University of Central Missouri
Department of Computer Science & Cybersecurity

CS5720 Neural network and Deep learning
Spring 2025

Home Assignment 5. (Cover Ch 11, 12)

Student name: TADIKONDA GNANDEEP

Submission Requirements:

- Total Points: 100
- Once finished your assignment push your source code to your repo (GitHub) and explain the work through the ReadMe file properly. Make sure you add your student info in the ReadMe file.
- Submit your GitHub link and video on the BB.
- Comment your code appropriately ***IMPORTANT.***
- Make a simple video about 2 to 3 minutes which includes demonstration of your home assignment and explanation of code snippets.

- Any submission after provided deadline is considered as a late submission.

1. GAN Architecture

Explain the adversarial process in GAN training. What are the goals of the generator and discriminator, and how do they improve through competition? Diagram of the GAN architecture showing the data flow and objectives of each component.

The Adversarial Process in Action:
The training proceeds in a cyclical manner:

1. Generator Tries to Fool the Discriminator: The generator takes random noise as input and transforms it into a synthetic data sample (e.g., a fake image, a fake piece of text). It aims to produce outputs that look realistic enough to fool the discriminator.
2. Discriminator Tries to Identify Real from Fake: The discriminator receives two types of inputs: real data samples from the training set and fake data samples generated by the generator. Its task is to correctly classify each input as either "real" or "fake."
3. Feedback and Improvement:
o The discriminator provides feedback to both the generator and itself through the loss function. If the discriminator correctly identifies a fake sample, it's penalized for not being fooled. If it incorrectly identifies a real sample as fake, it's also penalized. This feedback helps the discriminator learn to better distinguish between real and fake data.
o The generator also receives feedback based on the discriminator's performance. If the discriminator correctly identifies its output as fake, the generator is penalized. This encourages the generator to adjust its parameters to produce more realistic samples that are more likely to fool the discriminator in the future.

Goals and Improvement Through Competition:

- Generator's Goal: To minimize the probability that the discriminator can correctly classify its generated samples as fake. In essence, it wants to make its fake data distribution Pg(x) as close as possible to the real data distribution Pdata(x).
- Discriminator's Goal: To maximize the probability of correctly classifying both real samples as real and fake samples as fake. It aims to learn the boundary between the real and generated data distributions.

How They Improve Through Competition:

The adversarial nature of this process drives the improvement of both networks:

- The Discriminator's Improvement Forces the Generator to Become Better: As the discriminator becomes better at distinguishing real from fake, the generator is pushed to produce more realistic and nuanced samples to avoid detection. This leads to the generator learning finer details and the underlying structure of the real data.
- The Generator's Improvement Forces the Discriminator to Become Better: As the generator produces increasingly realistic fake samples, the discriminator has to become more sophisticated in its analysis to avoid being fooled. It learns to look for subtle differences and deeper statistical properties that distinguish real from fake data.

This continuous competition, ideally, leads to a state where the generator produces synthetic data that is virtually indistinguishable from real data, and the discriminator is no better than random chance (50% accuracy) at telling them apart.

Real Data: Real samples from the training dataset are fed into the discriminator.

Generator (G):

- Takes random noise as input.
- Passes it through its network to generate synthetic data (Fake Data).

Discriminator (D):

- Receives both real data and fake data as input.
- Its network processes the input and outputs a probability (between 0 and 1) indicating whether the input is real (close to 1) or fake (close to 0).

Objectives:

- The generator's objective is to produce fake data that the discriminator classifies as real (i.e., maximize the discriminator's error on fake samples).
- The discriminator's objective is to correctly classify real samples as real and fake samples as fake.

2. Ethics and AI Harm

Choose one of the following real-world AI harms discussed in Chapter 12:

- Representational harm
- Allocational harm
- Misinformation in generative AI

Describe a real or hypothetical application where this harm may occur. Then, suggest **two harm mitigation strategies** that could reduce its impact based on the lecture.

Real-World AI Harm: Representational Harm

Representational harm occurs when AI systems perpetuate or amplify negative stereotypes, demeaning portrayals, or the erasure of certain groups of people. This can happen through biased training data that reflects existing societal prejudices, leading the AI to generate outputs that reinforce these harmful representations. It can also occur when AI systems fail to accurately or fairly represent the diversity within a population.

Hypothetical Application: AI-Powered Personalized Education Platform

Imagine an AI-powered platform designed to personalize educational content for students. This platform analyzes a student's learning style, strengths, weaknesses, and interests to tailor lessons, recommend resources, and even generate personalized learning materials like practice problems and explanations.

Potential for Representational Harm:

Representational harm could creep into this system in several ways:

- Bias in Visual Content Generation: If the AI is trained on a dataset where certain professions (e.g., scientist, engineer, CEO) are overwhelmingly depicted by one gender or ethnicity, the platform might disproportionately generate images and examples reflecting these biases when illustrating these roles. This could subtly reinforce stereotypes and limit students' perceptions of who can succeed in these fields. For instance, when explaining a physics concept, the AI might consistently show male scientists, subconsciously conveying that science is a male domain.
- Stereotypical Characterizations in Story Generation: If the platform includes a feature that generates personalized stories or scenarios to make learning engaging, and if the underlying language models are trained on biased text data, the AI might generate stories where characters from certain ethnic or socioeconomic backgrounds are consistently portrayed in stereotypical or negative ways (e.g., as the "villain," less intelligent, or in subservient roles). This can negatively impact students' self-esteem and perpetuate harmful societal biases.
- Limited Representation in Recommended Historical Figures or Cultural Examples: The AI might recommend historical figures or cultural examples that predominantly feature individuals from dominant groups, inadvertently marginalizing the contributions and experiences of underrepresented communities. This can lead to a skewed understanding of history and culture and a feeling of exclusion for students from those communities.

Harm Mitigation Strategies:

Based on lecture concepts, here are two harm mitigation strategies that could be implemented in this AI-powered personalized education platform:

1. Curated and Balanced Data Collection and Augmentation:

- Strategy: Proactively curate the training data used for generating visual content, stories, and recommendations to ensure balanced representation across various demographic groups (gender, race, ethnicity, socioeconomic status, ability, etc.). This involves not just collecting diverse data but also critically evaluating existing datasets for potential biases and filtering out harmful content.
- Implementation: For visual content, actively seek out and utilize datasets that intentionally showcase diversity in professions and roles. Employ data augmentation techniques to create variations of existing images that feature individuals from different backgrounds in the same roles. For text generation, utilize datasets known for their balanced and inclusive language. Implement bias detection tools to identify and mitigate stereotypical patterns in the training data. For historical and cultural recommendations, establish clear guidelines for ensuring a wide range of representation and actively seek out resources that highlight the contributions of marginalized groups.

2. Human Oversight and Feedback Mechanisms with Diverse Perspectives:
- Strategy: Integrate human review and feedback loops throughout the development and deployment process, prioritizing diverse perspectives. This ensures that potential representational harms are identified and addressed before they significantly impact users.
- Implementation: Establish a diverse review board comprising educators, ethicists, and representatives from various communities to evaluate the AI's generated content for potential biases and harmful representations. Implement user feedback mechanisms that allow students, parents, and educators to flag instances of biased or stereotypical content. Regularly audit the AI's outputs and recommendations using bias detection metrics and qualitative analysis. Incorporate these feedback loops to continuously refine the AI models and the underlying data to reduce representational harm over time. This iterative process, guided by diverse human insights, can act as a crucial safeguard against perpetuating harmful stereotypes.

By implementing these mitigation strategies, the developers of the AI-powered personalized education platform can strive to create a tool that not only enhances learning but also promotes inclusivity and equitable representation, avoiding the pitfalls of representational harm.

3. Programming Task (Basic GAN Implementation)

Implement a simple GAN using PyTorch or TensorFlow to generate handwritten digits from the MNIST dataset.

Requirements:

- Generator and Discriminator architecture
- Training loop with alternating updates
- Show sample images at Epoch 0, 50, and 100

Deliverables:

- Generated image samples
- Screenshot or plots comparing losses of generator and discriminator over time

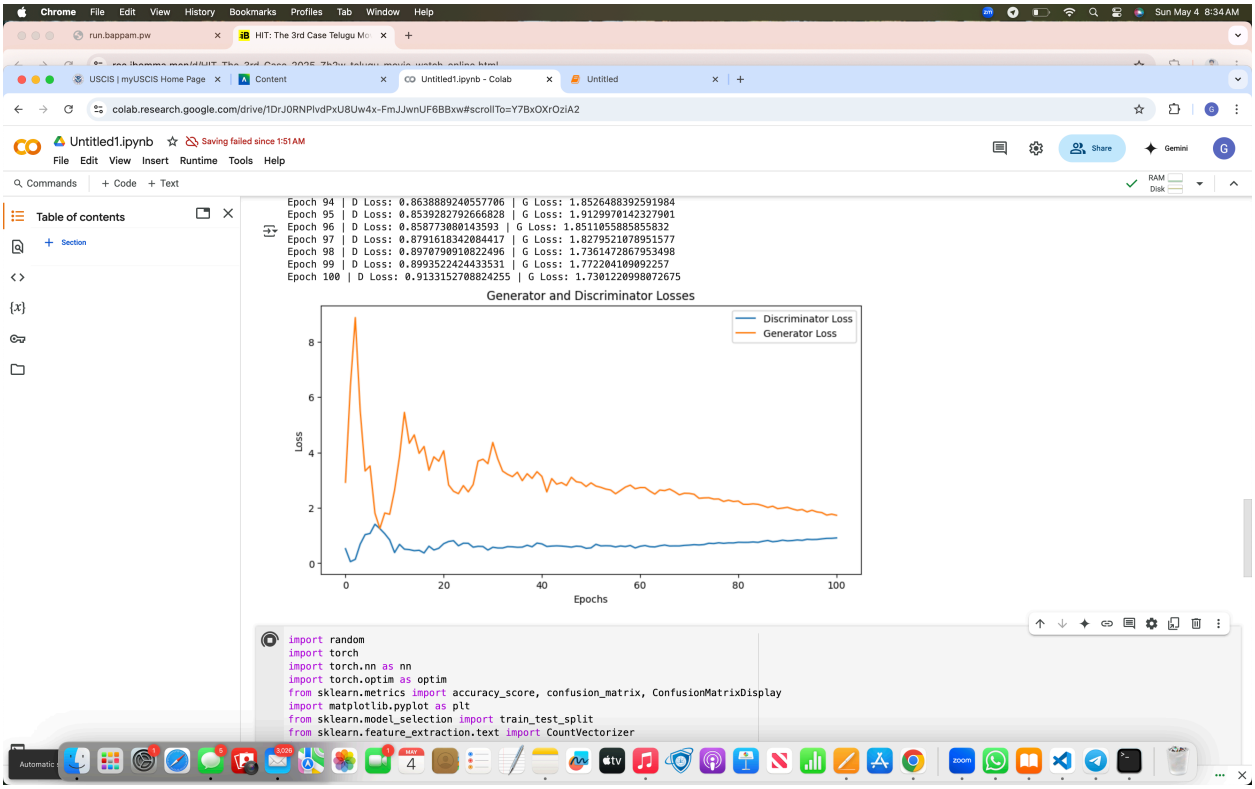
```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
from torchvision.utils import make_grid
import matplotlib.pyplot as plt
import numpy as np
import os

# Configuration
batch_size = 128
z_dim = 100
lr = 0.0002
epochs = 100
device = torch.device("cuda" if
torch.cuda.is_available() else "cpu")

# Data loading
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])
```

```
mnist = datasets.MNIST(root='./data', train=True,
download=True, transform=transform)
dataloader = DataLoader(mnist, batch_size=batch_size,
shuffle=True)

# Generator
class Generator(nn.Module):
    def __init__(self, z_dim):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(z_dim, 256),
            nn.ReLU(True),
            nn.Linear(256, 512),
            nn.ReLU(True),
            nn.Linear(512, 1024),
            nn.ReLU(True),
            nn.Linear(1024, 784),
            nn.Tanh()
        )

    def forward(self, z):
        return self.model(z).view(-1, 1, 28, 28)

# Discriminator
class Discriminator(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Flatten(),
            nn.Linear(784, 512),
            nn.LeakyReLU(0.2),
            nn.Linear(512, 256),
            nn.LeakyReLU(0.2),
            nn.Linear(256, 1),
            nn.Sigmoid()
        )

    def forward(self, img):
```

```
        return self.model(img)

# Initialize models
G = Generator(z_dim).to(device)
D = Discriminator().to(device)

# Loss and optimizers
criterion = nn.BCELoss()
optimizer_G = optim.Adam(G.parameters(), lr=lr)
optimizer_D = optim.Adam(D.parameters(), lr=lr)

# Track loss
losses_G, losses_D = [], []

# Fixed noise for image generation
fixed_noise = torch.randn(64, z_dim, device=device)

# Image saving function
def save_generated(epoch):
    G.eval()
    with torch.no_grad():
        fake = G(fixed_noise).detach().cpu()
    grid = make_grid(fake, nrow=8, normalize=True)
    plt.figure(figsize=(8, 8))
    plt.title(f"Generated Images at Epoch {epoch}")
    plt.imshow(np.transpose(grid, (1, 2, 0)))
    plt.axis('off')
    plt.savefig(f"gan_epoch_{epoch}.png")
    plt.close()
    G.train()

# Training loop
for epoch in range(epochs + 1):
    loss_G_epoch, loss_D_epoch = 0, 0
    for real_imgs, _ in dataloader:
        real_imgs = real_imgs.to(device)
        batch_size = real_imgs.size(0)
```

```
        real_labels = torch.ones(batch_size, 1, 
device=device)
        fake_labels = torch.zeros(batch_size, 1, 
device=device)

        # Train Discriminator
        z = torch.randn(batch_size, z_dim, 
device=device)
        fake_imgs = G(z).detach()

        real_loss = criterion(D(real_imgs), 
real_labels)
        fake_loss = criterion(D(fake_imgs), 
fake_labels)
        d_loss = real_loss + fake_loss

        optimizer_D.zero_grad()
        d_loss.backward()
        optimizer_D.step()

        # Train Generator
        z = torch.randn(batch_size, z_dim, 
device=device)
        generated_imgs = G(z)
        g_loss = criterion(D(generated_imgs), 
real_labels)

        optimizer_G.zero_grad()
        g_loss.backward()
        optimizer_G.step()

        loss_D_epoch += d_loss.item()
        loss_G_epoch += g_loss.item()

    losses_D.append(loss_D_epoch / len(dataloader))
    losses_G.append(loss_G_epoch / len(dataloader))
```

```
    print(f"Epoch {epoch} | D Loss: {loss_D_epoch /
len(dataloader)} | G Loss: {loss_G_epoch /
len(dataloader)}")

    # Save sample images at specific epochs
    if epoch == 0 or epoch == 50 or epoch == 100:
        save_generated(epoch)

# Plot loss curves
plt.figure(figsize=(10, 5))
plt.plot(range(epochs + 1), losses_D,
label='Discriminator Loss')
plt.plot(range(epochs + 1), losses_G, label='Generator
Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Generator and Discriminator Losses')
plt.savefig("loss_plot.png")
plt.show()
```



4. Programming Task (Data Poisoning Simulation)

Simulate a data poisoning attack on a sentiment classifier.
Start with a basic classifier trained on a small dataset (e.g., movie reviews). Then, poison some training data by flipping labels for phrases about a specific entity (e.g., "UC Berkeley").
Deliverables:

- Graphs showing accuracy and confusion matrix before and after poisoning
- How the poisoning affected results

```
import random
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.metrics import accuracy_score,
confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import
CountVectorizer
import numpy as np

# Simulated movie review dataset
data = [
    ("I loved the movie, it was fantastic!", 1),
    ("What a waste of time, boring and bad.", 0),
    ("The film was excellent and full of suspense.",
1),
    ("Horrible acting and weak plot.", 0),
```

```
    ("Absolutely brilliant direction and screenplay.",
1),
    ("Terrible, I hated every second.", 0),
    ("An outstanding performance by the lead actor.",
1),
    ("It was the worst movie I have seen.", 0),
    ("UC Berkeley is amazing and inspiring!", 1),
    ("UC Berkeley is awful and disappointing.", 0),
]

# Split data
texts, labels = zip(*data)
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(texts).toarray()
y = np.array(labels)

X_train, X_test, y_train, y_test = train_test_split(X,
y, test_size=0.3, random_state=42)

# Define a simple classifier
class SimpleSentimentClassifier(nn.Module):
    def __init__(self, input_dim):
        super().__init__()
        self.fc = nn.Linear(input_dim, 2)

    def forward(self, x):
        return self.fc(x)

# Training and evaluation
def train_and_evaluate(X_train, y_train, X_test,
y_test, input_dim, epochs=20):
    model = SimpleSentimentClassifier(input_dim)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=0.01)

    for epoch in range(epochs):
        model.train()
```

```
        inputs = torch.tensor(X_train,
dtype=torch.float32)
        targets = torch.tensor(y_train,
dtype=torch.long)
        outputs = model(inputs)
        loss = criterion(outputs, targets)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    model.eval()
    test_inputs = torch.tensor(X_test,
dtype=torch.float32)
    preds = model(test_inputs).argmax(dim=1).numpy()
    acc = accuracy_score(y_test, preds)
    cm = confusion_matrix(y_test, preds)
    return acc, cm, preds

# Baseline model
baseline_acc, baseline_cm, _ =
train_and_evaluate(X_train, y_train, X_test, y_test,
X.shape[1])

# Poison the data
poisoned_data = [
    (text, 0 if "UC Berkeley" in text and label == 1
else 1 if "UC Berkeley" in text and label == 0 else
label)
    for text, label in data
]
texts_poisoned, labels_poisoned = zip(*poisoned_data)
X_poisoned =
vectorizer.transform(texts_poisoned).toarray()
y_poisoned = np.array(labels_poisoned)
X_train_p, X_test_p, y_train_p, y_test_p =
train_test_split(X_poisoned, y_poisoned, test_size=0.3,
random_state=42)
```

```
# Retrain with poisoned data
poisoned_acc, poisoned_cm, _ =
train_and_evaluate(X_train_p, y_train_p, X_test_p,
y_test_p, X.shape[1])

# Plot the confusion matrices
fig, axs = plt.subplots(1, 2, figsize=(12, 5))
ConfusionMatrixDisplay(baseline_cm).plot(ax=axs[0])
axs[0].set_title(f"Before Poisoning\nAccuracy:
{baseline_acc:.2f}")
ConfusionMatrixDisplay(poisoned_cm).plot(ax=axs[1])
axs[1].set_title(f"After Poisoning\nAccuracy:
{poisoned_acc:.2f}")
plt.tight_la
```

5. Legal and Ethical Implications of GenAI

Discuss the legal and ethical concerns of AI-generated content based on the examples of:

- Memorizing private data (e.g., names in GPT-2)
- Generating copyrighted material (e.g., Harry Potter text)

Do you believe generative AI models should be restricted from certain data during training? Justify your answer.

Legal and Ethical Implications of Generative AI

Generative AI (GenAI) presents powerful capabilities—but also raises serious legal and ethical concerns. Two major areas of concern include privacy violations and copyright infringement.

1. Memorizing Private Data

Example: Early versions of GPT-2 were found to memorize and regurgitate snippets of private user data, including names, phone numbers, and email addresses scraped from the web.

Legal Concern:

- Violates data protection laws like the EU's GDPR or California's CCPA, which require that personal data not be used without consent.
- Breaches may result in legal liability for the model developers or platform hosts.

Ethical Concern:

- Users expect confidentiality when data is collected, even if publicly posted.
- There is a difference between information availability and intent to use—scraping does not imply ethical reuse.

2. Generating Copyrighted Material

Example: GPT models can generate entire paragraphs closely mimicking copyrighted books (e.g., Harry Potter), sometimes word-for-word if prompted specifically.

Legal Concern:

- Violates copyright law if outputs are substantially similar to protected texts.
- Even "transformative" use (a common legal defense) is questionable if the generation replicates large parts of the original work.

Ethical Concern:

- Undermines the rights of original creators and their ability to monetize work.
- If AI can replace or replicate creative labor without compensation or attribution, it poses long-term harm to the creative economy.

Justification:
1. Legal Compliance: Aligns with privacy and intellectual property laws globally.
2. Ethical AI Development: Protects individuals' rights and upholds the values of fairness and accountability.
3. Trust and Transparency: Encourages public confidence in AI systems and prevents misuse.
4. Risk Reduction: Reduces the likelihood of harmful outputs, lawsuits, and regulatory penalties.

Balanced Approach

While restrictions are essential, they shouldn't paralyze innovation. Key strategies include:

- Differential privacy during training
- Data filtering and deduplication
- Licensing agreements with data owners
- Transparency reports on training corporation.

6. Bias & Fairness Tools

Visit Aequitas Bias Audit Tool.
Choose a bias metric (e.g., false negative rate parity) and describe:

- What the metric measures
- Why it's important
- How a model might fail this metric

Optional: Try applying the tool to any small dataset or use demo data.
Bias Metric: False Negative Rate Parity

- What the metric measures: False Negative Rate Parity aims to ensure that the proportion of individuals who should have received a positive outcome (according to the ground truth) but were incorrectly denied (a false negative) is similar across different protected groups. In simpler terms, it checks if the model is

missing positive instances at roughly the same rate for everyone, regardless of their group affiliation (e.g., race, gender). The metric often compares the false negative rate of each protected group to a reference group (often the majority group). Perfect parity would mean the false negative rates are equal across all groups.

- Why it's important: This metric is crucial in applications where a "missed positive" can have significant negative consequences for the individual. For example:
  - Loan applications: A false negative means a creditworthy individual is denied a loan, potentially hindering their economic opportunities. If the false negative rate is significantly higher for a particular racial group, it indicates systemic bias in the lending process.
  - Hiring: A false negative means a qualified candidate is rejected for a job. If this rate is higher for women in STEM fields, it suggests the hiring process might be unfairly overlooking qualified female applicants.
  - Medical diagnosis: A false negative in a critical illness screening (e.g., cancer) means a person with the disease is not identified, potentially delaying treatment and worsening their prognosis. Disparities in this rate across demographic groups could have severe health equity implications.
  - Recidivism prediction: While the focus is often on false positives (incorrectly labeling someone as high-risk), a high false negative rate for a particular group could mean individuals who genuinely need support or intervention are being overlooked, potentially leading to negative outcomes for them and the community.
- How a model might fail this metric: A model might fail False Negative Rate Parity if its training data or its inherent algorithmic structure leads it to be less sensitive in identifying positive instances for certain protected groups compared to others. This can happen due to several reasons:
  - Imbalanced training data: If the training dataset has fewer positive examples for a particular group, the model might not learn to recognize the patterns associated with positive outcomes as effectively for that group.
  - Flawed or biased features: Certain features in the dataset might be correlated with the protected attribute and also spuriously correlated with the outcome. The model might learn to rely on these features in a way that disadvantages a particular group, leading to more false negatives. For instance, if certain indicators of creditworthiness are less prevalent or measured differently for a particular demographic due to systemic inequalities, the model might incorrectly classify creditworthy individuals from that group as low-risk.
  - Algorithmic bias: Some model architectures or optimization processes might inherently perform differently across different subgroups, leading to disparities in error rates, including the false negative rate.

o Proxy variables: If the dataset contains features that act as proxies for the protected attribute and are also correlated with the outcome, the model might inadvertently learn to discriminate based on these proxies, leading to different false negative rates.

Optional: Hypothetical Example

Imagine a hypothetical AI model used to predict which students are likely to succeed in a prestigious scholarship program. Let's say the protected group we're concerned about is "Socioeconomic Background" (categorized as "Low-Income" and "Higher-Income").

If the model has a False Negative Rate of 5% for "Higher-Income" students (meaning 5% of truly deserving higher-income students are incorrectly rejected) but a False Negative Rate of 15% for "Low-Income" students, then the model fails False Negative Rate Parity. This indicates that the model is disproportionately missing qualified low-income students, potentially due to factors like:

- The model might be relying heavily on extracurricular activities or resources more readily available to higher-income students as indicators of potential.
- The training data might have fewer examples of successful low-income students who overcame systemic barriers, leading the model to not recognize their potential as effectively.

This failure in False Negative Rate Parity would be a significant concern as it would perpetuate existing socioeconomic inequalities by unfairly denying opportunities to deserving low-income students.

Optional: Try applying the tool to any small dataset or use demo data.