In [1]:
```
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 re
import numpy as np
from sklearn.model_selection import train_test_split

# Set random seeds for reproducibility
torch.manual_seed(42)
np.random.seed(42)

# ─── DATASET (TEXT ONLY) ──────────────────────────────────────
class ClinicalNotesDataset(Dataset):
    def __init__(self, csv_path, tokenizer_name, max_length=512, mode='combined'):
        """
        Args:
            mode: 'text' (original note), 'combined' (all notes), or 'discharge' (d
        """
        self.data = pd.read_csv(csv_path)
        self.tokenizer = AutoTokenizer.from_pretrained(tokenizer_name)
        self.max_length = max_length
        self.mode = mode

        # Clean text during initialization
        self.data['cleaned_text'] = self.data['text_note'].apply(self._clean_text)
        self.data['cleaned_combined'] = self.data['combined_note'].apply(self._clea

    def _clean_text(self, text):
        """Remove clinical note artifacts and truncate intelligently"""
        text = str(text)
        # 1. Remove de-identification markers
        text = re.sub(r'\[\*\*.*?\*\*\]', '', text)
        # 2. Remove multiple newlines and whitespace
        text = re.sub(r'\s+', ' ', text).strip()
        # 3. Keep last 2048 characters (prioritize recent info)
        return text[-2048:] if len(text) > 2048 else text

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

    def __getitem__(self, idx):
        row = self.data.iloc[idx]

        # Select text based on mode
        if self.mode == 'text':
            text = row['cleaned_text']
        elif self.mode == 'discharge':
            text = self._extract_discharge_section(row['cleaned_combined'])
        else:  # combined
            text = row['cleaned_combined']
```

```python
        # Tokenize
        inputs = self.tokenizer(
            text,
            max_length=self.max_length,
            padding='max_length',
            truncation=True,
            return_tensors='pt'
        )

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

    def _extract_discharge_section(self, text):
        """Extract discharge summary section if available"""
        match = re.search(r'DISCHARGE SUMMARY:(.*?)(?=\n[A-Z]{2,}:|$)', text, re.IG
        return match.group(1).strip() if match else text

# ───── MODEL (BERT ONLY) ──────────────────────────────────────────────
class BERTMortalityPredictor(nn.Module):
    def __init__(self, model_name='emilyalsentzer/Bio_ClinicalBERT', dropout_rate=0
        super().__init__()
        self.bert = AutoModel.from_pretrained(model_name)
        self.dropout = nn.Dropout(dropout_rate)
        self.classifier = nn.Sequential(
            nn.Linear(768, 256),
            nn.ReLU(),
            nn.Dropout(dropout_rate),
            nn.Linear(256, 1)
        )

    def forward(self, input_ids, attention_mask):
        outputs = self.bert(input_ids=input_ids, attention_mask=attention_mask)
        pooled = outputs.last_hidden_state[:, 0, :]  # [CLS] token
        pooled = self.dropout(pooled)
        return self.classifier(pooled)

# ───── TRAINING UTILITIES ─────────────────────────────────────────────
def train_epoch(model, dataloader, device, optimizer, loss_fn):
    model.train()
    total_loss = 0
    pbar = tqdm(dataloader, desc="Training", leave=False)

    for batch in pbar:
        optimizer.zero_grad()

        inputs = batch['input_ids'].to(device)
        masks = batch['attention_mask'].to(device)
        labels = batch['label'].to(device).unsqueeze(1)

        outputs = model(inputs, masks)
        loss = loss_fn(outputs, labels)
```

```python
        loss.backward()
        optimizer.step()

        total_loss += loss.item()
        pbar.set_postfix({'loss': loss.item()})

    return total_loss / len(dataloader)

def evaluate(model, dataloader, device, threshold=None):
    model.eval()
    preds, labels = [], []

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

            outputs = model(inputs, masks).squeeze()
            probs = torch.sigmoid(outputs).cpu().numpy()
            preds.extend(probs)
            labels.extend(batch['label'].cpu().numpy())

    # Dynamic thresholding if not specified
    if threshold is None:
        threshold = np.percentile(preds, 100 * (1 - np.mean(labels)))

    bin_preds = (np.array(preds) > threshold).astype(int)

    print(f"\nEvaluation (Threshold={threshold:.3f})")
    print(f"AUC: {roc_auc_score(labels, preds):.4f}")
    print(f"F1: {f1_score(labels, bin_preds):.4f}")
    print(classification_report(labels, bin_preds))
    print("Confusion Matrix:")
    print(confusion_matrix(labels, bin_preds))

    return {
        'auc': roc_auc_score(labels, preds),
        'f1': f1_score(labels, bin_preds),
        'threshold': threshold
    }

# ──── MAIN EXPERIMENT ─────────────────────────────────────────
def run_experiment():
    # Config
    MODEL_NAME = "emilyalsentzer/Bio_ClinicalBERT"  # Clinical BERT variant
    MAX_LENGTH = 512
    BATCH_SIZE = 16
    EPOCHS = 10
    MODE = 'combined'  # 'text', 'combined', or 'discharge'

    # Setup
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print(f"Using device: {device}")

    # Data
    dataset = ClinicalNotesDataset("final.csv", MODEL_NAME, MAX_LENGTH, mode=MODE)
```

```python
    # Stratified split
    train_idx, val_idx = train_test_split(
        range(len(dataset)),
        test_size=0.2,
        stratify=dataset.data['mortality_label'],
        random_state=42
    )
    train_ds = torch.utils.data.Subset(dataset, train_idx)
    val_ds = torch.utils.data.Subset(dataset, val_idx)

    train_loader = DataLoader(train_ds, batch_size=BATCH_SIZE, shuffle=True, pin_me
    val_loader = DataLoader(val_ds, batch_size=BATCH_SIZE*2, shuffle=False, pin_mem

    # Model
    model = BERTMortalityPredictor(MODEL_NAME).to(device)

    # Handle class imbalance
    pos_weight = torch.tensor([
        (len(train_ds) - sum(dataset.data.iloc[train_idx]['mortality_label'])) /
        sum(dataset.data.iloc[train_idx]['mortality_label'])
    ]).to(device)
    loss_fn = nn.BCEWithLogitsLoss(pos_weight=pos_weight)

    # Optimizer
    optimizer = torch.optim.AdamW([
        {'params': model.bert.parameters(), 'lr': 2e-5},
        {'params': model.classifier.parameters(), 'lr': 1e-4}
    ], weight_decay=1e-4)

    # Training Loop
    best_f1 = 0
    for epoch in range(EPOCHS):
        print(f"\nEpoch {epoch+1}/{EPOCHS}")

        # Train
        train_loss = train_epoch(model, train_loader, device, optimizer, loss_fn)

        # Evaluate
        val_metrics = evaluate(model, val_loader, device)

        # Save best model
        if val_metrics['f1'] > best_f1:
            best_f1 = val_metrics['f1']
            torch.save(model.state_dict(), f"best_bert_{MODE}.pt")
            print(f"New best model saved (F1={best_f1:.4f})")

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

```
Using device: cuda

Epoch 1/10
```

```
Evaluation (Threshold=0.968)
AUC: 0.9982
F1: 0.8889
              precision    recall  f1-score   support

         0.0       0.99      0.99      0.99       214
         1.0       0.89      0.89      0.89        18

    accuracy                           0.98       232
   macro avg       0.94      0.94      0.94       232
weighted avg       0.98      0.98      0.98       232

Confusion Matrix:
[[212   2]
 [  2  16]]
New best model saved (F1=0.8889)

Epoch 2/10

Evaluation (Threshold=0.991)
AUC: 0.9995
F1: 0.9444
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       214
         1.0       0.94      0.94      0.94        18

    accuracy                           0.99       232
   macro avg       0.97      0.97      0.97       232
weighted avg       0.99      0.99      0.99       232

Confusion Matrix:
[[213   1]
 [  1  17]]
New best model saved (F1=0.9444)

Epoch 3/10
```

```
Evaluation (Threshold=0.184)
AUC: 1.0000
F1: 1.0000
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       214
         1.0       1.00      1.00      1.00        18

    accuracy                           1.00       232
   macro avg       1.00      1.00      1.00       232
weighted avg       1.00      1.00      1.00       232

Confusion Matrix:
[[214   0]
 [  0  18]]
New best model saved (F1=1.0000)

Epoch 4/10

Evaluation (Threshold=0.987)
AUC: 0.9987
F1: 0.8889
              precision    recall  f1-score   support

         0.0       0.99      0.99      0.99       214
         1.0       0.89      0.89      0.89        18

    accuracy                           0.98       232
   macro avg       0.94      0.94      0.94       232
weighted avg       0.98      0.98      0.98       232

Confusion Matrix:
[[212   2]
 [  2  16]]

Epoch 5/10

Evaluation (Threshold=0.991)
AUC: 1.0000
F1: 1.0000
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       214
         1.0       1.00      1.00      1.00        18

    accuracy                           1.00       232
   macro avg       1.00      1.00      1.00       232
weighted avg       1.00      1.00      1.00       232

Confusion Matrix:
[[214   0]
 [  0  18]]

Epoch 6/10
```

```
Evaluation (Threshold=0.947)
AUC: 1.0000
F1: 1.0000
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       214
         1.0       1.00      1.00      1.00        18

    accuracy                           1.00       232
   macro avg       1.00      1.00      1.00       232
weighted avg       1.00      1.00      1.00       232

Confusion Matrix:
[[214   0]
 [  0  18]]

Epoch 7/10
```

```
Evaluation (Threshold=0.976)
AUC: 1.0000
F1: 1.0000
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       214
         1.0       1.00      1.00      1.00        18

    accuracy                           1.00       232
   macro avg       1.00      1.00      1.00       232
weighted avg       1.00      1.00      1.00       232

Confusion Matrix:
[[214   0]
 [  0  18]]

Epoch 8/10
```

```
Evaluation (Threshold=0.412)
AUC: 1.0000
F1: 1.0000
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       214
         1.0       1.00      1.00      1.00        18

    accuracy                           1.00       232
   macro avg       1.00      1.00      1.00       232
weighted avg       1.00      1.00      1.00       232

Confusion Matrix:
[[214   0]
 [  0  18]]

Epoch 9/10
```

```
Evaluation (Threshold=0.555)
AUC: 1.0000
F1: 1.0000
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       214
         1.0       1.00      1.00      1.00        18

    accuracy                           1.00       232
   macro avg       1.00      1.00      1.00       232
weighted avg       1.00      1.00      1.00       232

Confusion Matrix:
[[214   0]
 [  0  18]]

Epoch 10/10
```

```
Evaluation (Threshold=0.606)
AUC: 1.0000
F1: 1.0000
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       214
         1.0       1.00      1.00      1.00        18

    accuracy                           1.00       232
   macro avg       1.00      1.00      1.00       232
weighted avg       1.00      1.00      1.00       232

Confusion Matrix:
[[214   0]
 [  0  18]]
```

In [ ]: