In [2]:

```
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, Dataset, random_split
from transformers import ByT5Tokenizer, ByT5Model
from sklearn.metrics import classification_report
import pandas as pd
from tqdm import tqdm
import os


class TextDataset(Dataset):
    def __init__(self, texts, labels, tokenizer, max_length=128):
        self.texts = texts
        self.labels = labels
        self.tokenizer = tokenizer
        self.max_length = max_length

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

    def __getitem__(self, idx):
        encoding = self.tokenizer(
            self.texts[idx],
            truncation=True,
            padding='max_length',
            max_length=self.max_length,
            return_tensors="pt"
        )
        return {
            'input_ids': encoding['input_ids'].squeeze(0),
            'attention_mask': encoding['attention_mask'].squeeze(0),
            'labels': torch.tensor(self.labels[idx], dtype=torch.long)
        }


class ByT5ForClassification(nn.Module):
    def __init__(self, num_labels=2):
        super(ByT5ForClassification, self).__init__()
        self.byt5 = ByT5Model.from_pretrained("google/byt5-small")
        self.dropout = nn.Dropout(0.1)
        self.classifier = nn.Linear(self.byt5.config.d_model, num_labels)

    def forward(self, input_ids, attention_mask):
        outputs = self.byt5(input_ids=input_ids, attention_mask=attention_mask)
        hidden_states = outputs.last_hidden_state
        pooled_output = hidden_states.mean(dim=1)
        pooled_output = self.dropout(pooled_output)
        logits = self.classifier(pooled_output)
        return logits


train_df = pd.read_csv('toxic_eng/train.csv')
test_df = pd.read_csv('toxic_eng/test.csv')

train_texts = train_df['comment_text'].tolist()
train_labels = train_df['toxic'].tolist()
test_texts = test_df['comment_text'].tolist()
test_labels = test_df['toxic'].tolist()
```

```
tokenizer = ByT5Tokenizer.from_pretrained("google/byt5-small")

MAX_LEN = 128
full_train_dataset = TextDataset(train_texts, train_labels, tokenizer, max_lengt
test_dataset = TextDataset(test_texts, test_labels, tokenizer, max_length=MAX_LE

train_size = int(0.9 * len(full_train_dataset))
val_size = len(full_train_dataset) - train_size
train_dataset, val_dataset = random_split(full_train_dataset, [train_size, val_s

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32)
test_loader = DataLoader(test_dataset, batch_size=32)

def save_checkpoint(model, optimizer, epoch, checkpoint_dir, model_name, best_va
    os.makedirs(checkpoint_dir, exist_ok=True)
    checkpoint_path = os.path.join(checkpoint_dir, f"{model_name}_epoch{epoch+1}
    torch.save({
        'epoch': epoch + 1,
        'model_state_dict': model.state_dict(),
        'optimizer_state_dict': optimizer.state_dict(),
        'best_val_loss': best_val_loss
    }, checkpoint_path)
    print(f"Checkpoint saved at: {checkpoint_path}")

def compute_val_loss(model, val_loader, criterion, device):
    model.eval()
    total_loss = 0
    with torch.no_grad():
        for batch in val_loader:
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            labels = batch['labels'].to(device)
            logits = model(input_ids=input_ids, attention_mask=attention_mask)
            loss = criterion(logits, labels)
            total_loss += loss.item()
    return total_loss / len(val_loader)

def load_best_model(model_class, checkpoint_dir, model_name, device):
    checkpoint_path = os.path.join(checkpoint_dir, f"{model_name}_best.pt")
    if os.path.exists(checkpoint_path):
        print(f"Loading best model from {checkpoint_path}")
        model = model_class().to(device)
        checkpoint = torch.load(checkpoint_path, map_location=device)
        model.load_state_dict(checkpoint['model_state_dict'])
        return model
    else:
        raise FileNotFoundError(f"Best checkpoint not found at {checkpoint_path}

def train_with_early_stopping(
    model, train_loader, val_loader, criterion, optimizer,
    device, num_epochs, checkpoint_dir, model_name,
    patience=3, resume_checkpoint_name=None
):
    best_val_loss = float('inf')
    patience_counter = 0
    start_epoch = 0
    if resume_checkpoint_name:
        resume_path = os.path.join(checkpoint_dir, resume_checkpoint_name)
        if os.path.exists(resume_path):
            print(f"Loading checkpoint from: {resume_path}")
            checkpoint = torch.load(resume_path, map_location=device)
            model.load_state_dict(checkpoint['model_state_dict'])
            optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
            best_val_loss = checkpoint.get('best_val_loss', float('inf'))
            start_epoch = checkpoint['epoch']
            print(f"Resumed from epoch {start_epoch}, best_val_loss={best_val_lo
        else:
            raise FileNotFoundError(f"Checkpoint {resume_path} not found!")

    for epoch in range(start_epoch, num_epochs):
        model.train()
        total_loss = 0
        correct = 0
        total = 0
        loop = tqdm(train_loader, desc=f"Epoch {epoch+1}/{num_epochs}")

        for batch in loop:
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            labels = batch['labels'].to(device)

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

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

            total_loss += loss.item()
            _, predicted = torch.max(logits, dim=1)
            correct += (predicted == labels).sum().item()
            total += labels.size(0)

            loop.set_postfix(loss=loss.item())

        train_accuracy = correct / total
        val_loss = compute_val_loss(model, val_loader, criterion, device)
        print(f"Epoch {epoch+1}, Train Loss: {total_loss:.4f}, Train Acc: {train

        save_checkpoint(model, optimizer, epoch, checkpoint_dir, model_name, bes

        if val_loss < best_val_loss:
            best_val_loss = val_loss
            patience_counter = 0
            best_path = os.path.join(checkpoint_dir, f"{model_name}_best.pt")
            torch.save({
                'epoch': epoch + 1,
                'model_state_dict': model.state_dict(),
                'optimizer_state_dict': optimizer.state_dict(),
                'best_val_loss': best_val_loss
            }, best_path)
            print(f"Best model saved at: {best_path}")
        else:
            patience_counter += 1
            print(f"Early stopping patience: {patience_counter}/{patience}")
            if patience_counter >= patience:
                print("Early stopping triggered.")
                break

def evaluate_byt5_model(model, test_loader, device):
    model.eval()
    correct = 0
    total = 0
    all_preds = []
    all_labels = []
    test_loader_tqdm = tqdm(test_loader, desc="Evaluating", leave=False)
    with torch.no_grad():
        for batch in test_loader_tqdm:
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            labels = batch['labels'].to(device)
            logits = model(input_ids=input_ids, attention_mask=attention_mask)
            _, preds = torch.max(logits, 1)
            correct += (preds == labels).sum().item()
            total += labels.size(0)
            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    accuracy = correct / total
    print(f"Test Accuracy: {accuracy:.4f}")
    print("\nClassification Report:\n")
    print(classification_report(all_labels, all_preds, digits=4))


device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = ByT5ForClassification(num_labels=2).to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=3e-5)
criterion = nn.CrossEntropyLoss()

# trenujeem model
train_with_early_stopping(
    model, train_loader, val_loader,
    criterion, optimizer,
    device, num_epochs=15,
    checkpoint_dir="checkpoints_byt5",
    model_name="byt5_toxicity",
    patience=2,
    resume_checkpoint_name="byt5_toxicity_epoch2.pt"
)

#nacitame najlespi model a spustime testovnaie
best_model = load_best_model(ByT5ForClassification, "checkpoints_byt5", "byt5_to
evaluate_byt5_model(best_model, test_loader, device)
```

```
You are using a model of type t5 to instantiate a model of type mt5. This is not
supported for all configurations of models and can yield errors.
Some weights of MT5ForSequenceClassification were not initialized from the model
checkpoint at google/byt5-small and are newly initialized: ['classification_head.
dense.bias', 'classification_head.dense.weight', 'classification_head.out_proj.bi
as', 'classification_head.out_proj.weight']
You should probably TRAIN this model on a down-stream task to be able to use it f
or predictions and inference.
Loading checkpoint from: checkpoints_byt5/byt5_toxicity_epoch2.pt
Resumed from epoch 2, best_val_loss=0.2709
```

```
Epoch 3/15: 100%|██████████| 3375/3375 [8:27:32<00:00,  9.02s/it, loss=0.148]
Epoch 3, Train Loss: 729.6052, Train Acc: 0.9146, Val Loss: 0.1766
Checkpoint saved at: checkpoints_byt5/byt5_toxicity_epoch3.pt
Best model saved at: checkpoints_byt5/byt5_toxicity_best.pt
```

```
IOPub message rate exceeded. 3240/3375 [8:05:20<20:14,  9.00s/it, loss=0.0716]
The Jupyter server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--ServerApp.iopub_msg_rate_limit`.

Current values:
ServerApp.iopub_msg_rate_limit=1000.0 (msgs/sec)
ServerApp.rate_limit_window=3.0 (secs)

Epoch 5/15: 100%|██████████| 3375/3375 [8:20:28<00:00,  8.90s/it, loss=0.0565]
```

```
Epoch 5, Train Loss: 573.4367, Train Acc: 0.9346, Val Loss: 0.1625
Checkpoint saved at: checkpoints_byt5/byt5_toxicity_epoch5.pt
Best model saved at: checkpoints_byt5/byt5_toxicity_best.pt
```

```
Epoch 6/15: 100%|██████████| 3375/3375 [8:39:49<00:00,  9.24s/it, loss=0.0901]
```

```
Epoch 6, Train Loss: 526.2422, Train Acc: 0.9400, Val Loss: 0.1572
Checkpoint saved at: checkpoints_byt5/byt5_toxicity_epoch6.pt
Best model saved at: checkpoints_byt5/byt5_toxicity_best.pt
```

```
IOPub message rate exceeded. 1863/3375 [4:49:33<4:02:10,  9.61s/it, loss=0.123]
The Jupyter server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--ServerApp.iopub_msg_rate_limit`.

Current values:
ServerApp.iopub_msg_rate_limit=1000.0 (msgs/sec)
ServerApp.rate_limit_window=3.0 (secs)

Epoch 8/15: 100%|██████████| 3375/3375 [8:35:22<00:00,  9.16s/it, loss=0.302]
```

```
Epoch 8, Train Loss: 439.5669, Train Acc: 0.9499, Val Loss: 0.1678
Checkpoint saved at: checkpoints_byt5/byt5_toxicity_epoch8.pt
Early stopping patience: 2/2
Early stopping triggered.
Loading best model from checkpoints_byt5/byt5_toxicity_best.pt
```

```
You are using a model of type t5 to instantiate a model of type mt5. This is not
supported for all configurations of models and can yield errors.
Some weights of MT5ForSequenceClassification were not initialized from the model
checkpoint at google/byt5-small and are newly initialized: ['classification_head.dense.bias', 'classification_head.dense.weight', 'classification_head.out_proj.bias', 'classification_head.out_proj.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```
Test Accuracy: 0.9343

Classification Report:

              precision    recall  f1-score   support

           0     0.9286    0.9408    0.9347     10000
           1     0.9400    0.9277    0.9338     10000

    accuracy                         0.9343     20000
   macro avg     0.9343    0.9343    0.9342     20000
weighted avg     0.9343    0.9343    0.9342     20000
```

In [ ]: