,512,512)
        x = self.srm(x)                    # -> (N,3,512,512)
        x = torch.clamp(x, -self.tlu, self.tlu)
        x = torch.abs(x)
        x = self.block1(x)
        x = self.block2(x)
        x = self.block3(x)
        x = self.block4(x)
        x = self.block5(x)
        x = self.pool(x)
        return self.head(x)
```

4) Dataset class & transforms

Defines a paired dataset loader (matched cover/stego filenames) and the transforms used.

```
# Dataset loader (paired)
class PairedStegoDataset(Dataset):
    def __init__(self, cover_dir, stego_dir, transform=None):
        self.cover_dir = cover_dir
        self.stego_dir = stego_dir
        self.transform = transform

        cover_files = sorted(os.listdir(cover_dir))
        stego_files = sorted(os.listdir(stego_dir))
        matched = sorted(list(set(cover_files) & set(stego_files)))
        if len(matched) == 0:
            raise RuntimeError("No matched filenames found between
cover and stego directories")

        self.samples = []
        for fname in matched:
            self.samples.append((os.path.join(cover_dir, fname), 0.0))
            self.samples.append((os.path.join(stego_dir, fname), 1.0))

        print(f"Found {len(matched)} matched pairs")
        print(f"Total samples: {len(self.samples)}")

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

    def __getitem__(self, idx):
        p, label = self.samples[idx]
        img = Image.open(p).convert("L")
        if self.transform is not None:
            img = self.transform(img)
        else:
            img = transforms.ToTensor()(img)
            img = img * 255.0
        return img, torch.tensor(label, dtype=torch.float32)



# Transforms: no normalization! only ToTensor * 255
train_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Lambda(lambda t: t * 255.0)
])
eval_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Lambda(lambda t: t * 255.0)
])
```

5) Quick embedding diagnostics

Utility to inspect matched cover/stego pairs and SRM differences before running evaluation.

```
# Utility: quick embedding check (runs once)
def quick_embedding_check(cover_dir, stego_dir, n=5):
    # pick up to n matched filenames and compute pixel diffs and SRM
diffs
    files = sorted(list(set(os.listdir(cover_dir)) &
set(os.listdir(stego_dir))))
    files = files[:n]
    if len(files) == 0:
        print("No pairs to check")
        return
    srm = SRMFilter().to(DEVICE)
    srm.eval()
    print("Quick embedding diagnostics on up to", n, "pairs:")
    for fname in files:
        cpath = os.path.join(cover_dir, fname)
        spath = os.path.join(stego_dir, fname)
        c = Image.open(cpath).convert("L")
        s = Image.open(spath).convert("L")
        ca = np.array(c, dtype=np.int16)
        sa = np.array(s, dtype=np.int16)
        diff = sa - ca
        nonzero = int((diff != 0).sum())
        maxabs = int(np.max(np.abs(diff)))
        print(f" {fname}: pixel nonzero count={nonzero},
maxabs={maxabs}")
        # SRM diff
        ct = transforms.ToTensor()(c).unsqueeze(0).to(DEVICE) * 255.0
        st = transforms.ToTensor()(s).unsqueeze(0).to(DEVICE) * 255.0
        with torch.no_grad():
            rc = srm(ct)
            rs = srm(st)
            rdiff = (rs - rc)
            print(f"   SRM std cover={rc.std().item():.3f},
stego={rs.std().item():.3f}, diff_std={rdiff.std().item():.5f}")
```

6) Training helpers (metrics, training/eval loops)

Contains helper functions used by training and evaluation: accuracy computation, training epoch loop and evaluation loop.

```
# Metrics & train/eval loops (AMP + tqdm)
def accuracy_from_logits(logits, labels):
    probs = torch.sigmoid(logits)
    preds = (probs >= 0.5).float()
    return (preds == labels).float().mean().item()

def train_one_epoch(model, loader, optimizer, scaler, criterion,
epoch):
    model.train()
    running_loss = 0.0
    running_acc = 0.0
    total = 0

    pbar = tqdm(enumerate(loader), total=len(loader), desc=f"Epoch
{epoch}", ncols=120)
    for batch_idx, (imgs, labels) in pbar:
        imgs = imgs.to(DEVICE)
        labels = labels.to(DEVICE).unsqueeze(1)

        optimizer.zero_grad()
        with amp.autocast(device_type="cuda",
enabled=DEVICE.startswith("cuda")):
            logits = model(imgs)
            loss = criterion(logits, labels)

        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()

        bs = imgs.size(0)
        running_loss += loss.item() * bs
        batch_acc = accuracy_from_logits(logits, labels)
        running_acc += batch_acc * bs
        total += bs

        if (batch_idx + 1) % PRINT_EVERY == 0 or (batch_idx + 1) ==
len(loader):
            pbar.set_postfix({
                "loss": f"{running_loss/total:.4f}",
                "acc": f"{running_acc/total:.4f}",
                "lr": f"{optimizer.param_groups[0]['lr']:.6f}"
            })

    return running_loss / total, running_acc / total

def eval_model(model, loader, criterion):
    model.eval()
    running_loss = 0.0
    running_acc = 0.0
    total = 0
    with torch.no_grad():
        for imgs, labels in tqdm(loader, desc="Val", ncols=120):
            imgs = imgs.to(DEVICE)
            labels = labels.to(DEVICE).unsqueeze(1)
            logits = model(imgs)
            loss = criterion(logits, labels)
            bs = imgs.size(0)
            running_loss += loss.item() * bs
            running_acc += accuracy_from_logits(logits, labels) * bs
            total += bs
    return running_loss / total, running_acc / total
```

7) Main training script

The main training entrypoint used to train the model and save the best checkpoint.

```


# Main training script
def main():
    # quick diagnostics
    quick_embedding_check(COVER_DIR, STEGO_DIR, n=5)

    # dataset and splits
    dataset = PairedStegoDataset(COVER_DIR, STEGO_DIR, 
train_transform)
    N = len(dataset)
    n_test = int(TEST_SPLIT * N)
    n_val = int(VAL_SPLIT * N)
    n_train = N - n_val - n_test
    print("Dataset size:", N, "train/val/test:", n_train, n_val, 
n_test)

    train_ds, val_ds, test_ds = random_split(dataset, [n_train, n_val, 
n_test],
generator=torch.Generator().manual_seed(SEED))
    # ensure transforms
    train_ds.dataset.transform = train_transform
    val_ds.dataset.transform = eval_transform
    test_ds.dataset.transform = eval_transform

    use_cuda = DEVICE.startswith("cuda")
    train_loader = DataLoader(train_ds, batch_size=BATCH_SIZE,
shuffle=True, num_workers=4, pin_memory=use_cuda)
    val_loader = DataLoader(val_ds, batch_size=BATCH_SIZE,
shuffle=False, num_workers=4, pin_memory=use_cuda)
    test_loader = DataLoader(test_ds, batch_size=BATCH_SIZE,
shuffle=False, num_workers=4, pin_memory=use_cuda)

    # model, optimizer, scheduler
    model = YedroudjNet512(tlu=3.0).to(DEVICE)
    optimizer = torch.optim.Adam(model.parameters(), lr=LR,
weight_decay=WEIGHT_DECAY)
    scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer,
mode='max', factor=0.5, patience=4, min_lr=1e-6)
    criterion = nn.BCEWithLogitsLoss()
    scaler = amp.GradScaler(device="cuda", enabled=use_cuda)

    # remove existing checkpoint
    if os.path.exists("best_yedroudj_512_test.pt"):
        os.remove("best_yedroudj_512_test.pt")

    best_val = 0.0
    for epoch in range(1, NUM_EPOCHS + 1):
        train_loss, train_acc = train_one_epoch(model, train_loader,
optimizer, scaler, criterion, epoch)
        val_loss, val_acc = eval_model(model, val_loader, criterion)

        print(f"Epoch {epoch:02d}/{NUM_EPOCHS} | Train:
{train_loss:.4f}, acc {train_acc:.4f} | Val: {val_loss:.4f}, acc
{val_acc:.4f}")

        if val_acc > best_val:
            best_val = val_acc
            torch.save(model.state_dict(),
"best_yedroudj_512_test.pt")
            print(" → Saved best model")

        scheduler.step(val_acc)

if __name__ == "__main__":
    main()
```

```
Using device: cuda
Covers dir: c
Stego dir: stego_suniward
Image size: 512 Batch size: 8
Quick embedding diagnostics on up to 5 pairs:
 1.pgm: pixel nonzero count=21344, maxabs=1
   SRM std cover=5.357, stego=5.387, diff_std=0.48355
 10.pgm: pixel nonzero count=19698, maxabs=1
   SRM std cover=6.932, stego=6.948, diff_std=0.46754
 100.pgm: pixel nonzero count=19769, maxabs=1
   SRM std cover=5.161, stego=5.183, diff_std=0.46882
 1000.pgm: pixel nonzero count=21967, maxabs=1
   SRM std cover=7.444, stego=7.462, diff_std=0.49247
 10000.pgm: pixel nonzero count=20458, maxabs=1
   SRM std cover=4.406, stego=4.431, diff_std=0.47340
Found 10000 matched pairs
Total samples: 20000
Dataset size: 20000 train/val/test: 14000 3000 3000

/tmp/ipykernel_58099/956831704.py:327: FutureWarning:
`torch.cuda.amp.GradScaler(args...)` is deprecated. Please use
`torch.amp.GradScaler('cuda', args...)` instead.
  scaler = torch.cuda.amp.GradScaler(enabled=use_cuda)
Epoch 1:   0%|
| 0/1750 [00:00<?, ?it/s]/tmp/ipykernel_58099/956831704.py:254:
FutureWarning: `torch.cuda.amp.autocast(args...)` is deprecated.
Please use `torch.amp.autocast('cuda', args...)` instead.
  with torch.cuda.amp.autocast(enabled=DEVICE.startswith("cuda")):
Epoch 1: 100%|████████████████████████████████| 1750/1750
[03:12<00:00,  9.11it/s, loss=0.6908, acc=0.5335, lr=0.001000]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:20<00:00, 18.22it/s]

Epoch 01/60 | Train: 0.6908, acc 0.5335 | Val: 0.6820, acc 0.5720
 → Saved best model


Epoch 2: 100%|████████████████████████████████| 1750/1750
[03:08<00:00,  9.28it/s, loss=0.6816, acc=0.5514, lr=0.001000]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:17<00:00, 21.18it/s]

Epoch 02/60 | Train: 0.6816, acc 0.5514 | Val: 0.6777, acc 0.5443


Epoch 3: 100%|████████████████████████████████| 1750/1750
[03:08<00:00,  9.30it/s, loss=0.6894, acc=0.5132, lr=0.001000]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:20<00:00, 18.10it/s]

Epoch 03/60 | Train: 0.6894, acc 0.5132 | Val: 0.7038, acc 0.5117


Epoch 4: 100%|████████████████████████████████| 1750/1750
[03:09<00:00,  9.22it/s, loss=0.6921, acc=0.4999, lr=0.001000]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:20<00:00, 18.75it/s]

Epoch 04/60 | Train: 0.6921, acc 0.4999 | Val: 0.6932, acc 0.5000


Epoch 5: 100%|████████████████████████████████| 1750/1750
[03:11<00:00,  9.12it/s, loss=0.6917, acc=0.5128, lr=0.001000]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:20<00:00, 18.54it/s]

Epoch 05/60 | Train: 0.6917, acc 0.5128 | Val: 1.1835, acc 0.5053


Epoch 6: 100%|████████████████████████████████| 1750/1750
[03:11<00:00,  9.12it/s, loss=0.6765, acc=0.5577, lr=0.001000]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:20<00:00, 18.16it/s]

Epoch 06/60 | Train: 0.6765, acc 0.5577 | Val: 0.6400, acc 0.5830
 → Saved best model


Epoch 7: 100%|████████████████████████████████| 1750/1750
[03:08<00:00,  9.30it/s, loss=0.6533, acc=0.5799, lr=0.001000]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:20<00:00, 18.38it/s]

Epoch 07/60 | Train: 0.6533, acc 0.5799 | Val: 0.6445, acc 0.5687


Epoch 8: 100%|████████████████████████████████| 1750/1750
[03:11<00:00,  9.16it/s, loss=0.6502, acc=0.5826, lr=0.001000]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:20<00:00, 18.29it/s]

Epoch 08/60 | Train: 0.6502, acc 0.5826 | Val: 0.6066, acc 0.6163
 → Saved best model
```

```
Epoch 9: 100%|████████████████████████████████| 1750/1750
[03:07<00:00,  9.32it/s, loss=0.6399, acc=0.5855, lr=0.001000]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:17<00:00, 21.21it/s]

Epoch 09/60 | Train: 0.6399, acc 0.5855 | Val: 0.6401, acc 0.6100


Epoch 10: 100%|███████████████████████████████| 1750/1750
[03:10<00:00,  9.19it/s, loss=0.6305, acc=0.5960, lr=0.001000]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:20<00:00, 18.63it/s]

Epoch 10/60 | Train: 0.6305, acc 0.5960 | Val: 0.5939, acc 0.6197
 → Saved best model


Epoch 11: 100%|███████████████████████████████| 1750/1750
[03:11<00:00,  9.12it/s, loss=0.6278, acc=0.6107, lr=0.001000]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:20<00:00, 18.21it/s]

Epoch 11/60 | Train: 0.6278, acc 0.6107 | Val: 0.6360, acc 0.5707


Epoch 12: 100%|███████████████████████████████| 1750/1750
[03:07<00:00,  9.33it/s, loss=0.6179, acc=0.6109, lr=0.001000]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:20<00:00, 18.59it/s]

Epoch 12/60 | Train: 0.6179, acc 0.6109 | Val: 0.5878, acc 0.6167


Epoch 13: 100%|███████████████████████████████| 1750/1750
[03:08<00:00,  9.26it/s, loss=0.6100, acc=0.6216, lr=0.001000]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:20<00:00, 18.74it/s]

Epoch 13/60 | Train: 0.6100, acc 0.6216 | Val: 0.5818, acc 0.6407
 → Saved best model


Epoch 14: 100%|███████████████████████████████| 1750/1750
[03:08<00:00,  9.29it/s, loss=0.6048, acc=0.6288, lr=0.001000]
Val: 100%|
███████████████████████████████████████████████████████████████████████
██████| 375/375 [00:18<00:00, 19.77it/s]

Epoch 14/60 | Train: 0.6048, acc 0.6288 | Val: 0.5585, acc 0.6690
 → Saved best model


Epoch 15: 100%|███████████████████████████████| 1750/1750
[03:07<00:00,  9.35it/s, loss=0.6017, acc=0.6295, lr=0.001000]
Val: 100%|
███████████████████████████████████████████████████████████████████████
██████| 375/375 [00:20<00:00, 18.68it/s]

Epoch 15/60 | Train: 0.6017, acc 0.6295 | Val: 0.5993, acc 0.6410


Epoch 16: 100%|███████████████████████████████| 1750/1750
[03:09<00:00,  9.24it/s, loss=0.6036, acc=0.6335, lr=0.001000]
Val: 100%|
███████████████████████████████████████████████████████████████████████
██████| 375/375 [00:20<00:00, 18.40it/s]

Epoch 16/60 | Train: 0.6036, acc 0.6335 | Val: 0.5760, acc 0.6200


Epoch 17: 100%|███████████████████████████████| 1750/1750
[03:07<00:00,  9.33it/s, loss=0.6012, acc=0.6378, lr=0.001000]
Val: 100%|
███████████████████████████████████████████████████████████████████████
██████| 375/375 [00:20<00:00, 18.29it/s]

Epoch 17/60 | Train: 0.6012, acc 0.6378 | Val: 0.8657, acc 0.5357


Epoch 18: 100%|███████████████████████████████| 1750/1750
[03:10<00:00,  9.18it/s, loss=0.5938, acc=0.6408, lr=0.001000]
Val: 100%|
███████████████████████████████████████████████████████████████████████
██████| 375/375 [00:20<00:00, 18.52it/s]

Epoch 18/60 | Train: 0.5938, acc 0.6408 | Val: 0.5796, acc 0.6293


Epoch 19: 100%|███████████████████████████████| 1750/1750
[03:09<00:00,  9.24it/s, loss=0.5921, acc=0.6461, lr=0.001000]
Val: 100%|
███████████████████████████████████████████████████████████████████████
██████| 375/375 [00:20<00:00, 18.60it/s]

Epoch 19/60 | Train: 0.5921, acc 0.6461 | Val: 0.6199, acc 0.5877
Epoch 20: 100%|███████████████████████████████| 1750/1750
[03:09<00:00,  9.23it/s, loss=0.5765, acc=0.6482, lr=0.000500]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:20<00:00, 18.44it/s]

Epoch 20/60 | Train: 0.5765, acc 0.6482 | Val: 0.5389, acc 0.6700
 → Saved best model


Epoch 21: 100%|███████████████████████████████| 1750/1750
[03:12<00:00,  9.09it/s, loss=0.5700, acc=0.6612, lr=0.000500]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:20<00:00, 18.38it/s]

Epoch 21/60 | Train: 0.5700, acc 0.6612 | Val: 0.5330, acc 0.6973
 → Saved best model


Epoch 22: 100%|███████████████████████████████| 1750/1750
[03:04<00:00,  9.49it/s, loss=0.5681, acc=0.6645, lr=0.000500]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:19<00:00, 19.42it/s]

Epoch 22/60 | Train: 0.5681, acc 0.6645 | Val: 0.5824, acc 0.6313


Epoch 23: 100%|███████████████████████████████| 1750/1750
[03:04<00:00,  9.48it/s, loss=0.5702, acc=0.6619, lr=0.000500]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:19<00:00, 19.59it/s]

Epoch 23/60 | Train: 0.5702, acc 0.6619 | Val: 0.6759, acc 0.5733


Epoch 24: 100%|███████████████████████████████| 1750/1750
[03:02<00:00,  9.57it/s, loss=0.5712, acc=0.6647, lr=0.000500]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:18<00:00, 19.87it/s]

Epoch 24/60 | Train: 0.5712, acc 0.6647 | Val: 0.5479, acc 0.6713


Epoch 25: 100%|███████████████████████████████| 1750/1750
[03:10<00:00,  9.17it/s, loss=0.5667, acc=0.6706, lr=0.000500]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:20<00:00, 18.22it/s]

Epoch 25/60 | Train: 0.5667, acc 0.6706 | Val: 0.5300, acc 0.6807


Epoch 26: 100%|███████████████████████████████| 1750/1750
[03:09<00:00,  9.22it/s, loss=0.5642, acc=0.6675, lr=0.000500]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:20<00:00, 18.22it/s]

Epoch 26/60 | Train: 0.5642, acc 0.6675 | Val: 0.5576, acc 0.6613


Epoch 27: 100%|███████████████████████████████| 1750/1750
[03:09<00:00,  9.24it/s, loss=0.5528, acc=0.6749, lr=0.000250]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:20<00:00, 18.73it/s]

Epoch 27/60 | Train: 0.5528, acc 0.6749 | Val: 0.5303, acc 0.6843


Epoch 28: 100%|███████████████████████████████| 1750/1750
[03:08<00:00,  9.31it/s, loss=0.5521, acc=0.6744, lr=0.000250]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:20<00:00, 18.69it/s]

Epoch 28/60 | Train: 0.5521, acc 0.6744 | Val: 0.5410, acc 0.6790


Epoch 29: 100%|███████████████████████████████| 1750/1750
[03:09<00:00,  9.22it/s, loss=0.5456, acc=0.6809, lr=0.000250]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:20<00:00, 18.26it/s]

Epoch 29/60 | Train: 0.5456, acc 0.6809 | Val: 0.5274, acc 0.6920


Epoch 30: 100%|███████████████████████████████| 1750/1750
[03:11<00:00,  9.16it/s, loss=0.5463, acc=0.6809, lr=0.000250]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:20<00:00, 18.46it/s]

Epoch 30/60 | Train: 0.5463, acc 0.6809 | Val: 0.5219, acc 0.7057
 → Saved best model
```

```
Epoch 31: 100%|███████████████████████████████| 1750/1750
[02:59<00:00,  9.74it/s, loss=0.5428, acc=0.6835, lr=0.000250]
Val: 100%|
████████████████████████████████████████████████████████████████████████
██████| 375/375 [00:18<00:00, 19.92it/s]

Epoch 31/60 | Train: 0.5428, acc 0.6835 | Val: 0.5544, acc 0.6690


Epoch 32: 100%|███████████████████████████████| 1750/1750
[03:04<00:00,  9.47it/s, loss=0.5412, acc=0.6872, lr=0.000250]
Val: 100%|
████████████████████████████████████████████████████████████████████████
██████| 375/375 [00:20<00:00, 18.20it/s]

Epoch 32/60 | Train: 0.5412, acc 0.6872 | Val: 0.5335, acc 0.6837


Epoch 33: 100%|███████████████████████████████| 1750/1750
[03:08<00:00,  9.27it/s, loss=0.5409, acc=0.6879, lr=0.000250]
Val: 100%|
████████████████████████████████████████████████████████████████████████
██████| 375/375 [00:19<00:00, 18.85it/s]

Epoch 33/60 | Train: 0.5409, acc 0.6879 | Val: 0.5115, acc 0.7133
 → Saved best model


Epoch 34: 100%|███████████████████████████████| 1750/1750
[03:01<00:00,  9.63it/s, loss=0.5387, acc=0.6887, lr=0.000250]
Val: 100%|
████████████████████████████████████████████████████████████████████████
██████| 375/375 [00:19<00:00, 19.59it/s]

Epoch 34/60 | Train: 0.5387, acc 0.6887 | Val: 0.5133, acc 0.7103


Epoch 35: 100%|███████████████████████████████| 1750/1750
[03:09<00:00,  9.21it/s, loss=0.5373, acc=0.6856, lr=0.000250]
Val: 100%|
████████████████████████████████████████████████████████████████████████
██████| 375/375 [00:20<00:00, 18.21it/s]

Epoch 35/60 | Train: 0.5373, acc 0.6856 | Val: 0.5296, acc 0.6940


Epoch 36: 100%|███████████████████████████████| 1750/1750
[03:10<00:00,  9.17it/s, loss=0.5367, acc=0.6916, lr=0.000250]
Val: 100%|
████████████████████████████████████████████████████████████████████████
██████| 375/375 [00:19<00:00, 18.80it/s]
```

```
Epoch 36/60 | Train: 0.5367, acc 0.6916 | Val: 0.5961, acc 0.6470


Epoch 37: 100%|███████████████████████████████| 1750/1750
[03:10<00:00,  9.19it/s, loss=0.5390, acc=0.6852, lr=0.000250]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:20<00:00, 18.22it/s]

Epoch 37/60 | Train: 0.5390, acc 0.6852 | Val: 0.5126, acc 0.7177
 → Saved best model


Epoch 38: 100%|███████████████████████████████| 1750/1750
[03:10<00:00,  9.17it/s, loss=0.5344, acc=0.6882, lr=0.000250]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:20<00:00, 18.72it/s]

Epoch 38/60 | Train: 0.5344, acc 0.6882 | Val: 0.5229, acc 0.7060


Epoch 39: 100%|███████████████████████████████| 1750/1750
[03:04<00:00,  9.47it/s, loss=0.5377, acc=0.6902, lr=0.000250]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:18<00:00, 19.92it/s]

Epoch 39/60 | Train: 0.5377, acc 0.6902 | Val: 0.5267, acc 0.6857


Epoch 40: 100%|███████████████████████████████| 1750/1750
[02:58<00:00,  9.78it/s, loss=0.5322, acc=0.6907, lr=0.000250]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:18<00:00, 19.93it/s]

Epoch 40/60 | Train: 0.5322, acc 0.6907 | Val: 0.5167, acc 0.6967


Epoch 41: 100%|███████████████████████████████| 1750/1750
[02:48<00:00, 10.38it/s, loss=0.5305, acc=0.6956, lr=0.000250]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:17<00:00, 21.24it/s]

Epoch 41/60 | Train: 0.5305, acc 0.6956 | Val: 0.5096, acc 0.7123


Epoch 42: 100%|███████████████████████████████| 1750/1750
[02:46<00:00, 10.51it/s, loss=0.5304, acc=0.6969, lr=0.000250]
Val: 100%|
████████████████████████████████████████████████████████
█████| 375/375 [00:17<00:00, 21.24it/s]

Epoch 42/60 | Train: 0.5304, acc 0.6969 | Val: 0.5160, acc 0.7120


Epoch 43: 100%|█████████████████████████| 1750/1750
[02:46<00:00, 10.51it/s, loss=0.5250, acc=0.6949, lr=0.000125]
Val: 100%|
████████████████████████████████████████████████████████
█████| 375/375 [00:17<00:00, 21.24it/s]

Epoch 43/60 | Train: 0.5250, acc 0.6949 | Val: 0.5208, acc 0.6953


Epoch 44: 100%|█████████████████████████| 1750/1750
[02:46<00:00, 10.51it/s, loss=0.5239, acc=0.6968, lr=0.000125]
Val: 100%|
████████████████████████████████████████████████████████
█████| 375/375 [00:17<00:00, 21.24it/s]

Epoch 44/60 | Train: 0.5239, acc 0.6968 | Val: 0.5392, acc 0.6887


Epoch 45: 100%|█████████████████████████| 1750/1750
[02:46<00:00, 10.51it/s, loss=0.5227, acc=0.7016, lr=0.000125]
Val: 100%|
████████████████████████████████████████████████████████
█████| 375/375 [00:17<00:00, 21.22it/s]

Epoch 45/60 | Train: 0.5227, acc 0.7016 | Val: 0.6099, acc 0.6600


Epoch 46: 100%|█████████████████████████| 1750/1750
[02:46<00:00, 10.51it/s, loss=0.5231, acc=0.6988, lr=0.000125]
Val: 100%|
████████████████████████████████████████████████████████
█████| 375/375 [00:17<00:00, 21.24it/s]

Epoch 46/60 | Train: 0.5231, acc 0.6988 | Val: 0.5988, acc 0.6777


Epoch 47: 100%|█████████████████████████| 1750/1750
[02:46<00:00, 10.51it/s, loss=0.5223, acc=0.7026, lr=0.000125]
Val: 100%|
████████████████████████████████████████████████████████
█████| 375/375 [00:17<00:00, 21.21it/s]

Epoch 47/60 | Train: 0.5223, acc 0.7026 | Val: 0.4988, acc 0.7263
 → Saved best model
```

```
Epoch 48: 100%|███████████████████████████████| 1750/1750
[02:46<00:00, 10.51it/s, loss=0.5195, acc=0.7030, lr=0.000125]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:17<00:00, 21.24it/s]

Epoch 48/60 | Train: 0.5195, acc 0.7030 | Val: 0.6048, acc 0.6637


Epoch 49: 100%|███████████████████████████████| 1750/1750
[02:46<00:00, 10.51it/s, loss=0.5213, acc=0.7046, lr=0.000125]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:17<00:00, 21.24it/s]

Epoch 49/60 | Train: 0.5213, acc 0.7046 | Val: 0.5106, acc 0.7113


Epoch 50: 100%|███████████████████████████████| 1750/1750
[02:46<00:00, 10.51it/s, loss=0.5182, acc=0.7028, lr=0.000125]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:17<00:00, 21.25it/s]

Epoch 50/60 | Train: 0.5182, acc 0.7028 | Val: 0.4939, acc 0.7207


Epoch 51: 100%|███████████████████████████████| 1750/1750
[02:46<00:00, 10.51it/s, loss=0.5185, acc=0.7024, lr=0.000125]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:17<00:00, 21.23it/s]

Epoch 51/60 | Train: 0.5185, acc 0.7024 | Val: 0.4955, acc 0.7270
 → Saved best model

Epoch 52: 100%|███████████████████████████████| 1750/1750
[02:46<00:00, 10.51it/s, loss=0.5182, acc=0.7011, lr=0.000125]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:17<00:00, 21.20it/s]

Epoch 52/60 | Train: 0.5182, acc 0.7011 | Val: 0.5042, acc 0.7250


Epoch 53: 100%|███████████████████████████████| 1750/1750
[02:46<00:00, 10.51it/s, loss=0.5176, acc=0.7034, lr=0.000125]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:17<00:00, 21.21it/s]
```

```
Epoch 53/60 | Train: 0.5176, acc 0.7034 | Val: 0.4977, acc 0.7250


Epoch 54: 100%|███████████████████████████████| 1750/1750
[02:46<00:00, 10.51it/s, loss=0.5161, acc=0.7003, lr=0.000125]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:17<00:00, 21.20it/s]

Epoch 54/60 | Train: 0.5161, acc 0.7003 | Val: 0.5100, acc 0.7153


Epoch 55: 100%|███████████████████████████████| 1750/1750
[02:46<00:00, 10.51it/s, loss=0.5122, acc=0.7088, lr=0.000125]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:17<00:00, 21.21it/s]

Epoch 55/60 | Train: 0.5122, acc 0.7088 | Val: 0.4995, acc 0.7203


Epoch 56: 100%|███████████████████████████████| 1750/1750
[02:46<00:00, 10.50it/s, loss=0.5144, acc=0.7000, lr=0.000125]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:17<00:00, 21.21it/s]

Epoch 56/60 | Train: 0.5144, acc 0.7000 | Val: 0.5662, acc 0.6750

Epoch 57: 100%|███████████████████████████████| 1750/1750
[02:46<00:00, 10.51it/s, loss=0.5093, acc=0.7104, lr=0.000063]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:17<00:00, 21.22it/s]

Epoch 57/60 | Train: 0.5093, acc 0.7104 | Val: 0.5191, acc 0.7007


Epoch 58: 100%|███████████████████████████████| 1750/1750
[02:46<00:00, 10.50it/s, loss=0.5119, acc=0.7066, lr=0.000063]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:17<00:00, 21.18it/s]

Epoch 58/60 | Train: 0.5119, acc 0.7066 | Val: 0.4945, acc 0.7227


Epoch 59: 100%|███████████████████████████████| 1750/1750
[02:46<00:00, 10.51it/s, loss=0.5133, acc=0.7024, lr=0.000063]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:17<00:00, 21.19it/s]
```

```
Epoch 59/60 | Train: 0.5133, acc 0.7024 | Val: 0.4825, acc 0.7367
 → Saved best model


Epoch 60: 100%|███████████████████████████████| 1750/1750
[02:46<00:00, 10.50it/s, loss=0.5125, acc=0.7112, lr=0.000063]
Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:17<00:00, 21.22it/s]
/tmp/ipykernel_58099/956831704.py:349: 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.
  model.load_state_dict(torch.load("best_yedroudj_512.pt",
map_location=DEVICE))

Epoch 60/60 | Train: 0.5125, acc 0.7112 | Val: 0.5022, acc 0.7133
Loading best model for final test...

Val: 100%|
██████████████████████████████████████████████████████████████████████
██████| 375/375 [00:17<00:00, 21.20it/s]


FINAL TEST ACCURACY: 0.7287

---------------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
last)
Cell In[1], line 359
    356     print(f"Example ({example}) stego probability:
{prob:.4f}")
    358 if __name__ == "__main__":
--> 359     main()

Cell In[1], line 355, in main()
    353 # example inference
    354 example = os.listdir(COVER_DIR)[0]
--> 355 prob = predict_stego_probability(model,
os.path.join(COVER_DIR, example))
    356 print(f"Example ({example}) stego probability: {prob:.4f}")

NameError: name 'predict_stego_probability' is not defined
```

8) Prepare test loader for evaluation

When running evaluation (below) we need a `test_loader`. This small cell rebuilds the paired dataset and creates a test loader using the already-defined `PairedStegoDataset` to avoid duplicating dataset classes.

```
# Build dataset and test loader for evaluation (non-duplicate,
concise)
dataset = PairedStegoDataset(COVER_DIR, STEGO_DIR, eval_transform)
N = len(dataset)
n_test = int(TEST_SPLIT * N)
n_val = int(VAL_SPLIT * N)
n_train = N - n_val - n_test
from torch.utils.data import random_split
train_ds, val_ds, test_ds = random_split(dataset, [n_train, n_val,
n_test], generator=torch.Generator().manual_seed(SEED))
test_loader = DataLoader(test_ds, batch_size=BATCH_SIZE,
shuffle=False)
print("Test loader ready.")
```

9) Load model checkpoint

Load the best checkpoint saved during training and prepare the model for evaluation.

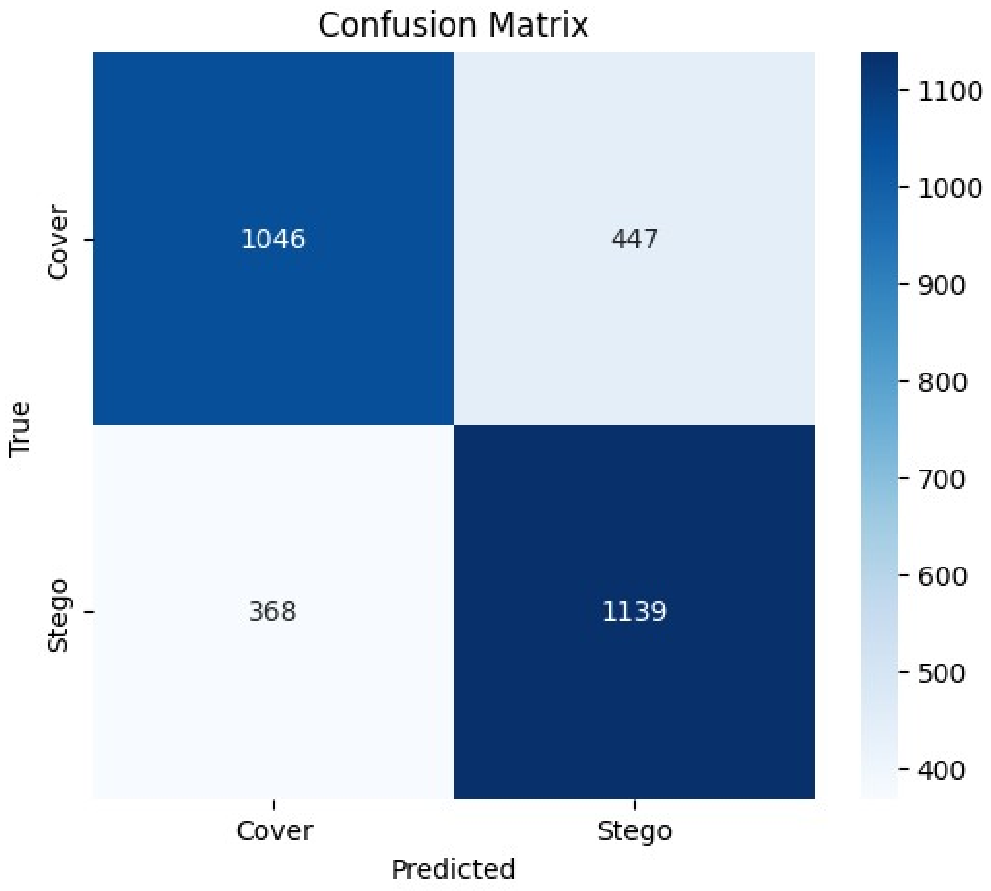
```
model = YedroudjNet512().to(DEVICE)

state = torch.load(
    "best_yedroudj_512.pt",
    map_location=DEVICE,
    weights_only=True
)

model.load_state_dict(state)
model.eval()

print("Model loaded successfully.")

Model loaded successfully.
```

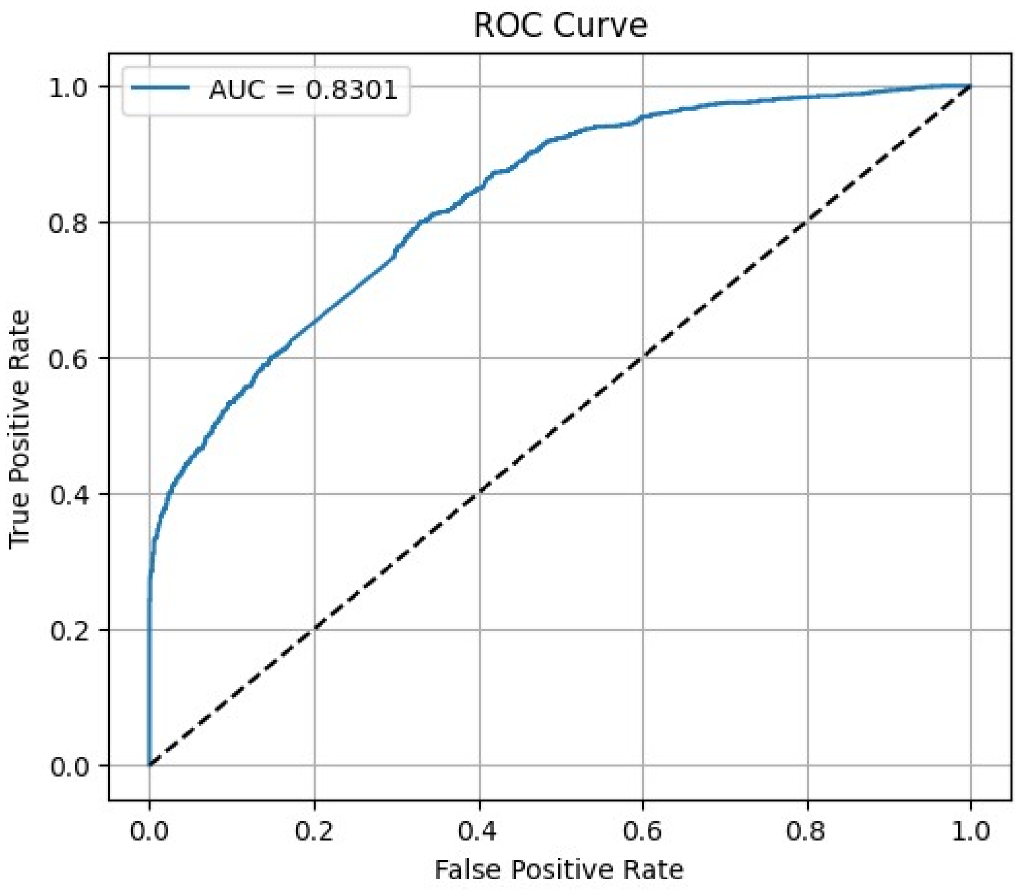
10) Evaluation & Metrics

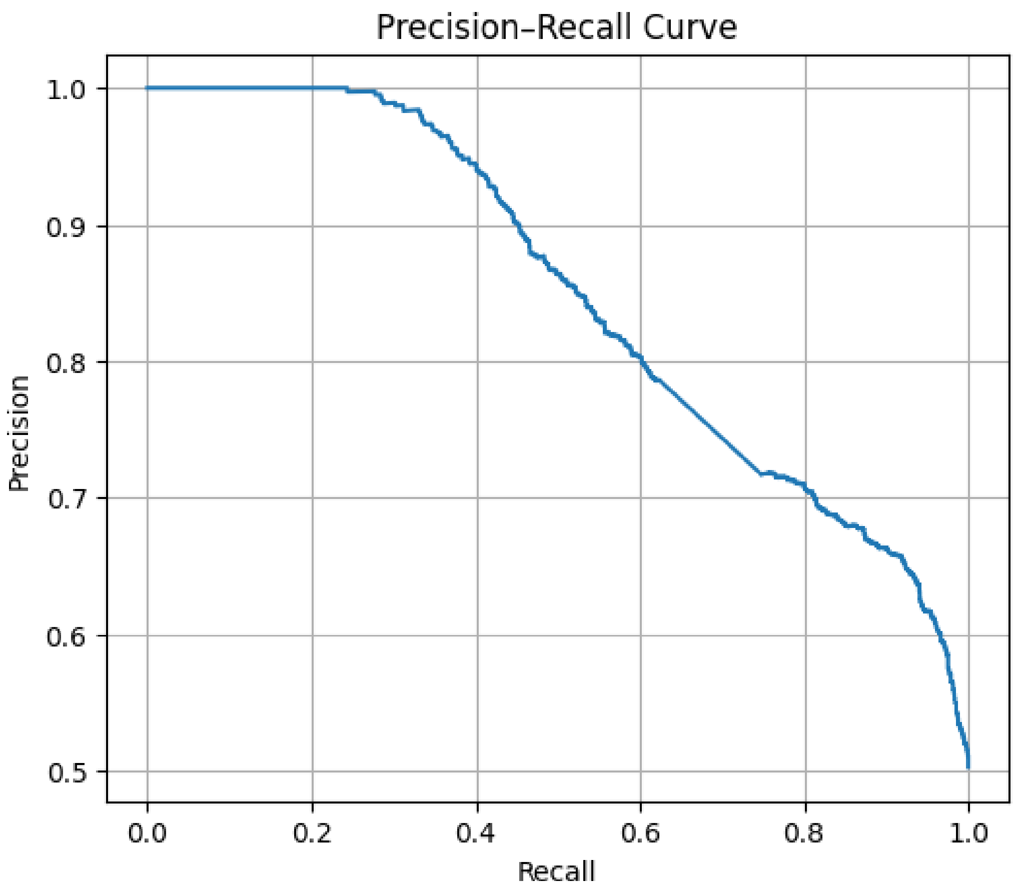
Run the model on the `test_loader`, collect predictions and compute common evaluation metrics and plots.

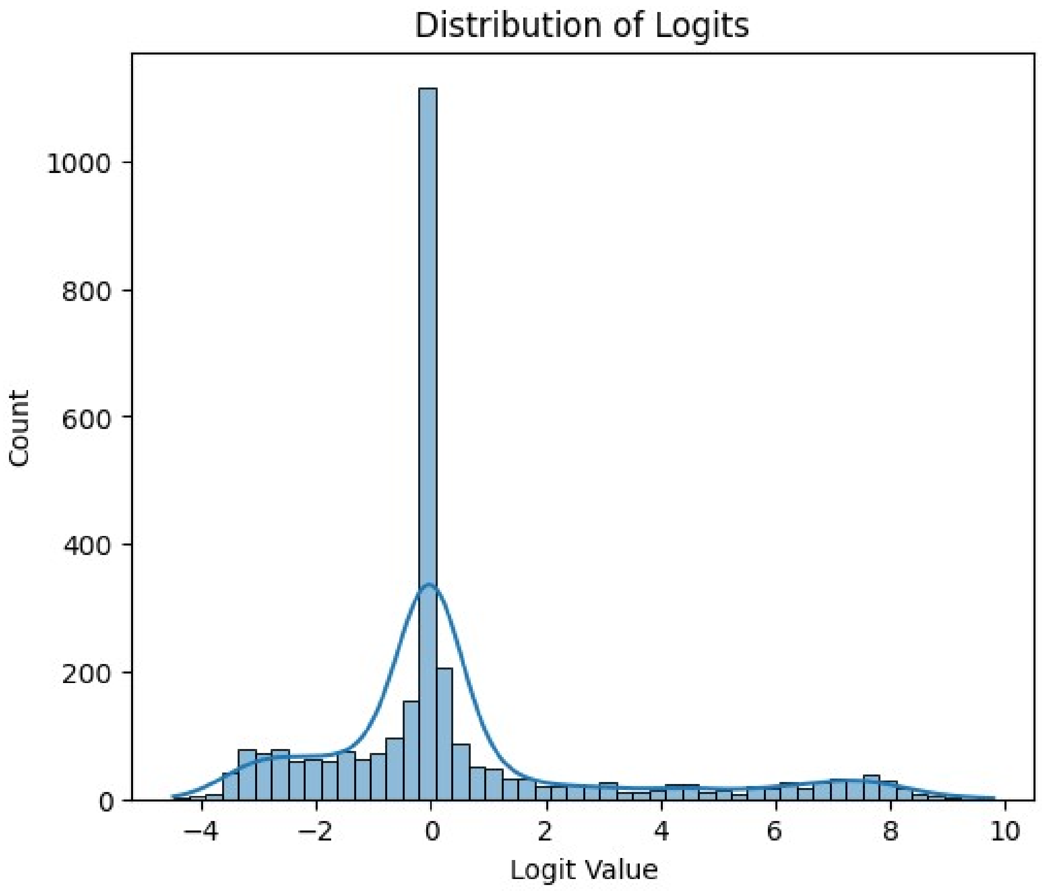
```python
# 1. Collect predictions on test set
model.eval()
all_labels = []
all_probs = []
all_logits = []

with torch.no_grad():
    for imgs, labels in test_loader:
        imgs = imgs.to(DEVICE)
        labels = labels.to(DEVICE)

        logits = model(imgs).squeeze(1)
        probs = torch.sigmoid(logits)

        all_labels.extend(labels.cpu().numpy())
        all_probs.extend(probs.cpu().numpy())
        all_logits.extend(logits.cpu().numpy())

all_labels = np.array(all_labels)
all_probs = np.array(all_probs)
all_logits = np.array(all_logits)

preds = (all_probs >= 0.5).astype(int)


# 2. Compute core metrics
acc = accuracy_score(all_labels, preds)
prec = precision_score(all_labels, preds)
rec = recall_score(all_labels, preds)
f1 = f1_score(all_labels, preds)

print("=====================================")
print("FINAL EVALUATION METRICS")
print("=====================================")
print(f"Accuracy  : {acc:.4f}")
print(f"Precision : {prec:.4f}")
print(f"Recall    : {rec:.4f}")
print(f"F1 Score  : {f1:.4f}")

# 3. Confusion Matrix
cm = confusion_matrix(all_labels, preds)

plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=["Cover","Stego"],
            yticklabels=["Cover","Stego"])
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```

```
# 4. ROC Curve
fpr, tpr, _ = roc_curve(all_labels, all_probs)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(6,5))
plt.plot(fpr, tpr, label=f"AUC = {roc_auc:.4f}")
plt.plot([0,1],[0,1],'k--')
plt.title("ROC Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.grid(True)
plt.show()

# 5. Precision–Recall Curve
prec_curve, rec_curve, _ = precision_recall_curve(all_labels,
all_probs)

plt.figure(figsize=(6,5))
plt.plot(rec_curve, prec_curve)
plt.title("Precision–Recall Curve")
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.grid(True)
plt.show()

# 6. Distribution of logits
plt.figure(figsize=(6,5))
sns.histplot(all_logits, bins=50, kde=True)
plt.title("Distribution of Logits")
plt.xlabel("Logit Value")
plt.show()

=====================================
FINAL EVALUATION METRICS
=====================================
Accuracy  : 0.7283
Precision : 0.7182
Recall    : 0.7558
F1 Score  : 0.7365
```


Confusion Matrix
True
Cover
Stego
Predicted
Cover
Stego
1046
447
368
1139
1100
1000
900
800
700
600
500
400


ROC Curve
AUC = 0.8301
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0


Precision–Recall Curve
Precision
Recall
1.0
0.9
0.8
0.7
0.6
0.5
0.0
0.2
0.4
0.6
0.8
1.0


Distribution of Logits
Count
0
200
400
600
800
1000
−4
−2
0
2
4
6
8
10
Logit Value