Data Scraping:
Done using an online scraper: https://apify.com/danek/steam-reviews

Game 1: Battlefield 2042

Data Cleaning

```
import pandas as pd
from langdetect import detect, DetectorFactory

# Ensure consistent language detection
DetectorFactory.seed = 0

# Load the full dataset
file_path = "/Users/nihar/Documents/WU/Thesis Data/Battlefield
2042/battlefield_scraped_reviews.xlsx"
df = pd.read_excel(file_path)

# Remove rows with empty or null reviews
df = df.dropna(subset=['review'])  # Drop if Review is NaN
df = df[df['review'].str.strip() != ""]  # Drop if Review is empty string

# Detect language
def detect_language(text):
  try:
    return detect(text)
  except:
    return "unknown"

df['Detected Language'] = df['review'].apply(detect_language)

# Keep only English reviews
df_english = df[df['Detected Language'] == 'en'].drop(columns=['Detected Language'])

# Save cleaned dataset
cleaned_path = "/Users/nihar/Documents/WU/Thesis Data/Battlefield
2042/battlefield_cleaned_reviews.xlsx"
df_english.to_excel(cleaned_path, index=False)
```

Data Sampling

```
import pandas as pd

# Load the cleaned dataset
```

```
file_path = "/Users/nihar/Documents/WU/Thesis Data/Battlefield
2042/battlefield_cleaned_reviews.xlsx"
df = pd.read_excel(file_path)

# Select 150 reviews with at least 50 funny votes
df_funny = df[df["votes_funny"] >= 50].sample(n=150, random_state=42)

# Select 250 random reviews from the remaining
df_remaining = df[~df.index.isin(df_funny.index)]
df_random = df_remaining.sample(n=250, random_state=42)

# Combine and shuffle
df_subset = pd.concat([df_funny, df_random]).sample(frac=1, random_state=42)

# Save the subset
subset_path = "/Users/nihar/Documents/WU/Thesis Data/Battlefield
2042/battlefield_400_reviews.xlsx"
df_subset.to_excel(subset_path, index=False)
```

Game 2: Barotrauma

Data Cleaning

```
import pandas as pd
from langdetect import detect, DetectorFactory

# Ensure consistent language detection
DetectorFactory.seed = 0

# Load the full dataset
file_path = "/Users/nihar/Documents/WU/Thesis
Data/Barotrauma/Barotrauma_scraped_reviews.xlsx"
df = pd.read_excel(file_path)

# Remove rows with empty or null reviews
df = df.dropna(subset=['review'])  # Drop if Review is NaN
df = df[df['review'].str.strip() != ""]  # Drop if Review is empty string

# Detect language
def detect_language(text):
    try:
        return detect(text)
    except:
        return "unknown"
```

```
df['Detected Language'] = df['review'].apply(detect_language)

# Keep only English reviews
df_english = df[df['Detected Language'] == 'en'].drop(columns=['Detected Language'])

# Save cleaned dataset
cleaned_path = "/Users/nihar/Documents/WU/Thesis
Data/Barotrauma/Barotrauma_cleaned_reviews.xlsx"
df_english.to_excel(cleaned_path, index=False)
```

Data Sampling

```
import pandas as pd

# Load the cleaned dataset
file_path = "/Users/nihar/Documents/WU/Thesis
Data/Barotrauma/Barotrauma_cleaned_reviews.xlsx"
df = pd.read_excel(file_path)

# Select 50 reviews with at least 50 funny votes
df_funny = df[df["votes_funny"] >= 50].sample(n=50, random_state=42)

# Select 350 random reviews from the remaining
df_remaining = df[~df.index.isin(df_funny.index)]
df_random = df_remaining.sample(n=350, random_state=42)

# Combine and shuffle
df_subset = pd.concat([df_funny, df_random]).sample(frac=1, random_state=42)

# Save the subset
subset_path = "/Users/nihar/Documents/WU/Thesis
Data/Barotrauma/Barotrauma_400_reviews.xlsx"
df_subset.to_excel(subset_path, index=False)
```

Building model for Game 1 (Battlefield 2042)

Data Preprocessing

```
import pandas as pd

# Step 1: Loading the Excel file
file_path = "/Users/nihar/Documents/WU/Thesis Data/Battlefield
2042/battlefield_400_reviews_marked.xlsx"
```

```
df = pd.read_excel(file_path)

# Step 2: Keeping only the relevant columns
df = df[['review', 'Sarcastic (True/False)']].copy()

# Step 3: Dropping empty reviews
df.dropna(subset=['review'], inplace=True)  # Removes rows where review is missing
df = df[df['review'].str.strip() != ""]     # Removes rows with blank strings (like "   ")

# Step 4: Renaming column for easier access
df.rename(columns={'Sarcastic (True/False)': 'label'}, inplace=True)

# Step 5: Light text cleaning
def clean_text(text):
  return text.strip()  # Remove extra spaces at start and end (not removing punctuation
because RoBERTa was trained on raw text with punctuation.)

df['review'] = df['review'].apply(clean_text)

# Step 6: Convert labels to 0/1
df['label'] = df['label'].astype(int)  # True → 1, False → 0

# Step 7: Preview the data
print(df.head())
print(df['label'].value_counts())  # Check how many sarcastic vs not sarcastic

df.to_csv("data/cleaned_battlefield_reviews.csv", index=False)
```

Tokenisation

```
import pickle
from transformers import RobertaTokenizer

# === 1. Load text data ===
with open("data/train_texts.pkl", "rb") as f:
  train_texts = pickle.load(f)
with open("data/test_texts.pkl", "rb") as f:
  test_texts = pickle.load(f)

# === 2. Load the RoBERTa tokenizer ===
tokenizer = RobertaTokenizer.from_pretrained('roberta-base')

# === 3. Tokenize the reviews ===
train_encodings = tokenizer(
  train_texts,
  truncation=True,
  padding=True,
```

```
  return_tensors="pt"  # Return PyTorch tensors
)

test_encodings = tokenizer(
  test_texts,
  truncation=True,
  padding=True,
  return_tensors="pt"
)

# === 4. Save tokenized encodings ===
with open("data/train_encodings.pkl", "wb") as f:
  pickle.dump(train_encodings, f)
with open("data/test_encodings.pkl", "wb") as f:
  pickle.dump(test_encodings, f)
```

Pytorch

```
import torch
from torch.utils.data import Dataset, DataLoader
import pickle

# === 1. Load encoded inputs and labels ===
with open("data/train_encodings.pkl", "rb") as f:
  train_encodings = pickle.load(f)
with open("data/test_encodings.pkl", "rb") as f:
  test_encodings = pickle.load(f)
with open("data/train_labels.pkl", "rb") as f:
  train_labels = pickle.load(f)
with open("data/test_labels.pkl", "rb") as f:
  test_labels = pickle.load(f)

# === 2. Custom Dataset Class ===
class SarcasmDataset(Dataset):
  def __init__(self, encodings, labels):
    self.encodings = encodings
    self.labels = labels

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

  def __getitem__(self, idx):
    item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
    item["labels"] = torch.tensor(self.labels[idx])
    return item
```

```
# === 3. Create Dataset objects ===
train_dataset = SarcasmDataset(train_encodings, train_labels)
test_dataset = SarcasmDataset(test_encodings, test_labels)

# === 4. Wrap in DataLoader for batching ===
train_loader = DataLoader(train_dataset, batch_size=8, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=8)
```

Train-Test Split

```
import pandas as pd
from sklearn.model_selection import train_test_split
import pickle
import os

# === 1. Load cleaned data from Step 1 ===
df = pd.read_csv("data/cleaned_battlefield_reviews.csv")

# === 2. Extract text and label ===
texts = df['review'].tolist()
labels = df['label'].tolist()

# === 3. Train-test split ===
train_texts, test_texts, train_labels, test_labels = train_test_split(
  texts, labels,
  test_size=0.2,
  stratify=labels,
  random_state=42
)

# === 4. Save splits ===
os.makedirs("data", exist_ok=True)

with open("data/train_texts.pkl", "wb") as f:
  pickle.dump(train_texts, f)
with open("data/test_texts.pkl", "wb") as f:
  pickle.dump(test_texts, f)
with open("data/train_labels.pkl", "wb") as f:
  pickle.dump(train_labels, f)
with open("data/test_labels.pkl", "wb") as f:
  pickle.dump(test_labels, f)
```

Model Training - Base

```
import torch
from torch.utils.data import DataLoader
from transformers import RobertaForSequenceClassification, RobertaTokenizer,
get_scheduler
from torch.optim import AdamW
from tqdm import tqdm
import pickle

# === 1. Load Data ===
with open("data/train_encodings.pkl", "rb") as f:
  train_encodings = pickle.load(f)
with open("data/train_labels.pkl", "rb") as f:
  train_labels = pickle.load(f)
with open("data/test_encodings.pkl", "rb") as f:
  test_encodings = pickle.load(f)
with open("data/test_labels.pkl", "rb") as f:
  test_labels = pickle.load(f)

# === 2. Dataset class again ===
class SarcasmDataset(torch.utils.data.Dataset):
  def __init__(self, encodings, labels):
    self.encodings = encodings
    self.labels = labels

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

  def __getitem__(self, idx):
    item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
    item["labels"] = torch.tensor(self.labels[idx])
    return item

train_dataset = SarcasmDataset(train_encodings, train_labels)
test_dataset = SarcasmDataset(test_encodings, test_labels)

train_loader = DataLoader(train_dataset, batch_size=8, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=8)

# === 3. Load RoBERTa Model ===
model = RobertaForSequenceClassification.from_pretrained('roberta-base', num_labels=2)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

# === 4. Optimizer, Loss, Scheduler ===
optimizer = AdamW(model.parameters(), lr=2e-5)
num_epochs = 3
num_training_steps = num_epochs * len(train_loader)
lr_scheduler = get_scheduler(
```

```
    "linear", optimizer=optimizer,
    num_warmup_steps=0,
    num_training_steps=num_training_steps
)
loss_fn = torch.nn.CrossEntropyLoss()

# === 5. Training Loop ===
model.train()
for epoch in range(num_epochs):
    print(f"🔁 Epoch {epoch + 1}")
    loop = tqdm(train_loader, leave=True)
    for batch in loop:
        batch = {k: v.to(device) for k, v in batch.items()}
        outputs = model(**batch)
        loss = outputs.loss

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

        loop.set_description(f"Epoch {epoch + 1}")
        loop.set_postfix(loss=loss.item())
```

Saving Models

```
import os
from transformers import RobertaTokenizer
from transformers import RobertaForSequenceClassification

# === Load the trained model and tokenizer (make sure you're in the same project directory)
===
model = RobertaForSequenceClassification.from_pretrained("saved_model_game1")
tokenizer = RobertaTokenizer.from_pretrained("roberta-base")  # Reuse base tokenizer if not
saved earlier

# === Save both model and tokenizer ===
os.makedirs("saved_model_game1", exist_ok=True)
model.save_pretrained("saved_model_game1")
tokenizer.save_pretrained("saved_model_game1")
```

Model Evaluation - Base

```
import torch
from transformers import RobertaTokenizer, RobertaForSequenceClassification
```

```
from sklearn.metrics import classification_report
import pickle
import os

# === 1. Load test data ===
with open("data/test_encodings.pkl", "rb") as f:
  test_encodings = pickle.load(f)
with open("data/test_labels.pkl", "rb") as f:
  test_labels = pickle.load(f)

# === 2. Load tokenizer and model from saved folder ===
tokenizer = RobertaTokenizer.from_pretrained("saved_model_game1")
model = RobertaForSequenceClassification.from_pretrained("saved_model_game1")


device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
model.eval()

# === 3. Convert test encodings to tensors ===
input_ids = torch.tensor(test_encodings['input_ids']).to(device)
attention_mask = torch.tensor(test_encodings['attention_mask']).to(device)
labels = torch.tensor(test_labels).to(device)

# === 4. Run model predictions ===
with torch.no_grad():
  outputs = model(input_ids=input_ids, attention_mask=attention_mask)
  predictions = torch.argmax(outputs.logits, dim=1)

# === 5. Print evaluation results ===
print("📊 Classification Report:")
print(classification_report(labels.cpu().numpy(), predictions.cpu().numpy()))
```

Hyperparameter Fine-tuning

```
import os
import torch
from transformers import RobertaForSequenceClassification, RobertaTokenizer,
get_scheduler, set_seed
from torch.optim import AdamW
from sklearn.metrics import classification_report
import pickle
from torch.utils.data import DataLoader, TensorDataset
import torch.nn as nn
import numpy as np
from tqdm import tqdm
```

```
# === 1. Reproducibility ===
set_seed(42)

# === 2. Load training data ===
with open("data/train_encodings.pkl", "rb") as f:
  train_encodings = pickle.load(f)
with open("data/train_labels.pkl", "rb") as f:
  train_labels = pickle.load(f)

input_ids = torch.tensor(train_encodings["input_ids"])
attention_mask = torch.tensor(train_encodings["attention_mask"])
labels = torch.tensor(train_labels)

train_dataset = TensorDataset(input_ids, attention_mask, labels)

# === 3. Device setup ===
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# === 4. Hyperparameter space ===
learning_rates = [2e-5, 3e-5, 5e-5]
epoch_options = [3, 4]

# === 5. Handle class imbalance ===
label_counts = np.bincount(train_labels)
weights = 1. / label_counts
class_weights = torch.tensor(weights, dtype=torch.float).to(device)
loss_fn = nn.CrossEntropyLoss(weight=class_weights)

# === 6. Training and saving models ===
for lr in learning_rates:
  for num_epochs in epoch_options:
    print(f"\n🔧 Training with learning rate {lr}, epochs {num_epochs}")

    model = RobertaForSequenceClassification.from_pretrained("roberta-base",
num_labels=2)
    tokenizer = RobertaTokenizer.from_pretrained("roberta-base")
    model.to(device)
    model.train()

    optimizer = AdamW(model.parameters(), lr=lr)
    train_loader = DataLoader(train_dataset, batch_size=8, shuffle=True)
    total_steps = num_epochs * len(train_loader)
    scheduler = get_scheduler("linear", optimizer=optimizer, num_warmup_steps=0,
num_training_steps=total_steps)

    for epoch in range(num_epochs):
      loop = tqdm(train_loader, desc=f"Epoch {epoch + 1}")
```

```
        for batch in loop:
            b_input_ids, b_mask, b_labels = [x.to(device) for x in batch]

            outputs = model(input_ids=b_input_ids, attention_mask=b_mask)
            loss = loss_fn(outputs.logits, b_labels)
            loss.backward()

            optimizer.step()
            scheduler.step()
            optimizer.zero_grad()
            loop.set_postfix(loss=loss.item())

    # === 7. Evaluate on training set ===
    model.eval()
    preds, true = [], []
    with torch.no_grad():
        for batch in train_loader:
            b_input_ids, b_mask, b_labels = [x.to(device) for x in batch]
            outputs = model(input_ids=b_input_ids, attention_mask=b_mask)
            pred = torch.argmax(outputs.logits, dim=1)
            preds.extend(pred.cpu().numpy())
            true.extend(b_labels.cpu().numpy())

    print("📊 Evaluation on training set:")
    print(classification_report(true, preds, digits=3))

    # === 8. Save model and tokenizer ===
    model_dir = f"saved_models/roberta_lr{str(lr).replace('.', '')}_ep{num_epochs}"
    os.makedirs(model_dir, exist_ok=True)
    model.save_pretrained(model_dir)
    tokenizer.save_pretrained(model_dir)
```

Best Model Evaluation

```
import torch
from transformers import RobertaTokenizer, RobertaForSequenceClassification
from sklearn.metrics import classification_report
import pickle

# === 1. Load test data ===
with open("data/test_encodings.pkl", "rb") as f:
  test_encodings = pickle.load(f)
with open("data/test_labels.pkl", "rb") as f:
  test_labels = pickle.load(f)
```

```
# === 2. Load best model ===
model_path = "saved_models/roberta_lr2e-05_ep4"
tokenizer = RobertaTokenizer.from_pretrained(model_path)
model = RobertaForSequenceClassification.from_pretrained(model_path)

# === 3. Set device and evaluation mode ===
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
model.eval()

# === 4. Convert test data to tensors ===
input_ids = torch.tensor(test_encodings['input_ids']).to(device)
attention_mask = torch.tensor(test_encodings['attention_mask']).to(device)
labels = torch.tensor(test_labels).to(device)

# === 5. Run predictions ===
with torch.no_grad():
  outputs = model(input_ids=input_ids, attention_mask=attention_mask)
  predictions = torch.argmax(outputs.logits, dim=1)

# === 6. Print evaluation results ===
print("📊 Classification Report (Best Hyperparams):")
print(classification_report(labels.cpu().numpy(), predictions.cpu().numpy()))
```

Generalisability Check - Testing the best model on unseen data of Game 2

```
import pandas as pd
import torch
from transformers import RobertaTokenizer, RobertaForSequenceClassification
from sklearn.metrics import classification_report

# === 1. Load Data ===
file_path = "/Users/nihar/Documents/WU/Thesis
Data/Barotrauma/sample_400_reviews(sarcasm_labeled).xlsx"
df = pd.read_excel(file_path)

# Clean & prepare
df = df[['review', 'Sarcastic (True/False)']].dropna()
df = df[df['review'].str.strip() != ""]
df.rename(columns={'Sarcastic (True/False)': 'label'}, inplace=True)
df['label'] = df['label'].astype(int)

# === 2. Tokenize ===
tokenizer = RobertaTokenizer.from_pretrained("roberta-base")
encodings = tokenizer(df['review'].tolist(), padding=True, truncation=True,
return_tensors="pt")
```

```
# === 3. Load Fine-tuned Model ===
model_path = "saved_models/roberta_lr3e-05_ep4"  # Change if needed
model = RobertaForSequenceClassification.from_pretrained(model_path)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
model.eval()

# === 4. Prepare tensors ===
input_ids = encodings['input_ids'].to(device)
attention_mask = encodings['attention_mask'].to(device)
labels = torch.tensor(df['label'].tolist()).to(device)

# === 5. Run Predictions ===
with torch.no_grad():
  outputs = model(input_ids=input_ids, attention_mask=attention_mask)
  preds = torch.argmax(outputs.logits, dim=1)

# === 6. Evaluation ===
print("📊 Evaluation on Barotrauma Reviews:")
print(classification_report(labels.cpu().numpy(), preds.cpu().numpy()))

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# === 7. Confusion Matrix ===
cm = confusion_matrix(labels.cpu().numpy(), preds.cpu().numpy())
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Not Sarcastic",
"Sarcastic"])
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix – Barotrauma (Fine-tuned RoBERTa)")
plt.show()
```

Adding Features to the best model from Game 1

```
import pandas as pd
import os

# === 1. Load Excel File ===
file_path = "/Users/nihar/Documents/WU/Thesis Data/Battlefield
2042/battlefield_400_reviews_marked.xlsx"
df = pd.read_excel(file_path)
```

```
# === 2. Select Columns: review + sarcasm label + selected features ===
df = df[['review', 'Sarcastic (True/False)', 'author/last_played', 'author_num_games_owned',
       'author_num_reviews', 'author_playtime_at_review', 'author_playtime_forever',
       'received_for_free', 'steam_purchase', 'timestamp_created', 'recommended',
       'votes_funny', 'votes_helpful', 'weighted_vote_score',
'written_during_early_access']].copy()

# === 3. Drop Missing Reviews ===
df.dropna(subset=['review'], inplace=True)
df = df[df['review'].str.strip() != ""]

# === 4. Rename label column and convert to integer ===
df = df.rename(columns={'Sarcastic (True/False)': 'label'})
df['label'] = df['label'].astype(int)

# === 5. Save cleaned version ===
os.makedirs("data", exist_ok=True)
df.to_csv("data/cleaned_feature_data.csv", index=False)
```

Normalising and Tokenising Features

```
import pandas as pd
import torch
from transformers import RobertaTokenizer
from sklearn.preprocessing import MinMaxScaler
import pickle
import os

# === 1. Load cleaned data with features ===
df = pd.read_csv("data/cleaned_feature_data.csv")

# === 2. Load train/test splits ===
with open("data/train_texts.pkl", "rb") as f:
  train_texts = pickle.load(f)
with open("data/test_texts.pkl", "rb") as f:
  test_texts = pickle.load(f)
with open("data/train_labels.pkl", "rb") as f:
  train_labels = pickle.load(f)
with open("data/test_labels.pkl", "rb") as f:
  test_labels = pickle.load(f)

# === 3. Get feature columns ===
feature_cols = ['author/last_played', 'author_num_games_owned', 'author_num_reviews',
            'author_playtime_at_review', 'author_playtime_forever', 'received_for_free',
            'steam_purchase', 'timestamp_created', 'recommended', 'votes_funny',
            'votes_helpful', 'weighted_vote_score', 'written_during_early_access']
```

```
# === 4. Normalize features ===
scaler = MinMaxScaler()
df[feature_cols] = scaler.fit_transform(df[feature_cols])

# === 5. Match text to features ===
train_features = df[df['review'].isin(train_texts)][feature_cols].values
test_features = df[df['review'].isin(test_texts)][feature_cols].values

# === 6. Tokenize review texts ===
tokenizer = RobertaTokenizer.from_pretrained('roberta-base')
train_encodings = tokenizer(train_texts, truncation=True, padding=True, return_tensors="pt")
test_encodings = tokenizer(test_texts, truncation=True, padding=True, return_tensors="pt")

# === 7. Save tokenized data and features ===
os.makedirs("data", exist_ok=True)
with open("data/train_encodings_feat.pkl", "wb") as f:
  pickle.dump(train_encodings, f)
with open("data/test_encodings_feat.pkl", "wb") as f:
  pickle.dump(test_encodings, f)
with open("data/train_features.pkl", "wb") as f:
  pickle.dump(train_features, f)
with open("data/test_features.pkl", "wb") as f:
  pickle.dump(test_features, f)
```

Train-Test Split

```
import pandas as pd
from sklearn.model_selection import train_test_split
import pickle
import os

# === 1. Load cleaned CSV ===
df = pd.read_csv("data/cleaned_feature_data.csv")

# === 2. Extract text and label ===
texts = df['review'].tolist()
labels = df['label'].tolist()

# === 3. Train-test split (80-20, reproducible) ===
train_texts, test_texts, train_labels, test_labels = train_test_split(
  texts, labels,
  test_size=0.2,
  stratify=labels,
  random_state=42  # ensures reproducibility
)
```

```
# === 4. Save splits to data/ ===
os.makedirs("data", exist_ok=True)

with open("data/train_texts.pkl", "wb") as f:
  pickle.dump(train_texts, f)
with open("data/test_texts.pkl", "wb") as f:
  pickle.dump(test_texts, f)
with open("data/train_labels.pkl", "wb") as f:
  pickle.dump(train_labels, f)
with open("data/test_labels.pkl", "wb") as f:
  pickle.dump(test_labels, f)
```

Model Training

```
import torch
import torch.nn as nn
from torch.utils.data import DataLoader
from transformers import RobertaModel, get_scheduler
from torch.optim import AdamW
import pickle
import numpy as np
from tqdm import tqdm
import os
from sklearn.metrics import classification_report

# === 1. Load tokenized inputs, features and labels ===
with open("data/train_encodings_feat.pkl", "rb") as f:
  train_encodings = pickle.load(f)
with open("data/train_features.pkl", "rb") as f:
  train_features = pickle.load(f)
with open("data/train_labels.pkl", "rb") as f:
  train_labels = pickle.load(f)

input_ids = torch.tensor(train_encodings["input_ids"])
attention_mask = torch.tensor(train_encodings["attention_mask"])
features = torch.tensor(train_features, dtype=torch.float32)
labels = torch.tensor(train_labels)

# === 2. Define dataset class ===
class SarcasmWithFeaturesDataset(torch.utils.data.Dataset):
  def __init__(self, input_ids, attention_mask, features, labels):
    self.input_ids = input_ids
    self.attention_mask = attention_mask
    self.features = features
    self.labels = labels
```

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

    def __getitem__(self, idx):
        return self.input_ids[idx], self.attention_mask[idx], self.features[idx], self.labels[idx]

train_dataset = SarcasmWithFeaturesDataset(input_ids, attention_mask, features, labels)
train_loader = DataLoader(train_dataset, batch_size=8, shuffle=True)

# === 3. Define model class ===
class RobertaWithFeatures(nn.Module):
    def __init__(self, feature_dim):
        super().__init__()
        self.roberta = RobertaModel.from_pretrained("roberta-base")
        self.classifier = nn.Sequential(
            nn.Linear(self.roberta.config.hidden_size + feature_dim, 128),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(128, 2)
        )

    def forward(self, input_ids, attention_mask, features):
        outputs = self.roberta(input_ids=input_ids, attention_mask=attention_mask)
        cls_output = outputs.last_hidden_state[:, 0, :]
        combined = torch.cat((cls_output, features), dim=1)
        return self.classifier(combined)

# === 4. Training Setup ===
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = RobertaWithFeatures(feature_dim=features.shape[1]).to(device)
optimizer = AdamW(model.parameters(), lr=3e-5)
loss_fn = nn.CrossEntropyLoss()
num_epochs = 4
num_training_steps = num_epochs * len(train_loader)
scheduler = get_scheduler("linear", optimizer=optimizer, num_warmup_steps=0,
num_training_steps=num_training_steps)

# === 5. Train the model ===
model.train()
for epoch in range(num_epochs):
    print(f"🔁 Epoch {epoch + 1}")
    loop = tqdm(train_loader, desc=f"Epoch {epoch+1}")
    for batch in loop:
        input_ids, attention_mask, features, labels = [b.to(device) for b in batch]
        outputs = model(input_ids, attention_mask, features)
        loss = loss_fn(outputs, labels)
```

```
        loss.backward()
        optimizer.step()
        scheduler.step()
        optimizer.zero_grad()
        loop.set_postfix(loss=loss.item())

# === 6. Save model weights ===
os.makedirs("saved_model_game1_feat", exist_ok=True)
torch.save(model.state_dict(), "saved_model_game1_feat/model.pt")
print("✅ Feature-based model training complete and saved!")

# === 7. Evaluate on training set ===
model.eval()
all_preds = []
all_labels = []

with torch.no_grad():
  for batch in train_loader:
      input_ids, attention_mask, features, labels = [b.to(device) for b in batch]
      outputs = model(input_ids, attention_mask, features)
      preds = torch.argmax(outputs, dim=1)

      all_preds.extend(preds.cpu().numpy())
      all_labels.extend(labels.cpu().numpy())

print("📊 Evaluation on training set:")
print(classification_report(all_labels, all_preds, digits=3))
```

Model Testing

```
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from transformers import RobertaModel
from sklearn.metrics import classification_report
import pickle

# === 1. Load saved test data ===
with open("data/test_encodings_feat.pkl", "rb") as f:
  test_encodings = pickle.load(f)
with open("data/test_features.pkl", "rb") as f:
  test_features = pickle.load(f)
with open("data/test_labels.pkl", "rb") as f:
  test_labels = pickle.load(f)
```

```
input_ids = torch.tensor(test_encodings["input_ids"])
attention_mask = torch.tensor(test_encodings["attention_mask"])
features = torch.tensor(test_features, dtype=torch.float32)
labels = torch.tensor(test_labels)

# === 2. Dataset class ===
class SarcasmWithFeaturesDataset(Dataset):
  def __init__(self, input_ids, attention_mask, features, labels):
    self.input_ids = input_ids
    self.attention_mask = attention_mask
    self.features = features
    self.labels = labels

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

  def __getitem__(self, idx):
    return self.input_ids[idx], self.attention_mask[idx], self.features[idx], self.labels[idx]

test_dataset = SarcasmWithFeaturesDataset(input_ids, attention_mask, features, labels)
test_loader = DataLoader(test_dataset, batch_size=8)

# === 3. Define same model architecture ===
class RobertaWithFeatures(nn.Module):
  def __init__(self, feature_dim):
    super().__init__()
    self.roberta = RobertaModel.from_pretrained("roberta-base")
    self.classifier = nn.Sequential(
      nn.Linear(self.roberta.config.hidden_size + feature_dim, 128),
      nn.ReLU(),
      nn.Dropout(0.3),
      nn.Linear(128, 2)
    )

  def forward(self, input_ids, attention_mask, features):
    outputs = self.roberta(input_ids=input_ids, attention_mask=attention_mask)
    cls_output = outputs.last_hidden_state[:, 0, :]
    combined = torch.cat((cls_output, features), dim=1)
    return self.classifier(combined)

# === 4. Load saved model weights ===
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = RobertaWithFeatures(feature_dim=features.shape[1]).to(device)
model.load_state_dict(torch.load("saved_model_game1_feat/model.pt",
map_location=device))
model.eval()

# === 5. Evaluate ===
```

```
all_preds = []
all_labels = []

with torch.no_grad():
  for batch in test_loader:
      input_ids, attention_mask, features, labels = [b.to(device) for b in batch]
      outputs = model(input_ids, attention_mask, features)
      preds = torch.argmax(outputs, dim=1)

      all_preds.extend(preds.cpu().numpy())
      all_labels.extend(labels.cpu().numpy())

# === 6. Results ===
print("📊 Test Set Evaluation (Feature-based Model):")
print(classification_report(all_labels, all_preds))

# === 7. Confusion Matrix ===
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

cm = confusion_matrix(all_labels, all_preds)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Not Sarcastic",
"Sarcastic"])
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix – Feature-Based Model")
plt.show()
```

Helpfulness Classifier

```
import pandas as pd
import numpy as np
import os
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader, random_split
from transformers import RobertaTokenizer, RobertaModel, get_scheduler
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import classification_report
from torch.optim import AdamW
from tqdm import tqdm
import random
import numpy as np
import torch
```

```
# === Set Seed for Reproducibility ===
def set_seed(seed=42):
  random.seed(seed)
  np.random.seed(seed)
  torch.manual_seed(seed)
  torch.cuda.manual_seed_all(seed)
  torch.backends.cudnn.deterministic = True
  torch.backends.cudnn.benchmark = False

set_seed(42)



# === 1. Load and Filter Sarcastic Reviews with Helpfulness ===
file_path = "/Users/nihar/Documents/WU/Thesis Data/Battlefield
2042/battlefield_400_reviews_marked.xlsx"
df = pd.read_excel(file_path)
df = df[df['Sarcastic (True/False)'] == True]  # Only sarcastic reviews

df = df[['review', 'Helpful (If Sarcastic)', 'author/last_played', 'author_num_games_owned',
       'author_num_reviews', 'author_playtime_at_review', 'author_playtime_forever',
       'received_for_free', 'steam_purchase', 'timestamp_created', 'recommended',
       'votes_funny', 'votes_helpful', 'weighted_vote_score',
'written_during_early_access']].dropna()

# Rename and encode label
# Rename and encode label
df.rename(columns={'Helpful (If Sarcastic)': 'label'}, inplace=True)
df['label'] = df['label'].str.lower().map({'yes': 1, 'no': 0})

# === 2. Normalize numeric features ===
features = df.iloc[:, 2:].copy()
for col in features.select_dtypes(include=['bool']).columns:
  features[col] = features[col].astype(int)

scaler = MinMaxScaler()
features_scaled = scaler.fit_transform(features)

# === 3. Tokenize Text ===
tokenizer = RobertaTokenizer.from_pretrained("roberta-base")
encodings = tokenizer(df['review'].tolist(), padding=True, truncation=True,
return_tensors="pt")

# === 4. PyTorch Dataset ===
class HelpfulnessDataset(Dataset):
  def __init__(self, encodings, features, labels):
    self.input_ids = encodings['input_ids']
    self.attention_mask = encodings['attention_mask']
```

```
        self.features = torch.tensor(features, dtype=torch.float32)
        self.labels = torch.tensor(labels, dtype=torch.long)

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

    def __getitem__(self, idx):
        return self.input_ids[idx], self.attention_mask[idx], self.features[idx], self.labels[idx]

dataset = HelpfulnessDataset(encodings, features_scaled, df['label'].values)

# === 5. Train/Test Split ===
train_size = int(0.8 * len(dataset))
train_dataset, test_dataset = random_split(dataset, [train_size, len(dataset) - train_size],
generator=torch.Generator().manual_seed(42))

train_loader = DataLoader(train_dataset, batch_size=8, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=8)

# === 6. Model Definition ===
class RobertaWithFeatures(nn.Module):
    def __init__(self, feature_dim):
        super().__init__()
        self.roberta = RobertaModel.from_pretrained("roberta-base")
        self.classifier = nn.Sequential(
            nn.Linear(self.roberta.config.hidden_size + feature_dim, 128),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(128, 2)
        )

    def forward(self, input_ids, attention_mask, features):
        outputs = self.roberta(input_ids=input_ids, attention_mask=attention_mask)
        cls_output = outputs.last_hidden_state[:, 0, :]
        combined = torch.cat((cls_output, features), dim=1)
        return self.classifier(combined)

# === 7. Train the Model ===
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = RobertaWithFeatures(feature_dim=features_scaled.shape[1]).to(device)
optimizer = AdamW(model.parameters(), lr=3e-5)
loss_fn = nn.CrossEntropyLoss()
epochs = 4
total_steps = epochs * len(train_loader)
scheduler = get_scheduler("linear", optimizer=optimizer, num_warmup_steps=0,
num_training_steps=total_steps)

# Training
```

```
model.train()
for epoch in range(epochs):
  print(f"🔁 Epoch {epoch+1}")
  loop = tqdm(train_loader)
  for batch in loop:
      input_ids, attention_mask, features, labels = [b.to(device) for b in batch]
      outputs = model(input_ids, attention_mask, features)
      loss = loss_fn(outputs, labels)

      loss.backward()
      optimizer.step()
      scheduler.step()
      optimizer.zero_grad()
      loop.set_postfix(loss=loss.item())

# === Evaluation on Training Set ===
model.eval()
train_preds, train_labels_eval = [], []
with torch.no_grad():
  for batch in train_loader:
      input_ids, attention_mask, features, labels = [b.to(device) for b in batch]
      outputs = model(input_ids, attention_mask, features)
      preds = torch.argmax(outputs, dim=1)
      train_preds.extend(preds.cpu().numpy())
      train_labels_eval.extend(labels.cpu().numpy())

print("\n📊 Evaluation on Training Set:")
print(classification_report(train_labels_eval, train_preds, digits=3))

# === 8. Evaluate on Test Set ===
model.eval()
all_preds, all_labels = [], []
with torch.no_grad():
  for batch in test_loader:
      input_ids, attention_mask, features, labels = [b.to(device) for b in batch]
      outputs = model(input_ids, attention_mask, features)
      preds = torch.argmax(outputs, dim=1)
      all_preds.extend(preds.cpu().numpy())
      all_labels.extend(labels.cpu().numpy())

from sklearn.metrics import classification_report
report = classification_report(all_labels, all_preds, digits=3)
print("📊 Helpfulness Evaluation Report:")
print(report)
```

Combined Dataset Modelling

Combining Data

```
import pandas as pd
import os

# === Load both datasets ===
bf = pd.read_excel("/Users/nihar/Documents/WU/Thesis Data/Battlefield
2042/battlefield_400_reviews_marked.xlsx")
bt = pd.read_excel("/Users/nihar/Documents/WU/Thesis
Data/Barotrauma/sample_400_reviews(sarcasm_labeled).xlsx")

# === Keep only necessary columns ===
bf = bf[['review', 'Sarcastic (True/False)']].copy()
bt = bt[['review', 'Sarcastic (True/False)']].copy()

# === Rename column for consistency ===
bf.rename(columns={'Sarcastic (True/False)': 'label'}, inplace=True)
bt.rename(columns={'Sarcastic (True/False)': 'label'}, inplace=True)

# === Drop empty reviews ===
bf.dropna(subset=['review'], inplace=True)
bt.dropna(subset=['review'], inplace=True)

bf = bf[bf['review'].str.strip() != ""]
bt = bt[bt['review'].str.strip() != ""]

# === Convert labels to integers ===
bf['label'] = bf['label'].astype(int)
bt['label'] = bt['label'].astype(int)

# === Combine ===
combined = pd.concat([bf, bt], ignore_index=True)
os.makedirs("data", exist_ok=True)
combined.to_csv("data/combined_800_reviews.csv", index=False)
```

Model Training and Testing

```
import torch
from transformers import RobertaTokenizer, RobertaForSequenceClassification,
get_scheduler
from torch.utils.data import TensorDataset, DataLoader, random_split
```

```
from torch import nn
from torch.optim import AdamW
from sklearn.metrics import classification_report
import pandas as pd
import numpy as np
import os
from tqdm import tqdm
import random

# === 1. Set Seed for Reproducibility ===
def set_seed(seed=42):
  random.seed(seed)
  np.random.seed(seed)
  torch.manual_seed(seed)
  torch.cuda.manual_seed_all(seed)

set_seed(42)

# === 2. Load Dataset ===
df = pd.read_csv("data/combined_800_reviews.csv")
df['label'] = df['label'].astype(int)

# === 3. Tokenize ===
tokenizer = RobertaTokenizer.from_pretrained("roberta-base")
encodings = tokenizer(df['review'].tolist(), padding=True, truncation=True,
return_tensors="pt")
input_ids = encodings['input_ids']
attention_mask = encodings['attention_mask']
labels = torch.tensor(df['label'].values)

# === 4. Prepare Dataset and DataLoaders ===
dataset = TensorDataset(input_ids, attention_mask, labels)
train_size = int(0.8 * len(dataset))
train_dataset, test_dataset = random_split(dataset, [train_size, len(dataset) - train_size],
generator=torch.Generator().manual_seed(42))
train_loader = DataLoader(train_dataset, batch_size=8, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=8)

# === 5. Hyperparameters ===
learning_rates = [2e-5, 3e-5, 5e-5]
epochs_list = [3, 4]
dropouts = [0.1, 0.3, 0.5]

# === 6. Training + Evaluation Function ===
results = []

for lr in learning_rates:
  for ep in epochs_list:
```

```
    for drop in dropouts:
        print(f"\n🔧 Training: LR={lr}, Epochs={ep}, Dropout={drop}")
        model = RobertaForSequenceClassification.from_pretrained("roberta-base",
num_labels=2)
        model.classifier.dropout = nn.Dropout(drop)
        device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        model.to(device)

        optimizer = AdamW(model.parameters(), lr=lr)
        total_steps = ep * len(train_loader)
        scheduler = get_scheduler("linear", optimizer=optimizer, num_warmup_steps=0,
num_training_steps=total_steps)
        loss_fn = nn.CrossEntropyLoss()

        # === Training ===
        model.train()
        for epoch in range(ep):
            loop = tqdm(train_loader, desc=f"Epoch {epoch+1}/{ep}")
            for batch in loop:
                b_input_ids, b_mask, b_labels = [x.to(device) for x in batch]
                outputs = model(input_ids=b_input_ids, attention_mask=b_mask)
                loss = loss_fn(outputs.logits, b_labels)
                loss.backward()
                optimizer.step()
                scheduler.step()
                optimizer.zero_grad()
                loop.set_postfix(loss=loss.item())

        # === Evaluation ===
        model.eval()
        all_preds, all_labels = [], []
        with torch.no_grad():
            for batch in test_loader:
                b_input_ids, b_mask, b_labels = [x.to(device) for x in batch]
                outputs = model(input_ids=b_input_ids, attention_mask=b_mask)
                preds = torch.argmax(outputs.logits, dim=1)
                all_preds.extend(preds.cpu().numpy())
                all_labels.extend(b_labels.cpu().numpy())

        report = classification_report(all_labels, all_preds, output_dict=True)
        results.append({
            "learning_rate": lr,
            "epochs": ep,
            "dropout": drop,
            "accuracy": report["accuracy"],
            "precision (macro avg)": report["macro avg"]["precision"],
            "recall (macro avg)": report["macro avg"]["recall"],
            "f1-score (macro avg)": report["macro avg"]["f1-score"]
```

```
            })

            # === Save Model ===
            folder_name = f"saved_models_g2/roberta_lr{lr}_ep{ep}_drop{int(drop*10)}"
            os.makedirs(folder_name, exist_ok=True)
            model.save_pretrained(folder_name)
            tokenizer.save_pretrained(folder_name)

# === 7. Export Results ===
pd.DataFrame(results).to_csv("data/hyperparameter_tuning_results_g2.csv", index=False)
```

Confusion Matrix

```
import torch
from transformers import RobertaForSequenceClassification, RobertaTokenizer
from torch.utils.data import DataLoader, TensorDataset
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import pandas as pd

# === 1. Load test set from combined 800 reviews ===
df = pd.read_csv("data/combined_800_reviews.csv")
df = df.dropna(subset=["review"])
df['label'] = df['label'].astype(int)

# === 2. Tokenize ===
tokenizer = RobertaTokenizer.from_pretrained("roberta-base")
encodings = tokenizer(df['review'].tolist(), padding=True, truncation=True,
return_tensors="pt")
input_ids = encodings['input_ids']
attention_mask = encodings['attention_mask']
labels = torch.tensor(df['label'].values)

# === 3. Use same split logic (80/20) ===
dataset = TensorDataset(input_ids, attention_mask, labels)
train_size = int(0.8 * len(dataset))
test_dataset = torch.utils.data.Subset(dataset, range(train_size, len(dataset)))
test_loader = DataLoader(test_dataset, batch_size=8)

# === 4. Load fine-tuned model ===
model_path = "saved_models_g2/roberta_lr2e-05_ep4_drop5"
model = RobertaForSequenceClassification.from_pretrained(model_path)
model.eval()
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
```

```
# === 5. Predictions ===
all_preds, all_labels = [], []
with torch.no_grad():
  for batch in test_loader:
      input_ids, attention_mask, labels = [x.to(device) for x in batch]
      outputs = model(input_ids=input_ids, attention_mask=attention_mask)
      preds = torch.argmax(outputs.logits, dim=1)

      all_preds.extend(preds.cpu().numpy())
      all_labels.extend(labels.cpu().numpy())

# === 6. Confusion Matrix ===
cm = confusion_matrix(all_labels, all_preds)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Not Sarcastic",
"Sarcastic"])
disp.plot(cmap="Blues")
plt.title("Confusion Matrix - Best Fine-tuned Model (G2)")
plt.tight_layout()
plt.show()
```