In [9]:

```python
import pandas as pd
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader, random_split
from transformers import AutoTokenizer, AutoModel
from tqdm import tqdm
from sklearn.metrics import roc_auc_score, f1_score, confusion_matrix, classificati
import gc

class ICUTextDataset(Dataset):
    def __init__(self, csv_path, tokenizer_name, max_length, mode='text_only'):
        self.data = pd.read_csv(csv_path).reset_index(drop=True)
        self.tokenizer = AutoTokenizer.from_pretrained(tokenizer_name)
        self.max_length = max_length
        self.mode = mode

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

    def __getitem__(self, idx):
        row = self.data.iloc[idx]
        text = str(row['text_note'])
        combined = str(row['combined_note'])

        if self.mode == 'text_only':
            full_text = text
        elif self.mode == 'combined_only':
            full_text = combined
        else:
            full_text = text + ' ' + combined

        encoded = self.tokenizer(full_text, max_length=self.max_length, padding='ma

        return {
            'input_ids': encoded['input_ids'].squeeze(0),
            'attention_mask': encoded['attention_mask'].squeeze(0),
            'label': torch.tensor(row['mortality_label'], dtype=torch.float32)
        }

class BERTClassifier(nn.Module):
    def __init__(self, encoder_name, hidden_dim=768):
        super().__init__()
        self.encoder = AutoModel.from_pretrained(encoder_name)
        self.classifier = nn.Sequential(
            nn.Linear(hidden_dim, 128),
            nn.ReLU(),
            nn.Linear(128, 1)
        )

    def forward(self, input_ids, attention_mask):
        output = self.encoder(input_ids=input_ids, attention_mask=attention_mask)
        cls_embedding = output.last_hidden_state[:, 0, :]
        return self.classifier(cls_embedding)
```

```python
def train(model, dataloader, device, pos_weight):
    model.train()
    optimizer = torch.optim.AdamW(model.parameters(), lr=1e-4)
    loss_fn = nn.BCEWithLogitsLoss(pos_weight=pos_weight)

    pbar = tqdm(dataloader, desc='Training', leave=False)
    for batch in pbar:
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['label'].to(device).unsqueeze(1)

        logits = model(input_ids, attention_mask)
        loss = loss_fn(logits, labels)

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

        pbar.set_postfix({"loss": loss.item()})

def evaluate(model, dataloader, device):
    model.eval()
    preds, labels = [], []

    with torch.no_grad():
        for batch in tqdm(dataloader, desc='Evaluating', leave=False):
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            label = batch['label'].to(device)

            logits = model(input_ids, attention_mask).squeeze()
            probs = torch.sigmoid(logits)
            preds.extend(probs.cpu().numpy())
            labels.extend(label.cpu().numpy())

    bin_preds = [1 if p > 0.5 else 0 for p in preds]
    print(f"AUC: {roc_auc_score(labels, preds):.4f}, F1: {f1_score(labels, bin_pred
    print(confusion_matrix(labels, bin_preds))
    print(classification_report(labels, bin_preds))

def run_experiment(model_name, max_length, mode_label):
    print(f"\n=== Running mode: {mode_label} ===")
    dataset = ICUTextDataset("final.csv", tokenizer_name=model_name, max_length=max
    train_size = int(0.5 * len(dataset))
    val_size = len(dataset) - train_size
    train_data, val_data = random_split(dataset, [train_size, val_size])

    train_loader = DataLoader(train_data, batch_size=4, shuffle=True)
    val_loader = DataLoader(val_data, batch_size=4, shuffle=False)

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model = BERTClassifier(encoder_name=model_name).to(device)

    labels_all = [dataset[i]['label'].item() for i in train_data.indices]
    pos_weight_val = torch.tensor([(len(labels_all) - sum(labels_all)) / sum(labels
```

```python
    for epoch in range(10):
        print(f"Epoch {epoch+1}/10")
        train(model, train_loader, device, pos_weight_val)
        evaluate(model, val_loader, device)

    del model
    torch.cuda.empty_cache()
    gc.collect()
```

In [10]:
```python
base_model = "bert-base-uncased"
run_experiment(base_model, max_length=512, mode_label='text_only')
```

```
=== Running mode: text_only ===
Epoch 1/10
/home/cbb575_rw747/.conda/envs/cbb575/lib/python3.13/site-packages/sklearn/metrics/_
classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being s
et to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/cbb575_rw747/.conda/envs/cbb575/lib/python3.13/site-packages/sklearn/metrics/_
classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being s
et to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/cbb575_rw747/.conda/envs/cbb575/lib/python3.13/site-packages/sklearn/metrics/_
classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being s
et to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
AUC: 0.4860, F1: 0.1470
[[  0 534]
 [  0  46]]
              precision    recall  f1-score   support

         0.0       0.00      0.00      0.00       534
         1.0       0.08      1.00      0.15        46

    accuracy                           0.08       580
   macro avg       0.04      0.50      0.07       580
weighted avg       0.01      0.08      0.01       580

Epoch 2/10
```

```
/home/cbb575_rw747/.conda/envs/cbb575/lib/python3.13/site-packages/sklearn/metrics/_
classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being s
et to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/cbb575_rw747/.conda/envs/cbb575/lib/python3.13/site-packages/sklearn/metrics/_
classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being s
et to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/cbb575_rw747/.conda/envs/cbb575/lib/python3.13/site-packages/sklearn/metrics/_
classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being s
et to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
AUC: 0.4922, F1: 0.0000
[[534   0]
 [ 46   0]]
              precision    recall  f1-score   support

         0.0       0.92      1.00      0.96       534
         1.0       0.00      0.00      0.00        46

    accuracy                           0.92       580
   macro avg       0.46      0.50      0.48       580
weighted avg       0.85      0.92      0.88       580

Epoch 3/10
/home/cbb575_rw747/.conda/envs/cbb575/lib/python3.13/site-packages/sklearn/metrics/_
classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being s
et to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/cbb575_rw747/.conda/envs/cbb575/lib/python3.13/site-packages/sklearn/metrics/_
classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being s
et to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/cbb575_rw747/.conda/envs/cbb575/lib/python3.13/site-packages/sklearn/metrics/_
classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being s
et to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
AUC: 0.5000, F1: 0.1470
[[  0 534]
 [  0  46]]
              precision    recall  f1-score   support

         0.0       0.00      0.00      0.00       534
         1.0       0.08      1.00      0.15        46

    accuracy                           0.08       580
   macro avg       0.04      0.50      0.07       580
weighted avg       0.01      0.08      0.01       580

Epoch 4/10
/home/cbb575_rw747/.conda/envs/cbb575/lib/python3.13/site-packages/sklearn/metrics/_
classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being s
et to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/cbb575_rw747/.conda/envs/cbb575/lib/python3.13/site-packages/sklearn/metrics/_
classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being s
et to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/cbb575_rw747/.conda/envs/cbb575/lib/python3.13/site-packages/sklearn/metrics/_
classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being s
et to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
AUC: 0.4847, F1: 0.1470
[[  0 534]
 [  0  46]]
              precision    recall  f1-score   support

         0.0       0.00      0.00      0.00       534
         1.0       0.08      1.00      0.15        46

    accuracy                           0.08       580
   macro avg       0.04      0.50      0.07       580
weighted avg       0.01      0.08      0.01       580

Epoch 5/10
```

```
/home/cbb575_rw747/.conda/envs/cbb575/lib/python3.13/site-packages/sklearn/metrics/_
classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being s
et to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/cbb575_rw747/.conda/envs/cbb575/lib/python3.13/site-packages/sklearn/metrics/_
classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being s
et to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/cbb575_rw747/.conda/envs/cbb575/lib/python3.13/site-packages/sklearn/metrics/_
classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being s
et to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
AUC: 0.4594, F1: 0.0000
[[534   0]
 [ 46   0]]
              precision    recall  f1-score   support

         0.0       0.92      1.00      0.96       534
         1.0       0.00      0.00      0.00        46

    accuracy                           0.92       580
   macro avg       0.46      0.50      0.48       580
weighted avg       0.85      0.92      0.88       580

Epoch 6/10
/home/cbb575_rw747/.conda/envs/cbb575/lib/python3.13/site-packages/sklearn/metrics/_
classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being s
et to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/cbb575_rw747/.conda/envs/cbb575/lib/python3.13/site-packages/sklearn/metrics/_
classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being s
et to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/cbb575_rw747/.conda/envs/cbb575/lib/python3.13/site-packages/sklearn/metrics/_
classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being s
et to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
AUC: 0.5000, F1: 0.0000
[[534   0]
 [ 46   0]]
              precision    recall  f1-score   support

         0.0       0.92      1.00      0.96       534
         1.0       0.00      0.00      0.00        46

    accuracy                           0.92       580
   macro avg       0.46      0.50      0.48       580
weighted avg       0.85      0.92      0.88       580

Epoch 7/10
```

```
/home/cbb575_rw747/.conda/envs/cbb575/lib/python3.13/site-packages/sklearn/metrics/_
classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being s
et to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/cbb575_rw747/.conda/envs/cbb575/lib/python3.13/site-packages/sklearn/metrics/_
classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being s
et to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/cbb575_rw747/.conda/envs/cbb575/lib/python3.13/site-packages/sklearn/metrics/_
classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being s
et to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
AUC: 0.5299, F1: 0.0000
[[534   0]
 [ 46   0]]
              precision    recall  f1-score   support

         0.0       0.92      1.00      0.96       534
         1.0       0.00      0.00      0.00        46

    accuracy                           0.92       580
   macro avg       0.46      0.50      0.48       580
weighted avg       0.85      0.92      0.88       580

Epoch 8/10
```

```
/home/cbb575_rw747/.conda/envs/cbb575/lib/python3.13/site-packages/sklearn/metrics/_
classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being s
et to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/cbb575_rw747/.conda/envs/cbb575/lib/python3.13/site-packages/sklearn/metrics/_
classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being s
et to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/cbb575_rw747/.conda/envs/cbb575/lib/python3.13/site-packages/sklearn/metrics/_
classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being s
et to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
AUC: 0.5085, F1: 0.1470
[[  0 534]
 [  0  46]]
              precision    recall  f1-score   support

         0.0       0.00      0.00      0.00       534
         1.0       0.08      1.00      0.15        46

    accuracy                           0.08       580
   macro avg       0.04      0.50      0.07       580
weighted avg       0.01      0.08      0.01       580

Epoch 9/10
/home/cbb575_rw747/.conda/envs/cbb575/lib/python3.13/site-packages/sklearn/metrics/_
classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being s
et to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/cbb575_rw747/.conda/envs/cbb575/lib/python3.13/site-packages/sklearn/metrics/_
classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being s
et to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/cbb575_rw747/.conda/envs/cbb575/lib/python3.13/site-packages/sklearn/metrics/_
classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being s
et to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
AUC: 0.4703, F1: 0.0000
[[534   0]
 [ 46   0]]
              precision    recall  f1-score   support

         0.0       0.92      1.00      0.96       534
         1.0       0.00      0.00      0.00        46

    accuracy                           0.92       580
   macro avg       0.46      0.50      0.48       580
weighted avg       0.85      0.92      0.88       580

Epoch 10/10
/home/cbb575_rw747/.conda/envs/cbb575/lib/python3.13/site-packages/sklearn/metrics/_
classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being s
et to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/cbb575_rw747/.conda/envs/cbb575/lib/python3.13/site-packages/sklearn/metrics/_
classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being s
et to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/cbb575_rw747/.conda/envs/cbb575/lib/python3.13/site-packages/sklearn/metrics/_
classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being s
et to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
AUC: 0.5000, F1: 0.0000
[[534   0]
 [ 46   0]]
              precision    recall  f1-score   support

         0.0       0.92      1.00      0.96       534
         1.0       0.00      0.00      0.00        46

    accuracy                           0.92       580
   macro avg       0.46      0.50      0.48       580
weighted avg       0.85      0.92      0.88       580
```

In [1]:

```python
# bert_baseline_with_plots.py

import math
import pandas as pd
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader, random_split
from transformers import AutoTokenizer, AutoModel
from tqdm import tqdm
from sklearn.metrics import roc_auc_score, f1_score
import matplotlib.pyplot as plt
import gc

class ICUTextDataset(Dataset):
    def __init__(self, csv_path, tokenizer_name, max_length, mode='both'):
```

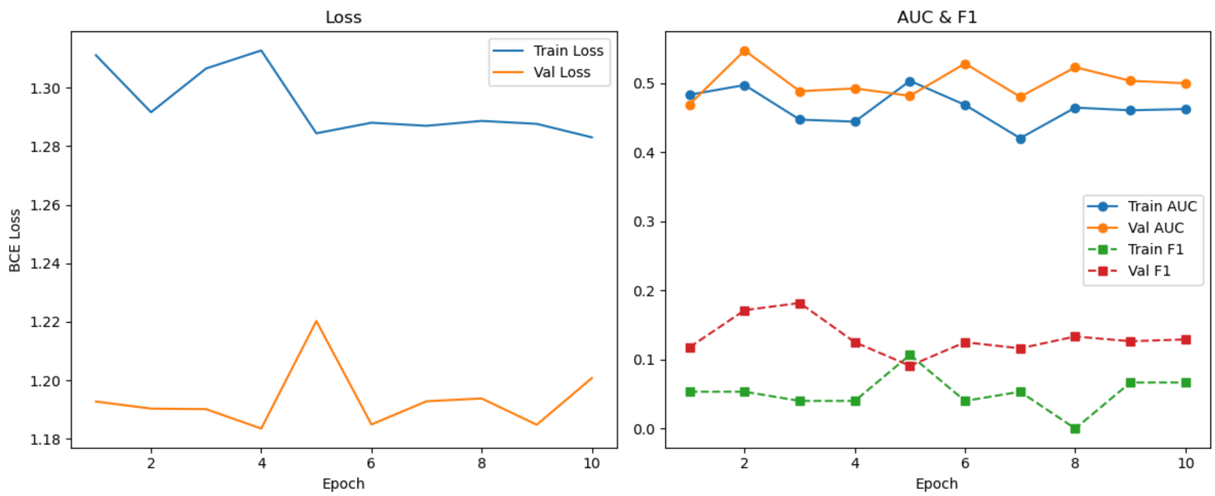
```python
        self.data = pd.read_csv(csv_path).reset_index(drop=True)
        self.tokenizer = AutoTokenizer.from_pretrained(tokenizer_name)
        self.max_length = max_length
        self.mode = mode

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

    def __getitem__(self, idx):
        row      = self.data.iloc[idx]
        text     = str(row['text_note'])
        combined = str(row['combined_note'])
        if   self.mode=='text_only':    full_text = text
        elif self.mode=='combined_only': full_text = combined
        else:                           full_text = text + ' ' + combined

        enc = self.tokenizer(
            full_text,
            max_length=self.max_length,
            padding='max_length',
            truncation=True,
            return_tensors='pt'
        )
        return {
            'input_ids':      enc['input_ids'].squeeze(0),
            'attention_mask': enc['attention_mask'].squeeze(0),
            'label':          torch.tensor(row['mortality_label'], dtype=torch.floa
        }

class BERTClassifier(nn.Module):
    def __init__(self, encoder_name, hidden_dim=768):
        super().__init__()
        self.encoder = AutoModel.from_pretrained(encoder_name)
        self.classifier = nn.Sequential(
            nn.Linear(hidden_dim, 128),
            nn.ReLU(),
            nn.Linear(128, 1)
        )

    def forward(self, input_ids, attention_mask):
        out = self.encoder(input_ids=input_ids, attention_mask=attention_mask)
        cls_emb = out.last_hidden_state[:,0,:]
        return self.classifier(cls_emb)

def compute_metrics_at_balanced_rate(probs, labels):
    P = int(sum(labels))
    if P > 0:
        thr = sorted(probs, reverse=True)[P-1]
    else:
        thr = 1.0
    preds = [1 if p>=thr else 0 for p in probs]
    auc = roc_auc_score(labels, probs)
    f1  = f1_score(labels, preds, zero_division=0)
    return auc, f1

def train_epoch(model, loader, device, loss_fn, optimizer):
```

```
    model.train()
    total_loss = 0.0
    all_probs, all_labels = [], []
    for batch in loader:
        ids    = batch['input_ids'].to(device)
        mask   = batch['attention_mask'].to(device)
        labels = batch['label'].to(device).unsqueeze(1)
        logits = model(ids, mask)
        loss   = loss_fn(logits, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        total_loss += loss.item()

        probs  = torch.sigmoid(logits).detach().cpu().squeeze().tolist()
        all_probs .extend(probs if isinstance(probs,list) else [probs])
        all_labels.extend(labels.cpu().squeeze().tolist())

    avg_loss = total_loss / len(loader)
    train_auc, train_f1 = compute_metrics_at_balanced_rate(all_probs, all_labels)
    return avg_loss, train_auc, train_f1

def eval_epoch(model, loader, device, loss_fn):
    model.eval()
    total_loss = 0.0
    all_probs, all_labels = [], []
    with torch.no_grad():
        for batch in loader:
            ids    = batch['input_ids'].to(device)
            mask   = batch['attention_mask'].to(device)
            labels = batch['label'].to(device).unsqueeze(1)
            logits = model(ids, mask)
            loss   = loss_fn(logits, labels)
            total_loss += loss.item()

            probs  = torch.sigmoid(logits).cpu().squeeze().tolist()
            all_probs .extend(probs if isinstance(probs,list) else [probs])
            all_labels.extend(labels.cpu().squeeze().tolist())

    avg_loss = total_loss / len(loader)
    val_auc, val_f1 = compute_metrics_at_balanced_rate(all_probs, all_labels)
    return avg_loss, val_auc, val_f1

def run_experiment(model_name="bert-base-uncased", max_length=512, mode_label="both
    # Prepare data
    ds = ICUTextDataset("final.csv", model_name, max_length, mode=mode_label)
    n  = len(ds)
    train_n = int(0.8 * n)
    train_ds, val_ds = random_split(ds, [train_n, n-train_n])

    train_loader  = DataLoader(train_ds, batch_size=4, shuffle=True)
    val_loader    = DataLoader(val_ds, batch_size=4, shuffle=False)

    # Model + optimizer + loss
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model  = BERTClassifier(model_name).to(device)
```

```
# Compute pos_weight
train_labels = [ds[i]['label'].item() for i in train_ds.indices]
neg = train_labels.count(0)
pos = train_labels.count(1)
pos_weight = torch.tensor([(neg/pos) if pos>0 else 1.0], device=device)
loss_fn = nn.BCEWithLogitsLoss(pos_weight=pos_weight)
optimizer = torch.optim.AdamW(model.parameters(), lr=1e-4)

# Storage
epochs = 10
train_losses, train_aucs, train_f1s = [], [], []
val_losses,   val_aucs,   val_f1s   = [], [], []

# Training Loop
for epoch in range(1, epochs+1):
    print(f"\nEpoch {epoch}/{epochs}")
    tr_loss, tr_auc, tr_f1 = train_epoch(model, train_loader, device, loss_fn,
    va_loss, va_auc, va_f1 = eval_epoch (model, val_loader,   device, loss_fn)

    train_losses.append(tr_loss)
    train_aucs.append(tr_auc)
    train_f1s.append(tr_f1)
    val_losses.append(va_loss)
    val_aucs.append(va_auc)
    val_f1s.append(va_f1)

    print(f" Train loss={tr_loss:.4f}, AUC={tr_auc:.4f}, F1={tr_f1:.4f}")
    print(f"   Val loss={va_loss:.4f}, AUC={va_auc:.4f}, F1={va_f1:.4f}")

# Plotting
plt.figure(figsize=(12,5))

plt.subplot(1,2,1)
plt.plot(range(1,epochs+1), train_losses, label="Train Loss")
plt.plot(range(1,epochs+1), val_losses,   label="Val Loss")
plt.xlabel("Epoch"); plt.ylabel("BCE Loss"); plt.title("Loss")
plt.legend()

plt.subplot(1,2,2)
plt.plot(range(1,epochs+1), train_aucs, '-o', label="Train AUC")
plt.plot(range(1,epochs+1), val_aucs,   '-o', label="Val AUC")
plt.plot(range(1,epochs+1), train_f1s,  '--s', label="Train F1")
plt.plot(range(1,epochs+1), val_f1s,    '--s', label="Val F1")
plt.xlabel("Epoch"); plt.title("AUC & F1")
plt.legend()

plt.tight_layout()
plt.show()

# Cleanup
del model
torch.cuda.empty_cache()
gc.collect()
```

In [2]:
```
run_experiment()
```

```
Epoch 1/10
 Train loss=1.3111, AUC=0.4833, F1=0.0533
   Val loss=1.1928, AUC=0.4693, F1=0.1176

Epoch 2/10
 Train loss=1.2917, AUC=0.4974, F1=0.0533
   Val loss=1.1903, AUC=0.5475, F1=0.1714

Epoch 3/10
 Train loss=1.3066, AUC=0.4473, F1=0.0400
   Val loss=1.1902, AUC=0.4884, F1=0.1818

Epoch 4/10
 Train loss=1.3128, AUC=0.4443, F1=0.0400
   Val loss=1.1835, AUC=0.4925, F1=0.1250

Epoch 5/10
 Train loss=1.2845, AUC=0.5034, F1=0.1067
   Val loss=1.2203, AUC=0.4818, F1=0.0909

Epoch 6/10
 Train loss=1.2881, AUC=0.4686, F1=0.0400
   Val loss=1.1849, AUC=0.5285, F1=0.1250

Epoch 7/10
 Train loss=1.2870, AUC=0.4204, F1=0.0533
   Val loss=1.1929, AUC=0.4803, F1=0.1161

Epoch 8/10
 Train loss=1.2887, AUC=0.4647, F1=0.0000
   Val loss=1.1938, AUC=0.5231, F1=0.1333

Epoch 9/10
 Train loss=1.2877, AUC=0.4608, F1=0.0667
   Val loss=1.1848, AUC=0.5035, F1=0.1264

Epoch 10/10
 Train loss=1.2831, AUC=0.4626, F1=0.0667
   Val loss=1.2008, AUC=0.5000, F1=0.1290
```



Tune the imbalance class

In [3]:

```
import math
import pandas as pd
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader, random_split
from transformers import AutoTokenizer, AutoModel
from tqdm import tqdm
from sklearn.metrics import (
    roc_auc_score, accuracy_score, precision_score, recall_score,
    f1_score, matthews_corrcoef, confusion_matrix
)
import matplotlib.pyplot as plt
import gc

class ICUTextDataset(Dataset):
    def __init__(self, csv_path, tokenizer_name, max_length, mode='both'):
        self.data = pd.read_csv(csv_path).reset_index(drop=True)
        self.tokenizer = AutoTokenizer.from_pretrained(tokenizer_name)
        self.max_length = max_length
        self.mode = mode

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

    def __getitem__(self, idx):
        row      = self.data.iloc[idx]
        text     = str(row['text_note'])
        combined = str(row['combined_note'])
        if   self.mode=='text_only':     full_text = text
        elif self.mode=='combined_only': full_text = combined
        else:                            full_text = text + ' ' + combined

        enc = self.tokenizer(
            full_text,
            max_length=self.max_length,
            padding='max_length',
            truncation=True,
```

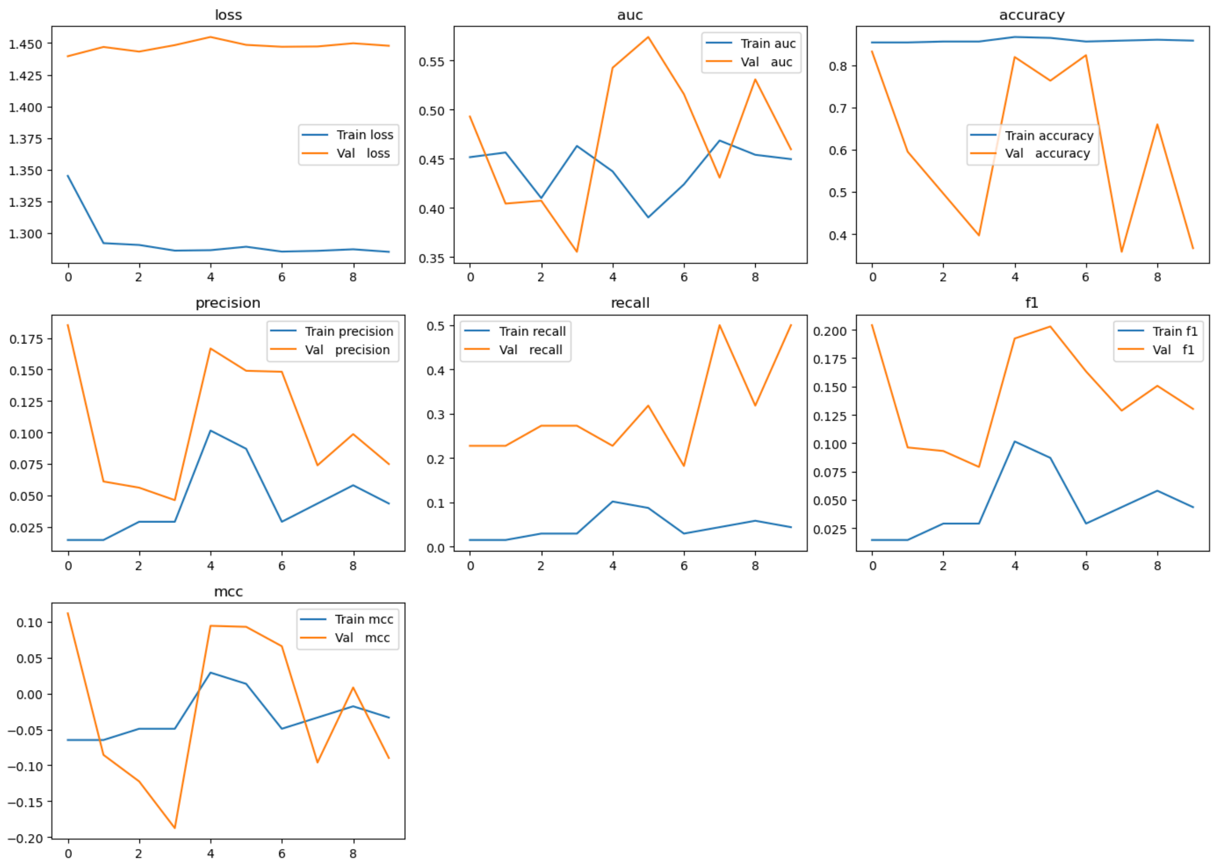
```
            return_tensors='pt'
        )
        return {
            'input_ids':      enc['input_ids'].squeeze(0),
            'attention_mask': enc['attention_mask'].squeeze(0),
            'label':          torch.tensor(row['mortality_label'], dtype=torch.floa

class BERTClassifier(nn.Module):
    def __init__(self, encoder_name, hidden_dim=768):
        super().__init__()
        self.encoder = AutoModel.from_pretrained(encoder_name)
        self.classifier = nn.Sequential(
            nn.Linear(hidden_dim, 128),
            nn.ReLU(),
            nn.Linear(128, 1)
        )

    def forward(self, input_ids, attention_mask):
        out = self.encoder(input_ids=input_ids, attention_mask=attention_mask)
        cls_emb = out.last_hidden_state[:,0,:]
        return self.classifier(cls_emb)

def compute_metrics(probs, labels):
    # determine threshold to match positive rate
    P = int(sum(labels))
    if P > 0:
        thr = sorted(probs, reverse=True)[P-1]
    else:
        thr = 1.0
    preds = [1 if p>=thr else 0 for p in probs]
    tn, fp, fn, tp = confusion_matrix(labels, preds).ravel()
    return {
        'auc': roc_auc_score(labels, probs),
        'accuracy': accuracy_score(labels, preds),
        'precision': precision_score(labels, preds, zero_division=0),
        'recall': recall_score(labels, preds, zero_division=0),
        'f1': f1_score(labels, preds, zero_division=0),
        'mcc': matthews_corrcoef(labels, preds),
        'specificity': tn/(tn+fp) if (tn+fp)>0 else 0,
        'npv': tn/(tn+fn) if (tn+fn)>0 else 0,
        'threshold': thr
    }

def train_epoch(model, loader, device, loss_fn, optimizer):
    model.train()
    total_loss = 0
    all_probs, all_labels = [], []
    for batch in tqdm(loader, desc="Train", leave=False):
        ids    = batch['input_ids'].to(device)
        mask   = batch['attention_mask'].to(device)
        labels = batch['label'].to(device).unsqueeze(1)
        logits = model(ids, mask)
        loss   = loss_fn(logits, labels)
        optimizer.zero_grad()
        loss.backward()
```

```python
        optimizer.step()
        total_loss += loss.item()
        probs = torch.sigmoid(logits).cpu().squeeze().tolist()
        all_probs.extend(probs if isinstance(probs, list) else [probs])
        all_labels.extend(labels.cpu().squeeze().tolist())
    metrics = compute_metrics(all_probs, all_labels)
    metrics['loss'] = total_loss/len(loader)
    return metrics

def eval_epoch(model, loader, device, loss_fn):
    model.eval()
    total_loss = 0
    all_probs, all_labels = [], []
    with torch.no_grad():
        for batch in tqdm(loader, desc="Eval", leave=False):
            ids    = batch['input_ids'].to(device)
            mask   = batch['attention_mask'].to(device)
            labels = batch['label'].to(device).unsqueeze(1)
            logits = model(ids, mask)
            loss   = loss_fn(logits, labels)
            total_loss += loss.item()
            probs = torch.sigmoid(logits).cpu().squeeze().tolist()
            all_probs.extend(probs if isinstance(probs, list) else [probs])
            all_labels.extend(labels.cpu().squeeze().tolist())
    metrics = compute_metrics(all_probs, all_labels)
    metrics['loss'] = total_loss/len(loader)
    return metrics

def run_experiment(model_name="bert-base-uncased", max_length=512, mode_label="both
    ds = ICUTextDataset("final.csv", model_name, max_length, mode_label)
    n  = len(ds)
    train_n = int(0.8*n)
    train_ds, val_ds = random_split(ds, [train_n, n-train_n])
    train_loader = DataLoader(train_ds, batch_size=4, shuffle=True)
    val_loader   = DataLoader(val_ds, batch_size=4, shuffle=False)

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model  = BERTClassifier(model_name).to(device)
    train_labels = [ds[i]['label'].item() for i in train_ds.indices]
    neg, pos = train_labels.count(0), train_labels.count(1)
    pos_weight = torch.tensor([(neg/pos) if pos>0 else 1.0], device=device)
    loss_fn = nn.BCEWithLogitsLoss(pos_weight=pos_weight)
    optimizer = torch.optim.AdamW(model.parameters(), lr=1e-4)

    epochs = 10
    history = {k: [] for k in ['loss','auc','accuracy','precision','recall','f1','m
    history_val = {k: [] for k in history}

    for epoch in range(1, epochs+1):
        print(f"\nEpoch {epoch}/{epochs}")
        m_tr = train_epoch(model, train_loader, device, loss_fn, optimizer)
        m_val= eval_epoch (model, val_loader,   device, loss_fn)
        for k in history:
            history[k].append(m_tr[k])
            history_val[k].append(m_val[k])
        print(" Train:", {k: f"{m_tr[k]:.4f}" for k in ['loss','auc','accuracy','pr
```

```
        print(" Val:  ", {k: f"{m_val[k]:.4f}" for k in ['loss','auc','accuracy','p

    # Plotting selected metrics
    plt.figure(figsize=(14,10))
    metrics_to_plot = ['loss','auc','accuracy','precision','recall','f1','mcc']
    for i, metric in enumerate(metrics_to_plot,1):
        plt.subplot(3,3,i)
        plt.plot(history[metric], label=f"Train {metric}")
        plt.plot(history_val[metric], label=f"Val   {metric}")
        plt.title(metric)
        plt.legend()
    plt.tight_layout()
    plt.show()

    # cleanup
    del model
    torch.cuda.empty_cache()
    gc.collect()


run_experiment()
```

```
Epoch 1/10

 Train: {'loss': '1.3451', 'auc': '0.4516', 'accuracy': '0.8534', 'precision': '0.01
45', 'recall': '0.0145', 'f1': '0.0145', 'mcc': '-0.0647'}
 Val:   {'loss': '1.4396', 'auc': '0.4929', 'accuracy': '0.8319', 'precision': '0.18
52', 'recall': '0.2273', 'f1': '0.2041', 'mcc': '0.1119'}

Epoch 2/10

 Train: {'loss': '1.2920', 'auc': '0.4563', 'accuracy': '0.8534', 'precision': '0.01
45', 'recall': '0.0145', 'f1': '0.0145', 'mcc': '-0.0647'}
 Val:   {'loss': '1.4469', 'auc': '0.4044', 'accuracy': '0.5948', 'precision': '0.06
10', 'recall': '0.2273', 'f1': '0.0962', 'mcc': '-0.0854'}

Epoch 3/10

 Train: {'loss': '1.2906', 'auc': '0.4100', 'accuracy': '0.8556', 'precision': '0.02
90', 'recall': '0.0290', 'f1': '0.0290', 'mcc': '-0.0490'}
 Val:   {'loss': '1.4433', 'auc': '0.4074', 'accuracy': '0.4957', 'precision': '0.05
61', 'recall': '0.2727', 'f1': '0.0930', 'mcc': '-0.1224'}

Epoch 4/10

 Train: {'loss': '1.2861', 'auc': '0.4630', 'accuracy': '0.8556', 'precision': '0.02
90', 'recall': '0.0290', 'f1': '0.0290', 'mcc': '-0.0490'}
 Val:   {'loss': '1.4484', 'auc': '0.3554', 'accuracy': '0.3966', 'precision': '0.04
62', 'recall': '0.2727', 'f1': '0.0789', 'mcc': '-0.1876'}

Epoch 5/10
```

```
 Train: {'loss': '1.2865', 'auc': '0.4372', 'accuracy': '0.8664', 'precision': '0.1014', 'recall': '0.1014', 'f1': '0.1014', 'mcc': '0.0293'}
 Val:   {'loss': '1.4548', 'auc': '0.5424', 'accuracy': '0.8190', 'precision': '0.1667', 'recall': '0.2273', 'f1': '0.1923', 'mcc': '0.0945'}

Epoch 6/10

 Train: {'loss': '1.2892', 'auc': '0.3904', 'accuracy': '0.8642', 'precision': '0.0870', 'recall': '0.0870', 'f1': '0.0870', 'mcc': '0.0136'}
 Val:   {'loss': '1.4486', 'auc': '0.5737', 'accuracy': '0.7629', 'precision': '0.1489', 'recall': '0.3182', 'f1': '0.2029', 'mcc': '0.0931'}

Epoch 7/10

 Train: {'loss': '1.2853', 'auc': '0.4240', 'accuracy': '0.8556', 'precision': '0.0290', 'recall': '0.0290', 'f1': '0.0290', 'mcc': '-0.0490'}
 Val:   {'loss': '1.4471', 'auc': '0.5156', 'accuracy': '0.8233', 'precision': '0.1481', 'recall': '0.1818', 'f1': '0.1633', 'mcc': '0.0660'}

Epoch 8/10

 Train: {'loss': '1.2859', 'auc': '0.4685', 'accuracy': '0.8578', 'precision': '0.0435', 'recall': '0.0435', 'f1': '0.0435', 'mcc': '-0.0334'}
 Val:   {'loss': '1.4473', 'auc': '0.4308', 'accuracy': '0.3578', 'precision': '0.0738', 'recall': '0.5000', 'f1': '0.1287', 'mcc': '-0.0960'}

Epoch 9/10

 Train: {'loss': '1.2872', 'auc': '0.4540', 'accuracy': '0.8599', 'precision': '0.0580', 'recall': '0.0580', 'f1': '0.0580', 'mcc': '-0.0177'}
 Val:   {'loss': '1.4498', 'auc': '0.5306', 'accuracy': '0.6595', 'precision': '0.0986', 'recall': '0.3182', 'f1': '0.1505', 'mcc': '0.0085'}

Epoch 10/10

 Train: {'loss': '1.2851', 'auc': '0.4496', 'accuracy': '0.8578', 'precision': '0.0435', 'recall': '0.0435', 'f1': '0.0435', 'mcc': '-0.0334'}
 Val:   {'loss': '1.4479', 'auc': '0.4596', 'accuracy': '0.3664', 'precision': '0.0748', 'recall': '0.5000', 'f1': '0.1302', 'mcc': '-0.0898'}
```



Run with 100 epoch

In [4]:

```
# bert_baseline_with_full_metrics.py

import math
import pandas as pd
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader, random_split
from transformers import AutoTokenizer, AutoModel
from tqdm import tqdm
from sklearn.metrics import (
    roc_auc_score, accuracy_score, precision_score, recall_score,
    f1_score, matthews_corrcoef, confusion_matrix
)
import matplotlib.pyplot as plt
import gc

class ICUTextDataset(Dataset):
    def __init__(self, csv_path, tokenizer_name, max_length, mode='both'):
        self.data = pd.read_csv(csv_path).reset_index(drop=True)
        self.tokenizer = AutoTokenizer.from_pretrained(tokenizer_name)
        self.max_length = max_length
        self.mode = mode

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

```python
    def __getitem__(self, idx):
        row      = self.data.iloc[idx]
        text     = str(row['text_note'])
        combined = str(row['combined_note'])
        if   self.mode=='text_only':    full_text = text
        elif self.mode=='combined_only': full_text = combined
        else:                           full_text = text + ' ' + combined

        enc = self.tokenizer(
            full_text,
            max_length=self.max_length,
            padding='max_length',
            truncation=True,
            return_tensors='pt'
        )
        return {
            'input_ids':      enc['input_ids'].squeeze(0),
            'attention_mask': enc['attention_mask'].squeeze(0),
            'label':          torch.tensor(row['mortality_label'], dtype=torch.floa
        }

class BERTClassifier(nn.Module):
    def __init__(self, encoder_name, hidden_dim=768):
        super().__init__()
        self.encoder = AutoModel.from_pretrained(encoder_name)
        self.classifier = nn.Sequential(
            nn.Linear(hidden_dim, 128),
            nn.ReLU(),
            nn.Linear(128, 1)
        )

    def forward(self, input_ids, attention_mask):
        out = self.encoder(input_ids=input_ids, attention_mask=attention_mask)
        cls_emb = out.last_hidden_state[:,0,:]
        return self.classifier(cls_emb)

def compute_metrics(probs, labels):
    # determine threshold to match positive rate
    P = int(sum(labels))
    if P > 0:
        thr = sorted(probs, reverse=True)[P-1]
    else:
        thr = 1.0
    preds = [1 if p>=thr else 0 for p in probs]
    tn, fp, fn, tp = confusion_matrix(labels, preds).ravel()
    return {
        'auc': roc_auc_score(labels, probs),
        'accuracy': accuracy_score(labels, preds),
        'precision': precision_score(labels, preds, zero_division=0),
        'recall': recall_score(labels, preds, zero_division=0),
        'f1': f1_score(labels, preds, zero_division=0),
        'mcc': matthews_corrcoef(labels, preds),
        'specificity': tn/(tn+fp) if (tn+fp)>0 else 0,
        'npv': tn/(tn+fn) if (tn+fn)>0 else 0,
        'threshold': thr
    }
```

```
def train_epoch(model, loader, device, loss_fn, optimizer):
    model.train()
    total_loss = 0
    all_probs, all_labels = [], []
    for batch in tqdm(loader, desc="Train", leave=False):
        ids    = batch['input_ids'].to(device)
        mask   = batch['attention_mask'].to(device)
        labels = batch['label'].to(device).unsqueeze(1)
        logits = model(ids, mask)
        loss   = loss_fn(logits, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
        probs = torch.sigmoid(logits).cpu().squeeze().tolist()
        all_probs.extend(probs if isinstance(probs, list) else [probs])
        all_labels.extend(labels.cpu().squeeze().tolist())
    metrics = compute_metrics(all_probs, all_labels)
    metrics['loss'] = total_loss/len(loader)
    return metrics

def eval_epoch(model, loader, device, loss_fn):
    model.eval()
    total_loss = 0
    all_probs, all_labels = [], []
    with torch.no_grad():
        for batch in tqdm(loader, desc="Eval", leave=False):
            ids    = batch['input_ids'].to(device)
            mask   = batch['attention_mask'].to(device)
            labels = batch['label'].to(device).unsqueeze(1)
            logits = model(ids, mask)
            loss   = loss_fn(logits, labels)
            total_loss += loss.item()
            probs = torch.sigmoid(logits).cpu().squeeze().tolist()
            all_probs.extend(probs if isinstance(probs, list) else [probs])
            all_labels.extend(labels.cpu().squeeze().tolist())
    metrics = compute_metrics(all_probs, all_labels)
    metrics['loss'] = total_loss/len(loader)
    return metrics

def run_experiment(model_name="bert-base-uncased", max_length=512, mode_label="both
    ds = ICUTextDataset("final.csv", model_name, max_length, mode_label)
    n  = len(ds)
    train_n = int(0.8*n)
    train_ds, val_ds = random_split(ds, [train_n, n-train_n])
    train_loader = DataLoader(train_ds, batch_size=4, shuffle=True)
    val_loader   = DataLoader(val_ds, batch_size=4, shuffle=False)

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model  = BERTClassifier(model_name).to(device)
    train_labels = [ds[i]['label'].item() for i in train_ds.indices]
    neg, pos = train_labels.count(0), train_labels.count(1)
    pos_weight = torch.tensor([(neg/pos) if pos>0 else 1.0], device=device)
    loss_fn = nn.BCEWithLogitsLoss(pos_weight=pos_weight)
    optimizer = torch.optim.AdamW(model.parameters(), lr=1e-4)
```

```python
    epochs = 100
    history = {k: [] for k in ['loss','auc','accuracy','precision','recall','f1','m
    history_val = {k: [] for k in history}

    for epoch in range(1, epochs+1):
        print(f"\nEpoch {epoch}/{epochs}")
        m_tr = train_epoch(model, train_loader, device, loss_fn, optimizer)
        m_val= eval_epoch (model, val_loader,   device, loss_fn)
        for k in history:
            history[k].append(m_tr[k])
            history_val[k].append(m_val[k])
        print(" Train:", {k: f"{m_tr[k]:.4f}" for k in ['loss','auc','accuracy','pr
        print(" Val:  ", {k: f"{m_val[k]:.4f}" for k in ['loss','auc','accuracy','p

    # Plotting selected metrics
    plt.figure(figsize=(14,10))
    metrics_to_plot = ['loss','auc','accuracy','precision','recall','f1','mcc']
    for i, metric in enumerate(metrics_to_plot,1):
        plt.subplot(3,3,i)
        plt.plot(history[metric], label=f"Train {metric}")
        plt.plot(history_val[metric], label=f"Val   {metric}")
        plt.title(metric)
        plt.legend()
    plt.tight_layout()
    plt.show()

    # cleanup
    del model
    torch.cuda.empty_cache()
    gc.collect()


run_experiment()
```

```
Epoch 1/100

 Train: {'loss': '1.3261', 'auc': '0.5010', 'accuracy': '0.8578', 'precision': '0.08
33', 'recall': '0.0833', 'f1': '0.0833', 'mcc': '0.0062'}
 Val:   {'loss': '1.3402', 'auc': '0.4328', 'accuracy': '0.8448', 'precision': '0.09
52', 'recall': '0.1053', 'f1': '0.1000', 'mcc': '0.0153'}

Epoch 2/100

 Train: {'loss': '1.3222', 'auc': '0.5095', 'accuracy': '0.8491', 'precision': '0.02
78', 'recall': '0.0278', 'f1': '0.0278', 'mcc': '-0.0540'}
 Val:   {'loss': '1.3281', 'auc': '0.5638', 'accuracy': '0.7888', 'precision': '0.00
00', 'recall': '0.0000', 'f1': '0.0000', 'mcc': '-0.1151'}

Epoch 3/100
```

```
 Train: {'loss': '1.3013', 'auc': '0.4951', 'accuracy': '0.8599', 'precision': '0.09
72', 'recall': '0.0972', 'f1': '0.0972', 'mcc': '0.0213'}
 Val:   {'loss': '1.3360', 'auc': '0.4733', 'accuracy': '0.6121', 'precision': '0.08
24', 'recall': '0.3684', 'f1': '0.1346', 'mcc': '0.0013'}

Epoch 4/100

 Train: {'loss': '1.2964', 'auc': '0.4884', 'accuracy': '0.8534', 'precision': '0.05
56', 'recall': '0.0556', 'f1': '0.0556', 'mcc': '-0.0239'}
 Val:   {'loss': '1.3119', 'auc': '0.4430', 'accuracy': '0.8147', 'precision': '0.07
14', 'recall': '0.1053', 'f1': '0.0851', 'mcc': '-0.0141'}

Epoch 5/100

 Train: {'loss': '1.3013', 'auc': '0.4413', 'accuracy': '0.8534', 'precision': '0.05
56', 'recall': '0.0556', 'f1': '0.0556', 'mcc': '-0.0239'}
 Val:   {'loss': '1.3108', 'auc': '0.5125', 'accuracy': '0.8448', 'precision': '0.13
04', 'recall': '0.1579', 'f1': '0.1429', 'mcc': '0.0587'}

Epoch 6/100

 Train: {'loss': '1.2991', 'auc': '0.4966', 'accuracy': '0.8534', 'precision': '0.05
56', 'recall': '0.0556', 'f1': '0.0556', 'mcc': '-0.0239'}
 Val:   {'loss': '1.3198', 'auc': '0.5561', 'accuracy': '0.7500', 'precision': '0.13
21', 'recall': '0.3684', 'f1': '0.1944', 'mcc': '0.0996'}

Epoch 7/100

 Train: {'loss': '1.2971', 'auc': '0.4537', 'accuracy': '0.8513', 'precision': '0.04
17', 'recall': '0.0417', 'f1': '0.0417', 'mcc': '-0.0389'}
 Val:   {'loss': '1.3118', 'auc': '0.4949', 'accuracy': '0.7586', 'precision': '0.04
88', 'recall': '0.1053', 'f1': '0.0667', 'mcc': '-0.0560'}

Epoch 8/100

 Train: {'loss': '1.2947', 'auc': '0.3778', 'accuracy': '0.8491', 'precision': '0.02
78', 'recall': '0.0278', 'f1': '0.0278', 'mcc': '-0.0540'}
 Val:   {'loss': '1.3122', 'auc': '0.5477', 'accuracy': '0.7284', 'precision': '0.09
26', 'recall': '0.2632', 'f1': '0.1370', 'mcc': '0.0215'}

Epoch 9/100

 Train: {'loss': '1.2880', 'auc': '0.4848', 'accuracy': '0.8534', 'precision': '0.05
56', 'recall': '0.0556', 'f1': '0.0556', 'mcc': '-0.0239'}
 Val:   {'loss': '1.3111', 'auc': '0.5190', 'accuracy': '0.6853', 'precision': '0.07
81', 'recall': '0.2632', 'f1': '0.1205', 'mcc': '-0.0085'}

Epoch 10/100
```

```
 Train: {'loss': '1.2846', 'auc': '0.5074', 'accuracy': '0.8578', 'precision': '0.08
33', 'recall': '0.0833', 'f1': '0.0833', 'mcc': '0.0062'}
 Val:   {'loss': '1.3108', 'auc': '0.4833', 'accuracy': '0.8405', 'precision': '0.05
00', 'recall': '0.0526', 'f1': '0.0513', 'mcc': '-0.0357'}

Epoch 11/100

 Train: {'loss': '1.2935', 'auc': '0.4626', 'accuracy': '0.8534', 'precision': '0.05
56', 'recall': '0.0556', 'f1': '0.0556', 'mcc': '-0.0239'}
 Val:   {'loss': '1.3110', 'auc': '0.5047', 'accuracy': '0.0905', 'precision': '0.08
26', 'recall': '1.0000', 'f1': '0.1526', 'mcc': '0.0279'}

Epoch 12/100

 Train: {'loss': '1.2860', 'auc': '0.4792', 'accuracy': '0.8491', 'precision': '0.02
78', 'recall': '0.0278', 'f1': '0.0278', 'mcc': '-0.0540'}
 Val:   {'loss': '1.3105', 'auc': '0.4732', 'accuracy': '0.1207', 'precision': '0.07
76', 'recall': '0.8947', 'f1': '0.1429', 'mcc': '-0.0639'}

Epoch 13/100

 Train: {'loss': '1.2881', 'auc': '0.4410', 'accuracy': '0.8513', 'precision': '0.04
17', 'recall': '0.0417', 'f1': '0.0417', 'mcc': '-0.0389'}
 Val:   {'loss': '1.3106', 'auc': '0.5098', 'accuracy': '0.2759', 'precision': '0.08
38', 'recall': '0.7895', 'f1': '0.1515', 'mcc': '0.0127'}

Epoch 14/100

 Train: {'loss': '1.2873', 'auc': '0.4328', 'accuracy': '0.8513', 'precision': '0.04
17', 'recall': '0.0417', 'f1': '0.0417', 'mcc': '-0.0389'}
 Val:   {'loss': '1.3120', 'auc': '0.5151', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 15/100

 Train: {'loss': '1.2838', 'auc': '0.4828', 'accuracy': '0.8578', 'precision': '0.08
33', 'recall': '0.0833', 'f1': '0.0833', 'mcc': '0.0062'}
 Val:   {'loss': '1.3147', 'auc': '0.5094', 'accuracy': '0.1078', 'precision': '0.08
41', 'recall': '1.0000', 'f1': '0.1551', 'mcc': '0.0487'}

Epoch 16/100

 Train: {'loss': '1.2854', 'auc': '0.4239', 'accuracy': '0.8556', 'precision': '0.06
94', 'recall': '0.0694', 'f1': '0.0694', 'mcc': '-0.0088'}
 Val:   {'loss': '1.3107', 'auc': '0.5618', 'accuracy': '0.5474', 'precision': '0.10
19', 'recall': '0.5789', 'f1': '0.1732', 'mcc': '0.0679'}

Epoch 17/100
```

```
 Train: {'loss': '1.2828', 'auc': '0.4726', 'accuracy': '0.8534', 'precision': '0.05
56', 'recall': '0.0556', 'f1': '0.0556', 'mcc': '-0.0239'}
 Val:   {'loss': '1.3105', 'auc': '0.5634', 'accuracy': '0.1983', 'precision': '0.09
27', 'recall': '1.0000', 'f1': '0.1696', 'mcc': '0.1084'}

Epoch 18/100

 Train: {'loss': '1.2819', 'auc': '0.4840', 'accuracy': '0.8534', 'precision': '0.05
56', 'recall': '0.0556', 'f1': '0.0556', 'mcc': '-0.0239'}
 Val:   {'loss': '1.3180', 'auc': '0.4619', 'accuracy': '0.4784', 'precision': '0.06
78', 'recall': '0.4211', 'f1': '0.1168', 'mcc': '-0.0523'}

Epoch 19/100

 Train: {'loss': '1.2830', 'auc': '0.5005', 'accuracy': '0.8599', 'precision': '0.09
72', 'recall': '0.0972', 'f1': '0.0972', 'mcc': '0.0213'}
 Val:   {'loss': '1.3106', 'auc': '0.4737', 'accuracy': '0.0776', 'precision': '0.07
79', 'recall': '0.9474', 'f1': '0.1440', 'mcc': '-0.2203'}

Epoch 20/100

 Train: {'loss': '1.2837', 'auc': '0.4709', 'accuracy': '0.8534', 'precision': '0.05
56', 'recall': '0.0556', 'f1': '0.0556', 'mcc': '-0.0239'}
 Val:   {'loss': '1.3105', 'auc': '0.5352', 'accuracy': '0.0991', 'precision': '0.08
33', 'recall': '1.0000', 'f1': '0.1538', 'mcc': '0.0396'}

Epoch 21/100

 Train: {'loss': '1.2942', 'auc': '0.5207', 'accuracy': '0.8578', 'precision': '0.08
33', 'recall': '0.0833', 'f1': '0.0833', 'mcc': '0.0062'}
 Val:   {'loss': '1.3441', 'auc': '0.4411', 'accuracy': '0.1595', 'precision': '0.06
86', 'recall': '0.7368', 'f1': '0.1256', 'mcc': '-0.1306'}

Epoch 22/100

 Train: {'loss': '1.3047', 'auc': '0.4860', 'accuracy': '0.8556', 'precision': '0.06
94', 'recall': '0.0694', 'f1': '0.0694', 'mcc': '-0.0088'}
 Val:   {'loss': '1.3105', 'auc': '0.5162', 'accuracy': '0.5517', 'precision': '0.08
74', 'recall': '0.4737', 'f1': '0.1475', 'mcc': '0.0179'}

Epoch 23/100

 Train: {'loss': '1.2865', 'auc': '0.4380', 'accuracy': '0.8491', 'precision': '0.02
78', 'recall': '0.0278', 'f1': '0.0278', 'mcc': '-0.0540'}
 Val:   {'loss': '1.3107', 'auc': '0.5179', 'accuracy': '0.8190', 'precision': '0.10
34', 'recall': '0.1579', 'f1': '0.1250', 'mcc': '0.0297'}

Epoch 24/100
```

```
 Train: {'loss': '1.2854', 'auc': '0.4307', 'accuracy': '0.8513', 'precision': '0.04
17', 'recall': '0.0417', 'f1': '0.0417', 'mcc': '-0.0389'}
 Val:   {'loss': '1.3107', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 25/100

 Train: {'loss': '1.2851', 'auc': '0.4868', 'accuracy': '0.8491', 'precision': '0.02
78', 'recall': '0.0278', 'f1': '0.0278', 'mcc': '-0.0540'}
 Val:   {'loss': '1.3105', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 26/100

 Train: {'loss': '1.2814', 'auc': '0.4985', 'accuracy': '0.8599', 'precision': '0.09
72', 'recall': '0.0972', 'f1': '0.0972', 'mcc': '0.0213'}
 Val:   {'loss': '1.3170', 'auc': '0.4958', 'accuracy': '0.2931', 'precision': '0.08
09', 'recall': '0.7368', 'f1': '0.1458', 'mcc': '-0.0061'}

Epoch 27/100

 Train: {'loss': '1.2848', 'auc': '0.4473', 'accuracy': '0.8513', 'precision': '0.04
17', 'recall': '0.0417', 'f1': '0.0417', 'mcc': '-0.0389'}
 Val:   {'loss': '1.3105', 'auc': '0.5183', 'accuracy': '0.1595', 'precision': '0.08
49', 'recall': '0.9474', 'f1': '0.1558', 'mcc': '0.0357'}

Epoch 28/100

 Train: {'loss': '1.2831', 'auc': '0.4376', 'accuracy': '0.8513', 'precision': '0.04
17', 'recall': '0.0417', 'f1': '0.0417', 'mcc': '-0.0389'}
 Val:   {'loss': '1.3106', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 29/100

 Train: {'loss': '1.2854', 'auc': '0.4931', 'accuracy': '0.8491', 'precision': '0.02
78', 'recall': '0.0278', 'f1': '0.0278', 'mcc': '-0.0540'}
 Val:   {'loss': '1.3168', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 30/100

 Train: {'loss': '1.2845', 'auc': '0.4452', 'accuracy': '0.8491', 'precision': '0.02
78', 'recall': '0.0278', 'f1': '0.0278', 'mcc': '-0.0540'}
 Val:   {'loss': '1.3109', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 31/100
```

```
 Train: {'loss': '1.2870', 'auc': '0.4539', 'accuracy': '0.8534', 'precision': '0.05
56', 'recall': '0.0556', 'f1': '0.0556', 'mcc': '-0.0239'}
 Val:   {'loss': '1.3127', 'auc': '0.5234', 'accuracy': '0.3448', 'precision': '0.08
70', 'recall': '0.7368', 'f1': '0.1556', 'mcc': '0.0278'}

Epoch 32/100

 Train: {'loss': '1.2862', 'auc': '0.4495', 'accuracy': '0.8491', 'precision': '0.02
78', 'recall': '0.0278', 'f1': '0.0278', 'mcc': '-0.0540'}
 Val:   {'loss': '1.3162', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 33/100

 Train: {'loss': '1.2822', 'auc': '0.5169', 'accuracy': '0.8534', 'precision': '0.05
56', 'recall': '0.0556', 'f1': '0.0556', 'mcc': '-0.0239'}
 Val:   {'loss': '1.3276', 'auc': '0.4657', 'accuracy': '0.1509', 'precision': '0.07
62', 'recall': '0.8421', 'f1': '0.1397', 'mcc': '-0.0643'}

Epoch 34/100

 Train: {'loss': '1.2904', 'auc': '0.4385', 'accuracy': '0.8578', 'precision': '0.08
33', 'recall': '0.0833', 'f1': '0.0833', 'mcc': '0.0062'}
 Val:   {'loss': '1.3106', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 35/100

 Train: {'loss': '1.2830', 'auc': '0.5002', 'accuracy': '0.8491', 'precision': '0.02
78', 'recall': '0.0278', 'f1': '0.0278', 'mcc': '-0.0540'}
 Val:   {'loss': '1.3122', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 36/100

 Train: {'loss': '1.2882', 'auc': '0.4688', 'accuracy': '0.8556', 'precision': '0.06
94', 'recall': '0.0694', 'f1': '0.0694', 'mcc': '-0.0088'}
 Val:   {'loss': '1.3188', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 37/100

 Train: {'loss': '1.2889', 'auc': '0.4512', 'accuracy': '0.8470', 'precision': '0.01
39', 'recall': '0.0139', 'f1': '0.0139', 'mcc': '-0.0691'}
 Val:   {'loss': '1.3106', 'auc': '0.5914', 'accuracy': '0.4698', 'precision': '0.10
61', 'recall': '0.7368', 'f1': '0.1854', 'mcc': '0.1012'}

Epoch 38/100
```

```
 Train: {'loss': '1.2852', 'auc': '0.3968', 'accuracy': '0.8513', 'precision': '0.0417', 'recall': '0.0417', 'f1': '0.0417', 'mcc': '-0.0389'}
 Val:   {'loss': '1.3108', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.0819', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 39/100

 Train: {'loss': '1.2830', 'auc': '0.4232', 'accuracy': '0.8513', 'precision': '0.0417', 'recall': '0.0417', 'f1': '0.0417', 'mcc': '-0.0389'}
 Val:   {'loss': '1.3106', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.0819', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 40/100

 Train: {'loss': '1.2816', 'auc': '0.4857', 'accuracy': '0.8534', 'precision': '0.0556', 'recall': '0.0556', 'f1': '0.0556', 'mcc': '-0.0239'}
 Val:   {'loss': '1.3119', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.0819', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 41/100

 Train: {'loss': '1.2818', 'auc': '0.4794', 'accuracy': '0.8491', 'precision': '0.0278', 'recall': '0.0278', 'f1': '0.0278', 'mcc': '-0.0540'}
 Val:   {'loss': '1.3127', 'auc': '0.5047', 'accuracy': '0.0905', 'precision': '0.0826', 'recall': '1.0000', 'f1': '0.1526', 'mcc': '0.0279'}

Epoch 42/100

 Train: {'loss': '1.2835', 'auc': '0.4880', 'accuracy': '0.8621', 'precision': '0.1111', 'recall': '0.1111', 'f1': '0.1111', 'mcc': '0.0363'}
 Val:   {'loss': '1.3105', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.0819', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 43/100

 Train: {'loss': '1.2839', 'auc': '0.4252', 'accuracy': '0.8513', 'precision': '0.0417', 'recall': '0.0417', 'f1': '0.0417', 'mcc': '-0.0389'}
 Val:   {'loss': '1.3108', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.0819', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 44/100

 Train: {'loss': '1.2827', 'auc': '0.4209', 'accuracy': '0.8534', 'precision': '0.0556', 'recall': '0.0556', 'f1': '0.0556', 'mcc': '-0.0239'}
 Val:   {'loss': '1.3117', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.0819', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 45/100
```

```
 Train: {'loss': '1.2829', 'auc': '0.4424', 'accuracy': '0.8491', 'precision': '0.02
78', 'recall': '0.0278', 'f1': '0.0278', 'mcc': '-0.0540'}
 Val:   {'loss': '1.3122', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 46/100

 Train: {'loss': '1.2851', 'auc': '0.4221', 'accuracy': '0.8513', 'precision': '0.04
17', 'recall': '0.0417', 'f1': '0.0417', 'mcc': '-0.0389'}
 Val:   {'loss': '1.3109', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 47/100

 Train: {'loss': '1.2830', 'auc': '0.4439', 'accuracy': '0.8491', 'precision': '0.02
78', 'recall': '0.0278', 'f1': '0.0278', 'mcc': '-0.0540'}
 Val:   {'loss': '1.3111', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 48/100

 Train: {'loss': '1.2819', 'auc': '0.4418', 'accuracy': '0.8513', 'precision': '0.04
17', 'recall': '0.0417', 'f1': '0.0417', 'mcc': '-0.0389'}
 Val:   {'loss': '1.3114', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 49/100

 Train: {'loss': '1.2811', 'auc': '0.4814', 'accuracy': '0.8513', 'precision': '0.04
17', 'recall': '0.0417', 'f1': '0.0417', 'mcc': '-0.0389'}
 Val:   {'loss': '1.3116', 'auc': '0.4054', 'accuracy': '0.3922', 'precision': '0.05
80', 'recall': '0.4211', 'f1': '0.1019', 'mcc': '-0.1057'}

Epoch 50/100

 Train: {'loss': '1.2822', 'auc': '0.4541', 'accuracy': '0.8513', 'precision': '0.04
17', 'recall': '0.0417', 'f1': '0.0417', 'mcc': '-0.0389'}
 Val:   {'loss': '1.3108', 'auc': '0.5582', 'accuracy': '0.2328', 'precision': '0.09
23', 'recall': '0.9474', 'f1': '0.1682', 'mcc': '0.0872'}

Epoch 51/100

 Train: {'loss': '1.2836', 'auc': '0.4041', 'accuracy': '0.8513', 'precision': '0.04
17', 'recall': '0.0417', 'f1': '0.0417', 'mcc': '-0.0389'}
 Val:   {'loss': '1.3105', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 52/100
```

```
 Train: {'loss': '1.2842', 'auc': '0.4685', 'accuracy': '0.8491', 'precision': '0.02
78', 'recall': '0.0278', 'f1': '0.0278', 'mcc': '-0.0540'}
 Val:   {'loss': '1.3177', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 53/100

 Train: {'loss': '1.2900', 'auc': '0.4339', 'accuracy': '0.8470', 'precision': '0.01
39', 'recall': '0.0139', 'f1': '0.0139', 'mcc': '-0.0691'}
 Val:   {'loss': '1.3115', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 54/100

 Train: {'loss': '1.2815', 'auc': '0.4613', 'accuracy': '0.8470', 'precision': '0.01
39', 'recall': '0.0139', 'f1': '0.0139', 'mcc': '-0.0691'}
 Val:   {'loss': '1.3120', 'auc': '0.4859', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 55/100

 Train: {'loss': '1.2874', 'auc': '0.4770', 'accuracy': '0.8534', 'precision': '0.05
56', 'recall': '0.0556', 'f1': '0.0556', 'mcc': '-0.0239'}
 Val:   {'loss': '1.3113', 'auc': '0.5216', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 56/100

 Train: {'loss': '1.2868', 'auc': '0.4957', 'accuracy': '0.8556', 'precision': '0.06
94', 'recall': '0.0694', 'f1': '0.0694', 'mcc': '-0.0088'}
 Val:   {'loss': '1.3132', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 57/100

 Train: {'loss': '1.2849', 'auc': '0.4595', 'accuracy': '0.8513', 'precision': '0.04
17', 'recall': '0.0417', 'f1': '0.0417', 'mcc': '-0.0389'}
 Val:   {'loss': '1.3122', 'auc': '0.4859', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 58/100

 Train: {'loss': '1.2843', 'auc': '0.4330', 'accuracy': '0.8513', 'precision': '0.04
17', 'recall': '0.0417', 'f1': '0.0417', 'mcc': '-0.0389'}
 Val:   {'loss': '1.3108', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 59/100
```

```
 Train: {'loss': '1.2814', 'auc': '0.5000', 'accuracy': '0.8491', 'precision': '0.02
78', 'recall': '0.0278', 'f1': '0.0278', 'mcc': '-0.0540'}
 Val:   {'loss': '1.3241', 'auc': '0.4631', 'accuracy': '0.5862', 'precision': '0.06
74', 'recall': '0.3158', 'f1': '0.1111', 'mcc': '-0.0417'}

Epoch 60/100

 Train: {'loss': '1.2803', 'auc': '0.4963', 'accuracy': '0.8578', 'precision': '0.08
33', 'recall': '0.0833', 'f1': '0.0833', 'mcc': '0.0062'}
 Val:   {'loss': '1.3174', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 61/100

 Train: {'loss': '1.2854', 'auc': '0.4459', 'accuracy': '0.8491', 'precision': '0.02
78', 'recall': '0.0278', 'f1': '0.0278', 'mcc': '-0.0540'}
 Val:   {'loss': '1.3118', 'auc': '0.5495', 'accuracy': '0.5690', 'precision': '0.09
90', 'recall': '0.5263', 'f1': '0.1667', 'mcc': '0.0548'}

Epoch 62/100

 Train: {'loss': '1.2818', 'auc': '0.4500', 'accuracy': '0.8491', 'precision': '0.02
78', 'recall': '0.0278', 'f1': '0.0278', 'mcc': '-0.0540'}
 Val:   {'loss': '1.3115', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 63/100

 Train: {'loss': '1.2836', 'auc': '0.4513', 'accuracy': '0.8556', 'precision': '0.06
94', 'recall': '0.0694', 'f1': '0.0694', 'mcc': '-0.0088'}
 Val:   {'loss': '1.3107', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 64/100

 Train: {'loss': '1.2818', 'auc': '0.3931', 'accuracy': '0.8556', 'precision': '0.06
94', 'recall': '0.0694', 'f1': '0.0694', 'mcc': '-0.0088'}
 Val:   {'loss': '1.3107', 'auc': '0.4422', 'accuracy': '0.1078', 'precision': '0.07
27', 'recall': '0.8421', 'f1': '0.1339', 'mcc': '-0.1432'}

Epoch 65/100

 Train: {'loss': '1.2834', 'auc': '0.4717', 'accuracy': '0.8621', 'precision': '0.11
11', 'recall': '0.1111', 'f1': '0.1111', 'mcc': '0.0363'}
 Val:   {'loss': '1.3107', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 66/100

```

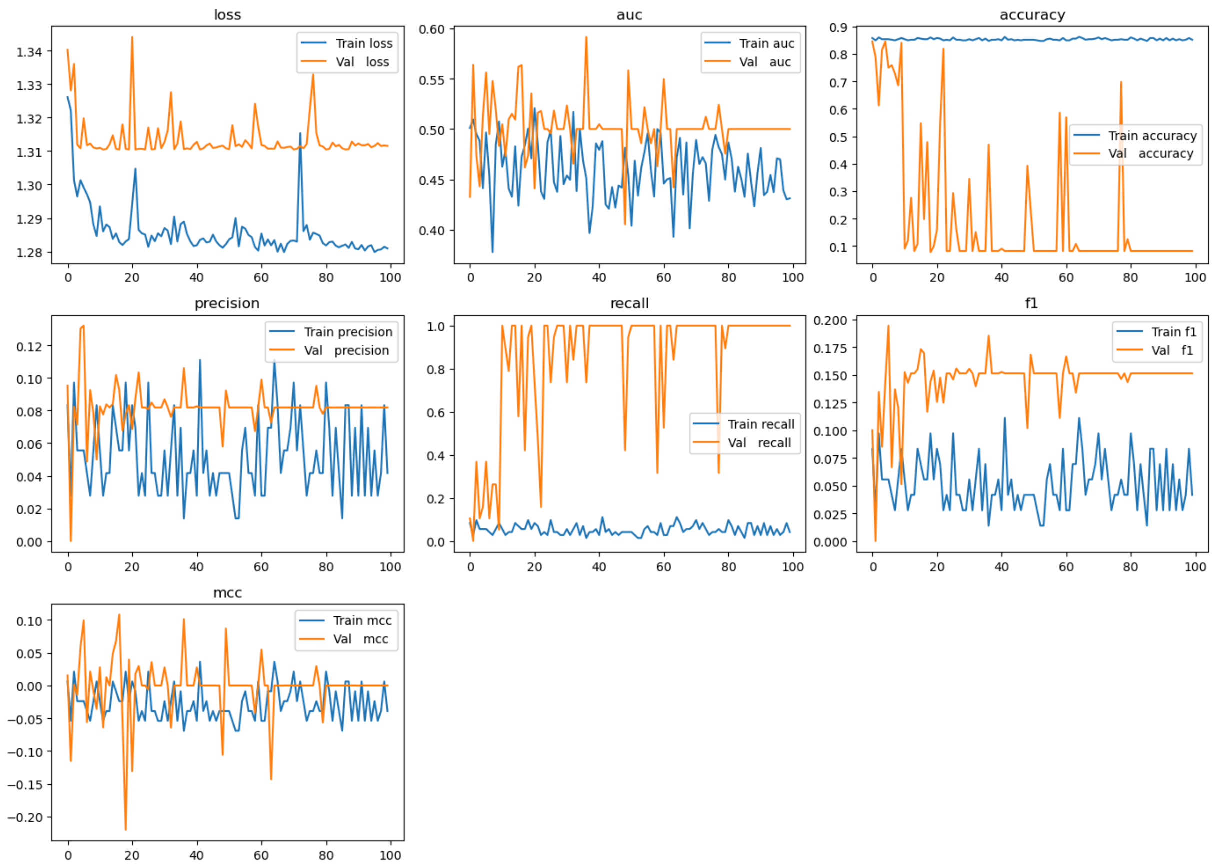
```
 Train: {'loss': '1.2799', 'auc': '0.4912', 'accuracy': '0.8578', 'precision': '0.08
33', 'recall': '0.0833', 'f1': '0.0833', 'mcc': '0.0062'}
 Val:   {'loss': '1.3129', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 67/100

 Train: {'loss': '1.2823', 'auc': '0.4350', 'accuracy': '0.8513', 'precision': '0.04
17', 'recall': '0.0417', 'f1': '0.0417', 'mcc': '-0.0389'}
 Val:   {'loss': '1.3111', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 68/100

 Train: {'loss': '1.2798', 'auc': '0.4866', 'accuracy': '0.8534', 'precision': '0.05
56', 'recall': '0.0556', 'f1': '0.0556', 'mcc': '-0.0239'}
 Val:   {'loss': '1.3110', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 69/100

 Train: {'loss': '1.2823', 'auc': '0.4014', 'accuracy': '0.8534', 'precision': '0.05
56', 'recall': '0.0556', 'f1': '0.0556', 'mcc': '-0.0239'}
 Val:   {'loss': '1.3112', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 70/100

 Train: {'loss': '1.2832', 'auc': '0.4559', 'accuracy': '0.8556', 'precision': '0.06
94', 'recall': '0.0694', 'f1': '0.0694', 'mcc': '-0.0088'}
 Val:   {'loss': '1.3113', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 71/100

 Train: {'loss': '1.2833', 'auc': '0.4892', 'accuracy': '0.8599', 'precision': '0.09
72', 'recall': '0.0972', 'f1': '0.0972', 'mcc': '0.0213'}
 Val:   {'loss': '1.3105', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 72/100

 Train: {'loss': '1.2830', 'auc': '0.4653', 'accuracy': '0.8534', 'precision': '0.05
56', 'recall': '0.0556', 'f1': '0.0556', 'mcc': '-0.0239'}
 Val:   {'loss': '1.3107', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 73/100
```

```
 Train: {'loss': '1.3154', 'auc': '0.4722', 'accuracy': '0.8578', 'precision': '0.08
33', 'recall': '0.0833', 'f1': '0.0833', 'mcc': '0.0062'}
 Val:   {'loss': '1.3115', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 74/100

 Train: {'loss': '1.2861', 'auc': '0.4657', 'accuracy': '0.8534', 'precision': '0.05
56', 'recall': '0.0556', 'f1': '0.0556', 'mcc': '-0.0239'}
 Val:   {'loss': '1.3109', 'auc': '0.5122', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 75/100

 Train: {'loss': '1.2879', 'auc': '0.4287', 'accuracy': '0.8491', 'precision': '0.02
78', 'recall': '0.0278', 'f1': '0.0278', 'mcc': '-0.0540'}
 Val:   {'loss': '1.3121', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 76/100

 Train: {'loss': '1.2835', 'auc': '0.4795', 'accuracy': '0.8513', 'precision': '0.04
17', 'recall': '0.0417', 'f1': '0.0417', 'mcc': '-0.0389'}
 Val:   {'loss': '1.3228', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 77/100

 Train: {'loss': '1.2856', 'auc': '0.4942', 'accuracy': '0.8513', 'precision': '0.04
17', 'recall': '0.0417', 'f1': '0.0417', 'mcc': '-0.0389'}
 Val:   {'loss': '1.3329', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 78/100

 Train: {'loss': '1.2852', 'auc': '0.4820', 'accuracy': '0.8534', 'precision': '0.05
56', 'recall': '0.0556', 'f1': '0.0556', 'mcc': '-0.0239'}
 Val:   {'loss': '1.3154', 'auc': '0.5241', 'accuracy': '0.6983', 'precision': '0.09
52', 'recall': '0.3158', 'f1': '0.1463', 'mcc': '0.0297'}

Epoch 79/100

 Train: {'loss': '1.2848', 'auc': '0.4750', 'accuracy': '0.8513', 'precision': '0.04
17', 'recall': '0.0417', 'f1': '0.0417', 'mcc': '-0.0389'}
 Val:   {'loss': '1.3117', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 80/100
```

```
 Train: {'loss': '1.2825', 'auc': '0.4499', 'accuracy': '0.8513', 'precision': '0.04
17', 'recall': '0.0417', 'f1': '0.0417', 'mcc': '-0.0389'}
 Val:   {'loss': '1.3114', 'auc': '0.4755', 'accuracy': '0.1250', 'precision': '0.07
80', 'recall': '0.8947', 'f1': '0.1435', 'mcc': '-0.0563'}

Epoch 81/100

 Train: {'loss': '1.2818', 'auc': '0.4867', 'accuracy': '0.8599', 'precision': '0.09
72', 'recall': '0.0972', 'f1': '0.0972', 'mcc': '0.0213'}
 Val:   {'loss': '1.3105', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 82/100

 Train: {'loss': '1.2828', 'auc': '0.4705', 'accuracy': '0.8556', 'precision': '0.06
94', 'recall': '0.0694', 'f1': '0.0694', 'mcc': '-0.0088'}
 Val:   {'loss': '1.3106', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 83/100

 Train: {'loss': '1.2830', 'auc': '0.4379', 'accuracy': '0.8491', 'precision': '0.02
78', 'recall': '0.0278', 'f1': '0.0278', 'mcc': '-0.0540'}
 Val:   {'loss': '1.3125', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 84/100

 Train: {'loss': '1.2817', 'auc': '0.4625', 'accuracy': '0.8556', 'precision': '0.06
94', 'recall': '0.0694', 'f1': '0.0694', 'mcc': '-0.0088'}
 Val:   {'loss': '1.3114', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 85/100

 Train: {'loss': '1.2813', 'auc': '0.4503', 'accuracy': '0.8513', 'precision': '0.04
17', 'recall': '0.0417', 'f1': '0.0417', 'mcc': '-0.0389'}
 Val:   {'loss': '1.3118', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 86/100

 Train: {'loss': '1.2817', 'auc': '0.4327', 'accuracy': '0.8470', 'precision': '0.01
39', 'recall': '0.0139', 'f1': '0.0139', 'mcc': '-0.0691'}
 Val:   {'loss': '1.3108', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 87/100

```

```
 Train: {'loss': '1.2821', 'auc': '0.4756', 'accuracy': '0.8578', 'precision': '0.08
33', 'recall': '0.0833', 'f1': '0.0833', 'mcc': '0.0062'}
 Val:   {'loss': '1.3105', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 88/100

 Train: {'loss': '1.2811', 'auc': '0.4478', 'accuracy': '0.8578', 'precision': '0.08
33', 'recall': '0.0833', 'f1': '0.0833', 'mcc': '0.0062'}
 Val:   {'loss': '1.3106', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 89/100

 Train: {'loss': '1.2829', 'auc': '0.4233', 'accuracy': '0.8491', 'precision': '0.02
78', 'recall': '0.0278', 'f1': '0.0278', 'mcc': '-0.0540'}
 Val:   {'loss': '1.3128', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 90/100

 Train: {'loss': '1.2809', 'auc': '0.4565', 'accuracy': '0.8556', 'precision': '0.06
94', 'recall': '0.0694', 'f1': '0.0694', 'mcc': '-0.0088'}
 Val:   {'loss': '1.3117', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 91/100

 Train: {'loss': '1.2807', 'auc': '0.4812', 'accuracy': '0.8491', 'precision': '0.02
78', 'recall': '0.0278', 'f1': '0.0278', 'mcc': '-0.0540'}
 Val:   {'loss': '1.3123', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 92/100

 Train: {'loss': '1.2819', 'auc': '0.4344', 'accuracy': '0.8578', 'precision': '0.08
33', 'recall': '0.0833', 'f1': '0.0833', 'mcc': '0.0062'}
 Val:   {'loss': '1.3117', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 93/100

 Train: {'loss': '1.2803', 'auc': '0.4377', 'accuracy': '0.8491', 'precision': '0.02
78', 'recall': '0.0278', 'f1': '0.0278', 'mcc': '-0.0540'}
 Val:   {'loss': '1.3117', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 94/100
```

```
 Train: {'loss': '1.2815', 'auc': '0.4545', 'accuracy': '0.8556', 'precision': '0.06
94', 'recall': '0.0694', 'f1': '0.0694', 'mcc': '-0.0088'}
 Val:   {'loss': '1.3121', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 95/100

 Train: {'loss': '1.2819', 'auc': '0.4373', 'accuracy': '0.8491', 'precision': '0.02
78', 'recall': '0.0278', 'f1': '0.0278', 'mcc': '-0.0540'}
 Val:   {'loss': '1.3112', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 96/100

 Train: {'loss': '1.2798', 'auc': '0.4708', 'accuracy': '0.8534', 'precision': '0.05
56', 'recall': '0.0556', 'f1': '0.0556', 'mcc': '-0.0239'}
 Val:   {'loss': '1.3116', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 97/100

 Train: {'loss': '1.2805', 'auc': '0.4699', 'accuracy': '0.8491', 'precision': '0.02
78', 'recall': '0.0278', 'f1': '0.0278', 'mcc': '-0.0540'}
 Val:   {'loss': '1.3124', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 98/100

 Train: {'loss': '1.2806', 'auc': '0.4391', 'accuracy': '0.8513', 'precision': '0.04
17', 'recall': '0.0417', 'f1': '0.0417', 'mcc': '-0.0389'}
 Val:   {'loss': '1.3115', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 99/100

 Train: {'loss': '1.2814', 'auc': '0.4303', 'accuracy': '0.8578', 'precision': '0.08
33', 'recall': '0.0833', 'f1': '0.0833', 'mcc': '0.0062'}
 Val:   {'loss': '1.3117', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}

Epoch 100/100

 Train: {'loss': '1.2809', 'auc': '0.4312', 'accuracy': '0.8513', 'precision': '0.04
17', 'recall': '0.0417', 'f1': '0.0417', 'mcc': '-0.0389'}
 Val:   {'loss': '1.3116', 'auc': '0.5000', 'accuracy': '0.0819', 'precision': '0.08
19', 'recall': '1.0000', 'f1': '0.1514', 'mcc': '0.0000'}
```


loss
Train loss
Val loss
auc
Train auc
Val auc
accuracy
Train accuracy
Val accuracy
precision
Train precision
Val precision
recall
Train recall
Val recall
f1
Train f1
Val f1
mcc
Train mcc
Val mcc

```
In [ ]:
```