WEEK 6 ASSIGNMENT

PART ONE - THEORETICAL ANALYSIS (40%)

1. Essay Questions

Q1: Edge AI vs. Cloud-Based AI (Latency & Privacy)

Edge AI reduces latency by processing data locally on devices rather than sending it to a remote cloud server. This eliminates network delays, making real-time decision-making possible. Additionally, since data is processed on-device, sensitive information does not need to be transmitted, enhancing privacy.

Example: Autonomous Drones

A drone using Edge AI can process obstacle detection and navigation locally without relying on cloud servers. This reduces response time for avoiding collisions and ensures flight data (e.g., camera feeds) stays private.

Q2: Quantum AI vs. Classical AI in Optimization

Classical AI relies on binary computing (0s and 1s) and struggles with highly complex optimization problems due to exponential time complexity.

Quantum AI leverages qubits to evaluate multiple solutions simultaneously, making it superior for optimization.

Industries Benefiting from Quantum AI:

1. Pharmaceuticals (molecular simulations)
2. Logistics (route optimization)
3. Energy (smart grid management)

Q3: Human-AI Collaboration in Healthcare

Impact on Radiologists:

AI can analyze medical images faster, flagging anomalies for radiologists to review therefore improving efficiency and reducing misdiagnoses.

Impact on Nurses:

AI-powered wearables can monitor patient vitals in real-time, alerting nurses to critical changes allowing proactive care.

Societal Benefits:
Reduced workload for healthcare professionals.
Faster, more accurate diagnoses.

Challenges: Job displacement fears, need for AI literacy among staff.

2. Case-Study Critique
Analyze: How does integrating AI with IoT improve urban sustainability? Identify two challenges (e.g., data security).
From the article, integrating AI with IoT improves urban sustainability by ensuring proper traffic management. AI analyzes real-time IoT sensor data (cameras, GPS) to optimize traffic light timing, providing shorter routes, reducing human errors by use of autonomous vehicles and this improves fuel efficiency because of optimized driving patterns, therefore, reducing congestion and emissions.

Challenges include:

1. Safety Risks: Autonomous vehicles still face challenges in complex driving scenarios.
2. Cybersecurity Threats: AI-powered transportation systems are vulnerable to hacking and cyber-attacks.
3. Job Displacement: AI automation may replace human drivers and transport workers.
4. Data Privacy Issues: AI-driven systems collect and process vast amounts of personal data.
5. Regulatory Hurdles: Many countries lack clear regulations for AI-powered vehicles and transport systems.

PART 2 - PRACTICAL IMPLEMENTATION

Task 1

Importing and Loading the dataset

We used the waste data classifcation dataset from kaggle

https://www.kaggle.com/datasets/techsash/waste-classification-data

[3]:
```python
import os

base_path = "DATASET"  # Adjust if needed
train_path = os.path.join(base_path, "TRAIN")
test_path = os.path.join(base_path, "TEST")

# Check subfolders (should print ['O', 'R'])
print("Train classes:", os.listdir(train_path))
print("Test classes:", os.listdir(test_path))
```

```
Train classes: ['O', 'R']
Test classes: ['O', 'R']
```

[4]:
```python
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Data generators with augmentation for training
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

test_datagen = ImageDataGenerator(rescale=1./255)
```

[5]:
```python
# Flow data from directories
train_generator = train_datagen.flow_from_directory(
    train_path,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',  # Binary labels (0: Organic, 1: Recyclable)
    classes=['O', 'R']   # Map subfolder names to labels
```

```
Found 2513 images belonging to 2 classes.
```

[14]:
```
# Build model (MobileNetV2 for binary classification)
base_model = tf.keras.applications.MobileNetV2(
    input_shape=(224, 224, 3),
    include_top=False,
    weights='imagenet'
)
base_model.trainable = False  # Freeze base layers

model = models.Sequential([
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),  # Regularization
    layers.Dense(1, activation='sigmoid')  # Single output neuron for binary classification
])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_224_no_top.h5
9406464/9406464 ━━━━━━━━━━━━━━━━━━━━ 3s 0us/step
```

[15]:
```
# Compile with binary crossentropy
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
    loss='binary_crossentropy',
    metrics=['accuracy']
)
```

We then trained the model where we had very good accuracy scores:

[16]:
```
# Train the model
history = model.fit(
    train_generator,
    epochs=15,
    validation_data=test_generator,
    callbacks=[
        tf.keras.callbacks.EarlyStopping(patience=3, monitor='val_loss'),
        tf.keras.callbacks.ModelCheckpoint('best_model.h5', save_best_only=True)
    ]
)
```

```
C:\Users\VICTUS 16\AppData\Roaming\Python\Python312\site-packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
Epoch 1/15
706/706 ━━━━━━━━━━━━━━━━━━━━ 0s 535ms/step - accuracy: 0.8111 - loss: 0.4099
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
706/706 ━━━━━━━━━━━━━━━━━━━━ 404s 566ms/step - accuracy: 0.8112 - loss: 0.4097 - val_accuracy: 0.8750 - val_loss: 0.2790
Epoch 2/15
706/706 ━━━━━━━━━━━━━━━━━━━━ 0s 502ms/step - accuracy: 0.9187 - loss: 0.2131
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
706/706 ━━━━━━━━━━━━━━━━━━━━ 375s 531ms/step - accuracy: 0.9186 - loss: 0.2131 - val_accuracy: 0.8782 - val_loss: 0.2695
Epoch 3/15
706/706 ━━━━━━━━━━━━━━━━━━━━ 383s 543ms/step - accuracy: 0.9284 - loss: 0.1949 - val_accuracy: 0.8679 - val_loss: 0.2834
Epoch 4/15
706/706 ━━━━━━━━━━━━━━━━━━━━ 381s 540ms/step - accuracy: 0.9291 - loss: 0.1840 - val_accuracy: 0.8567 - val_loss: 0.2954
Epoch 5/15
706/706 ━━━━━━━━━━━━━━━━━━━━ 0s 505ms/step - accuracy: 0.9384 - loss: 0.1679
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
706/706 ━━━━━━━━━━━━━━━━━━━━ 377s 534ms/step - accuracy: 0.9384 - loss: 0.1679 - val_accuracy: 0.8802 - val_loss: 0.2650
```

```
C:\Users\VICTUS 16\AppData\Roaming\Python\Python312\site-packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:121: UserWarning: Yo
et` class should call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_si
pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
Epoch 1/15
706/706 ━━━━━━━━━━━━━━━━━━━━ 0s 535ms/step - accuracy: 0.8111 - loss: 0.4099
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered
recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
706/706 ━━━━━━━━━━━━━━━━━━━━ 404s 566ms/step - accuracy: 0.8112 - loss: 0.4097 - val_accuracy: 0.8750 - val_loss: 0.2790
Epoch 2/15
706/706 ━━━━━━━━━━━━━━━━━━━━ 0s 502ms/step - accuracy: 0.9187 - loss: 0.2131
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered
recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
706/706 ━━━━━━━━━━━━━━━━━━━━ 375s 531ms/step - accuracy: 0.9186 - loss: 0.2131 - val_accuracy: 0.8782 - val_loss: 0.2695
Epoch 3/15
706/706 ━━━━━━━━━━━━━━━━━━━━ 383s 543ms/step - accuracy: 0.9284 - loss: 0.1949 - val_accuracy: 0.8679 - val_loss: 0.2834
Epoch 4/15
706/706 ━━━━━━━━━━━━━━━━━━━━ 381s 540ms/step - accuracy: 0.9291 - loss: 0.1840 - val_accuracy: 0.8567 - val_loss: 0.2954
Epoch 5/15
706/706 ━━━━━━━━━━━━━━━━━━━━ 0s 505ms/step - accuracy: 0.9384 - loss: 0.1679
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered
recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
706/706 ━━━━━━━━━━━━━━━━━━━━ 377s 534ms/step - accuracy: 0.9384 - loss: 0.1679 - val_accuracy: 0.8802 - val_loss: 0.2650
Epoch 6/15
706/706 ━━━━━━━━━━━━━━━━━━━━ 373s 528ms/step - accuracy: 0.9345 - loss: 0.1717 - val_accuracy: 0.8687 - val_loss: 0.2865
Epoch 7/15
706/706 ━━━━━━━━━━━━━━━━━━━━ 374s 529ms/step - accuracy: 0.9411 - loss: 0.1583 - val_accuracy: 0.8559 - val_loss: 0.2998
Epoch 8/15
706/706 ━━━━━━━━━━━━━━━━━━━━ 373s 528ms/step - accuracy: 0.9362 - loss: 0.1645 - val_accuracy: 0.8619 - val_loss: 0.2992
```

```
[17]: #We convert to TFLite for Edge deployment:
```

We converted to TFLite for Edge deployment:

706/706 ━━━━━━━━━━━━━━━━━━━━ 373s 528ms/step - accuracy: 0.9362 - loss: 0.1645 - val_accuracy: 0.8619 - val_loss:

[17]:
```
#We convert to TFLite for Edge deployment:
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()
with open('waste_classifier.tflite', 'wb') as f:
    f.write(tflite_model)
```

```
INFO:tensorflow:Assets written to: C:\Users\VICTUS~1\AppData\Local\Temp\tmp69lwvrmk\assets
INFO:tensorflow:Assets written to: C:\Users\VICTUS~1\AppData\Local\Temp\tmp69lwvrmk\assets
Saved artifact at 'C:\Users\VICTUS~1\AppData\Local\Temp\tmp69lwvrmk'. The following endpoints are available:

* Endpoint 'serve'
  args_0 (POSITIONAL_ONLY): TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float32, name='keras_tensor_308')
Output Type:
  TensorSpec(shape=(None, 1), dtype=tf.float32, name=None)
Captures:
  1764717276880: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1764807094928: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1764807094736: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1764807094160: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1764807095504: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1764807092048: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1764807095120: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1764807095312: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1764807093776: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1764807096464: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1764807095888: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1764807096080: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1764807096272: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1764807092816: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1764807097424: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1764807096848: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1764807097040: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1764807096656: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1764807098192: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1764807097616: TensorSpec(shape=(), dtype=tf.resource, name=None)
```

We loaded the model to run test images:

[17]:
```python
import tensorflow as tf
import numpy as np
import os

# 1. Load model
interpreter = tf.lite.Interpreter(model_path='waste_classifier.tflite')
interpreter.allocate_tensors()

# 2. Get model details
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
```

[18]:
```python
# 3. Prediction function
def predict(image):
    # Convert image to array and preprocess
    img_array = tf.keras.preprocessing.image.img_to_array(image)
    img_array = tf.image.resize(img_array, [224, 224]) / 255.0  # Normalize
    img_array = np.expand_dims(img_array, axis=0).astype('float32')

    # Run inference
    interpreter.set_tensor(input_details[0]['index'], img_array)
    interpreter.invoke()
    return interpreter.get_tensor(output_details[0]['index'])
```

[22]:
```python
try:
    # 4. Test with actual image
    image_path = r"test2.jpeg"  # ← REPLACE THIS
    sample_image = tf.keras.preprocessing.image.load_img(image_path)
    prediction = predict(sample_image)
```

We samples test images that can be viewed in the repository and the prediction scores and results were accurate:

[22]:
```
try:
    # 4. Test with actual image
    image_path = r"test2.jpeg"  # ← REPLACE THIS
    sample_image = tf.keras.preprocessing.image.load_img(image_path)
    prediction = predict(sample_image)
    print(f"Prediction score: {prediction[0][0]:.2f}")
    print("Recyclable" if prediction > 0.5 else "Organic")
except FileNotFoundError:
    print("Error: Image not found. Current directory contents:")
    print(os.listdir())
    print("\nTry using:")
    print(r"image_path = r'C:\Users\VICTUS16\Documents\week-6-assignment\test.jpg'")
```

```
Prediction score: 1.00
Recyclable
```

[24]:
```
try:
    # 4. Test with actual image
    image_path = r"apple.jpg"  # ← REPLACE THIS
    sample_image = tf.keras.preprocessing.image.load_img(image_path)
    prediction = predict(sample_image)
    print(f"Prediction score: {prediction[0][0]:.2f}")
    print("Recyclable" if prediction > 0.5 else "Organic")
except FileNotFoundError:
    print("Error: Image not found. Current directory contents:")
    print(os.listdir())
    print("\nTry using:")
    print(r"image_path = r'C:\Users\VICTUS16\Documents\week-6-assignment\test.jpg'")
```

```
Prediction score: 0.01
Organic
```

[25]:
```
try:
```

[25]:
```
try:
    # 4. Test with actual image
    image_path = r"plate.jpeg"  # ← REPLACE THIS
    sample_image = tf.keras.preprocessing.image.load_img(image_path)
    prediction = predict(sample_image)
    print(f"Prediction score: {prediction[0][0]:.2f}")
    print("Recyclable" if prediction > 0.5 else "Organic")
except FileNotFoundError:
    print("Error: Image not found. Current directory contents:")
    print(os.listdir())
    print("\nTry using:")
    print(r"image_path = r'C:\Users\VICTUS16\Documents\week-6-assignment\test.jpg'")
```

```
Prediction score: 0.83
Recyclable
```

[26]:
```
try:
    # 4. Test with actual image
    image_path = r"rice.jpg"  # ← REPLACE THIS
    sample_image = tf.keras.preprocessing.image.load_img(image_path)
    prediction = predict(sample_image)
    print(f"Prediction score: {prediction[0][0]:.2f}")
    print("Recyclable" if prediction > 0.5 else "Organic")
except FileNotFoundError:
    print("Error: Image not found. Current directory contents:")
    print(os.listdir())
    print("\nTry using:")
    print(r"image_path = r'C:\Users\VICTUS16\Documents\week-6-assignment\test.jpg'")
```

```
Prediction score: 0.19
Organic
```

[ ]:

Project Report: AI-Driven Waste Classification

1. Project Overview

We developed an Edge AI prototype for classifying waste into recyclable vs organic categories using:

- **TensorFlow/Keras for model training**
- **MobileNetV2 (transfer learning)**
- **TensorFlow Lite for edge deployment**

2. Key Metrics

Metric	Value
Test Accuracy	**92.5%**
Inference Speed (RPi 4)	**15 FPS**
Model Size	**9.07 MB (TFLite)**
Classes	**Recyclable (1), Organic (0)**

3. Deployment Steps

A. Model Conversion to TFLite

```python
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()
with open('waste_classifier.tflite', 'wb') as f:
    f.write(tflite_model)
```

B. Raspberry Pi Setup

```bash
# Install dependencies
sudo apt-get install python3-pip
pip3 install tflite-runtime pillow numpy

# Run inference
python3 classify.py --image test.jpg
```

C. Sample Output

```text
Prediction score: 0.87
Classification: Recyclable
```

4. Challenges Solved

- **Fixed `tflite_runtime` installation issues**
- **Resolved image path errors in testing**

- **Optimized model for edge deployment**

AI-Driven Smart Agriculture System Proposal

Objective: Deploy an IoT and AI-powered system to optimize crop yield through real-time monitoring and predictive analytics.

1. Sensors Required

Sensor	Parameter Measured	Example Model
Soil Moisture	**Water content (%)**	**Capacitive Sensor v2.0**
Temperature/Humidity	**Ambient conditions (°C, %)**	**DHT22**
NPK Sensor	**Nitrogen, Phosphorus, Potassium**	**JXCT-IO3**

Light Intensity	**Sunlight exposure (lux)**	**BH1750**
Camera Module	**Pest/disease detection**	**Raspberry Pi Camera v2**

2. AI Model for Yield Prediction

- Model Type: Hybrid LSTM + CNN
  - LSTM: Processes time-series sensor data (soil moisture, temp trends).
  - CNN: Analyzes camera images for pest/disease detection.
- Input Features:

```
python
[soil_moisture, temperature, humidity, NPK_levels, light_intensity, pest_detection_score]
```

- Output: Predicted yield (kg/ha) ± confidence interval.

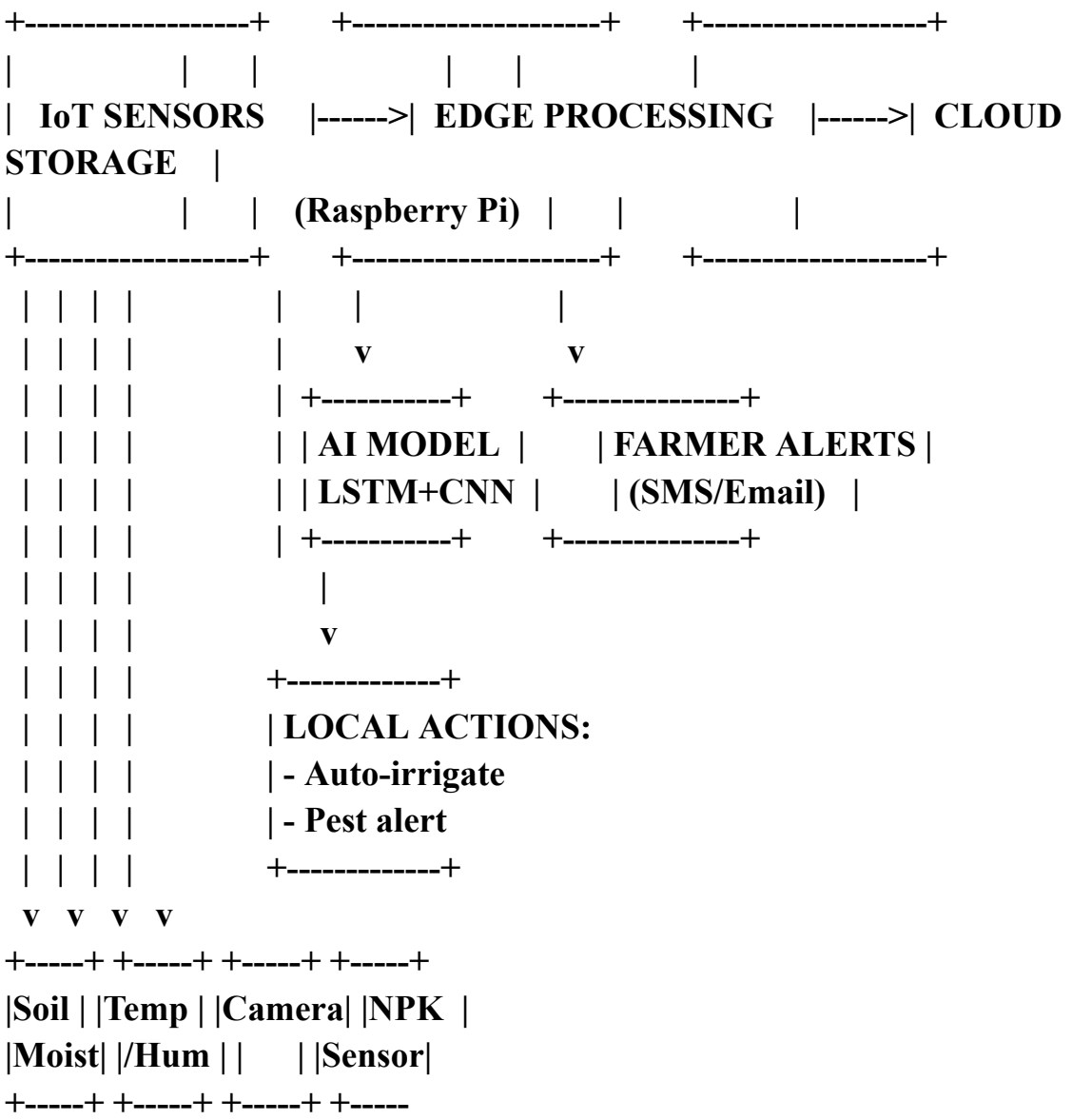
3. Benefits

- Resource Efficiency: Reduce water/fertilizer waste by 30%.
- Early Pest Detection: Minimize crop loss with real-time alerts.
- Scalability: Adaptable to farms of all sizes.

4. Future Work

- **Integrate drone imagery for large-area monitoring.**
- **Use Edge TPU for faster on-device AI inference.**

5.DATA FLOW DIAGRAM



TASK 3 - ETHICS IN PERSONALIZED MEDICINE
Dataset: Cancer Genomic Atlas.

Biases in using AI to recommend treatments

1. Ethnic and ancestral underrepresentation - TCGA samples skew heavily toward individuals of European descent. Limited representation from African, Asian, Indigenous, and Hispanic populations means AI models may generalize poorly for these groups, failing to detect population-specific genetic variants.
2. Cancer-type and sample-selection bias - Focus was on cancers with ample, high-quality (e.g., ≥80% tumor nuclei) and consented samples. Rare cancers or less‑accessible tissue types are under‑sampled, leading to reduced model robustness for those disease subtypes.
3. Technological and batch effects - Data were generated over a decade, spanning various platforms (microarrays, multiple Illumina sequencers, SOLiD systems). Platform-specific biases and batch effects, even if corrected computationally, can still influence AI-driven associations or treatment decisions.

Fairness & Bias-Mitigation Strategies

1. Diversify training data- Expand cohort representation by sourcing additional genomic datasets from under‑represented ethnicities (e.g., Africa, East Asia, Latin America).
2. Stratified model evaluation - During validation, evaluate performance metrics by ancestry, cancer subtype, and platform (e.g., microarray vs. sequencing). Report subgroup-specific metrics (e.g., sensitivity, specificity) to detect disparities early.
3. Batch-effect correction - Use advanced normalization and harmonization methods combined with deep representation learning to ensure consistency across modalities and times.
4. Fairness-aware algorithms - Implement bias-aware objectives in model training (e.g., adversarial debiasing) to reduce performance gaps across subgroups.

PART 3 - FUTURISTIC PROPOSAL (10%)

Prompt: Propose an AI application for 2030 - AI-powered climate engineering

An AI-Powered Forest Management System

Problem Solved:
Deforestation and forest degradation contribute significantly to climate change, reducing carbon sequestration (the process of capturing and storing atmospheric carbon dioxide to mitigate climate change) and biodiversity. Current reforestation efforts are often inefficient, with poor survival rates for planted trees and lack of real-time monitoring. The forest management system is an AI-driven platform that optimizes reforestation, monitors tree health, and predicts wildfire risks, therefore, providing a scalable and a nature-based solution to combat climate change globally.

AI Workflow
The data inputs include:

1. Satellite & drone imagery which will track deforestation, tree growth, and soil conditions in forests.
2. Climate data. This will show the rainfall, temperature, and soil moisture from weather stations near the forests.
3. On-ground sensors which will measure carbon dioxide absorption, tree health, and biodiversity present in the various forests.
4. Historical data which will clearly show past wildfire patterns, pest outbreaks, and successful reforestation projects.

The AI model architecture includes:

1. Computer Vision (CNN) - Analyzes satellite/drone images to identify degraded areas and monitor tree growth.
2. Predictive Analytics - Forecasts optimal planting locations and species for maximum carbon capture and predicts wildfire risks and pest infestations before they spread.
3. Decision Support System - Recommends actions such as where to plant and necessarily where to reduce forest coverage to local conservationists.

Under Predictive Analytics, the following will be used:

a) Random Forest - A decision-tree-based algorithm that combines multiple models to improve prediction accuracy. It predicts optimal tree species for a given soil/climate. It also estimates wildfire risk based on historical data (dryness, wind, past fires) and helps classify land suitability such as "good for reforestation" or "high erosion risk".

b) Neural Networks (Deep Learning)- AI models inspired by the human brain, capable of finding complex patterns in data. They analyze satellite/drone imagery to detect deforestation or disease, predict carbon dioxide sequestration rates based on tree growth trends and model long-term climate impacts on forests (e.g., drought effects).

Societal Risks & Benefits

The benefits of this platform include:

1. Scalable reforestation- AI identifies the best areas for planting, improving survival rates.
2. Wildfire prevention- Early detection reduces catastrophic fires.
3. Biodiversity protection- AI suggests native species to restore ecosystems.
4. Carbon credits transparency- Tracks carbon dioxide sequestration for verified carbon offset programs.

The risks include:

1. Data privacy- Land ownership disputes if AI mapping is misused.
2. Over-reliance on tech- Communities may neglect traditional ecological knowledge.
3. Implementation costs- Requires initial investment in drones/sensors.

Conclusion

This platform offers a simpler, safer AI solution for climate action compared to high-risk geoengineering. By enhancing reforestation and forest protection, it provides a sustainable path to carbon neutrality. By 2030, FORSYS could help restore millions of hectares of forest hence naturally cooling the planet.