```
from google.colab import drive

# Mount Google Drive
drive.mount('/content/drive')

# Verify the mounted path
!ls /content/drive/MyDrive
```

```
 IMT_2021093_BDBFS.gdoc
 Intern_Assignments
'Introduction to Machine Learning with Python ( PDFDrive ).pdf'
'Introduction to Mathematical Statistics (Robert V. Hogg, Allen Craig) (z-lib.org).gdoc'
'IPG.M.TECH_curriculum- fINAL 1 jULY  (1).pdf'
'IPG.M.TECH_curriculum- fINAL 1 jULY .pdf'
 Legal_Datasets.xlsx
 Letter.gdoc
'LLMs, Benchmarks, and Prompting.gdoc'
 logs
'MCom_Lab_Assg3 (1).pdf'
'MCom_Lab_Assg3 (2).pdf'
'MCom_Lab_Assg3 (3).pdf'
 MCom_Lab_Assg3.pdf
 messages.csv
'MessMenu-Jan-2025(BH-1)_Hindi.xlsx'
 ML_Amazon_Hack.gdoc
 Model_ckpt.h5
'My_Resume (1).pdf'
'My_Resume (2).pdf'
'My_Resume (3).pdf'
'My_Resume (5) (1).pdf'
'My_Resume (5).pdf'
'Nature and Man- An Insight into the Nexus.gdoc'
 OfferLetter_Shouvik.pdf
'Part II of Zero Shot Prompting versus Few Shots Prompting Techniques.gdoc'
 partnership_model.drawio.png
 PERS
'Practical Statistics for Data Scientists ( PDFDrive ).pdf'
'Python Data Science Handbook ( PDFDrive ).pdf'
 QuarkGluonBatches
 quark-gluon_data-set_n139306.hdf5
'Research Sources.txt'
'Saraswati Puja Chanda.gsheet'
 Screenshot_20230330-034019_GPay.jpg
 Screenshot_20230812-024511_Paytm.jpg
 Semantic_segmentation_dataset
 server.py
 Shell_Hackathon.gdoc
'Shouvik_Dey_Resume (1).pdf'
'Shouvik_Dey_Resume (2).pdf'
 Shouvik_Dey_Resume.pdf
 Snake_Image_Dataset
 train.csv
'trained model'
 untitled
'Untitled Diagram.drawio.png'
'Untitled document (1).gdoc'
'Untitled document (2).gdoc'
'Untitled document (3).gdoc'
'Untitled document (4).gdoc'
'Untitled document (5).gdoc'
'Untitled document (6).gdoc'
'Untitled document.gdoc'
'Untitled document.pdf'
'Untitled spreadsheet.gsheet'
 WBJEE
'Zero Shot Prompting versus Few Shots Prompting Techniques.gdoc'
```

```
!pip install h5py tensorflow numpy matplotlib
```

```
Requirement already satisfied: h5py in /usr/local/lib/python3.11/dist-packages (3.13.0)
Requirement already satisfied: tensorflow in /usr/local/lib/python3.11/dist-packages (2.18.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (2.0.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (25.2.10)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.6
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.4.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from tensorflow) (24.2)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<6.0.0dev,>=3.20.3 in /usr/local/lib/py
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.32.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (from tensorflow) (75.1.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.17.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.5.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (4.12.2)
```

```
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.17.2)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.71.0)
Requirement already satisfied: tensorboard<2.19,>=2.18 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.18.0)
Requirement already satisfied: keras>=3.5.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.8.0)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.4.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (4.56.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.11/dist-packages (from astunparse>=1.6.0->tensorflow) (0
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (0.0.8)
Requirement already satisfied: optree in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (0.14.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensor
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.10
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2.19,>=2.18->tensorflow
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2.19,>=2.18->tensorflow
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.11/dist-packages (from werkzeug>=1.0.1->tensorboard<2.19,
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0->tensorflow
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0->tensorfl
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich->keras>=3.5.0
```

```
import h5py

file_path = '/content/drive/MyDrive/quark-gluon_data-set_n139306.hdf5'

# Step 1: Check if the file can be opened
try:
    with h5py.File(file_path, 'r') as f:
        print("File opened successfully.")
except Exception as e:
    print("Error opening file:", e)

# Step 2: List top-level keys
try:
    with h5py.File(file_path, 'r') as f:
        print("Top-level keys:", list(f.keys()))
except Exception as e:
    print("Error listing keys:", e)

# Step 3: Try accessing a specific dataset
try:
    with h5py.File(file_path, 'r') as f:
        if 'X_jets' in f:
            print("Shape of X_jets:", f['X_jets'].shape)
        else:
            print("X_jets not found in the file.")
except Exception as e:
    print("Error accessing dataset:", e)
```

```
File opened successfully.
Top-level keys: ['X_jets', 'm0', 'pt', 'y']
Shape of X_jets: (139306, 125, 125, 3)
```

```
# !apt-get install h5utils
```

```
# !pip install --upgrade h5py
```

Analysis and Viz

```
# !h5repack /content/quark-gluon_data-set_n139306.hdf5 /content/quark-gluon_data-set_n139306_repacked.hdf5
```

```
import h5py
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Initialize variables to None to avoid NameError if they are not assigned in the try block
X_jets = None
m0 = None
pt = None
y = None
```

```
# Load and inspect the dataset with try-except handling
try:
    with h5py.File(file_path, 'r') as f:
        print("File opened successfully.")

        # Lazy loading by accessing only a slice of the data
        batch_size = 1000  # Load in batches of 1000
        X_jets = f['X_jets'][:batch_size]  # Load first batch
        m0 = f['m0'][:batch_size]
        pt = f['pt'][:batch_size]
        y = f['y'][:batch_size]

        print("\nDataset Overview (Partial Load):")
        print(f"X_jets shape: {X_jets.shape}")
        print(f"m0 shape: {m0.shape}")
        print(f"pt shape: {pt.shape}")
        print(f"y shape: {y.shape}")

except Exception as e:
    print("Error:", e)

print("\nDataset Overview:")
# Check if variables were assigned before printing their shapes
print(f"m0 (labels) shape: {m0.shape if m0 is not None else 'Not loaded'}")
print(f"pt (momentum) shape: {pt.shape if pt is not None else 'Not loaded'}")
print(f"y (rapidity) shape: {y.shape if y is not None else 'Not loaded'}")
print(f"X_jets shape: {X_jets.shape if X_jets is not None else 'Not loaded'}")
```

```
File opened successfully.

Dataset Overview (Partial Load):
X_jets shape: (1000, 125, 125, 3)
m0 shape: (1000,)
pt shape: (1000,)
y shape: (1000,)

Dataset Overview:
m0 (labels) shape: (1000,)
pt (momentum) shape: (1000,)
y (rapidity) shape: (1000,)
X_jets shape: (1000, 125, 125, 3)
```

```
# 1. Check the file integrity
# !h5stat /content/quark-gluon_data-set_n139306.hdf5
```

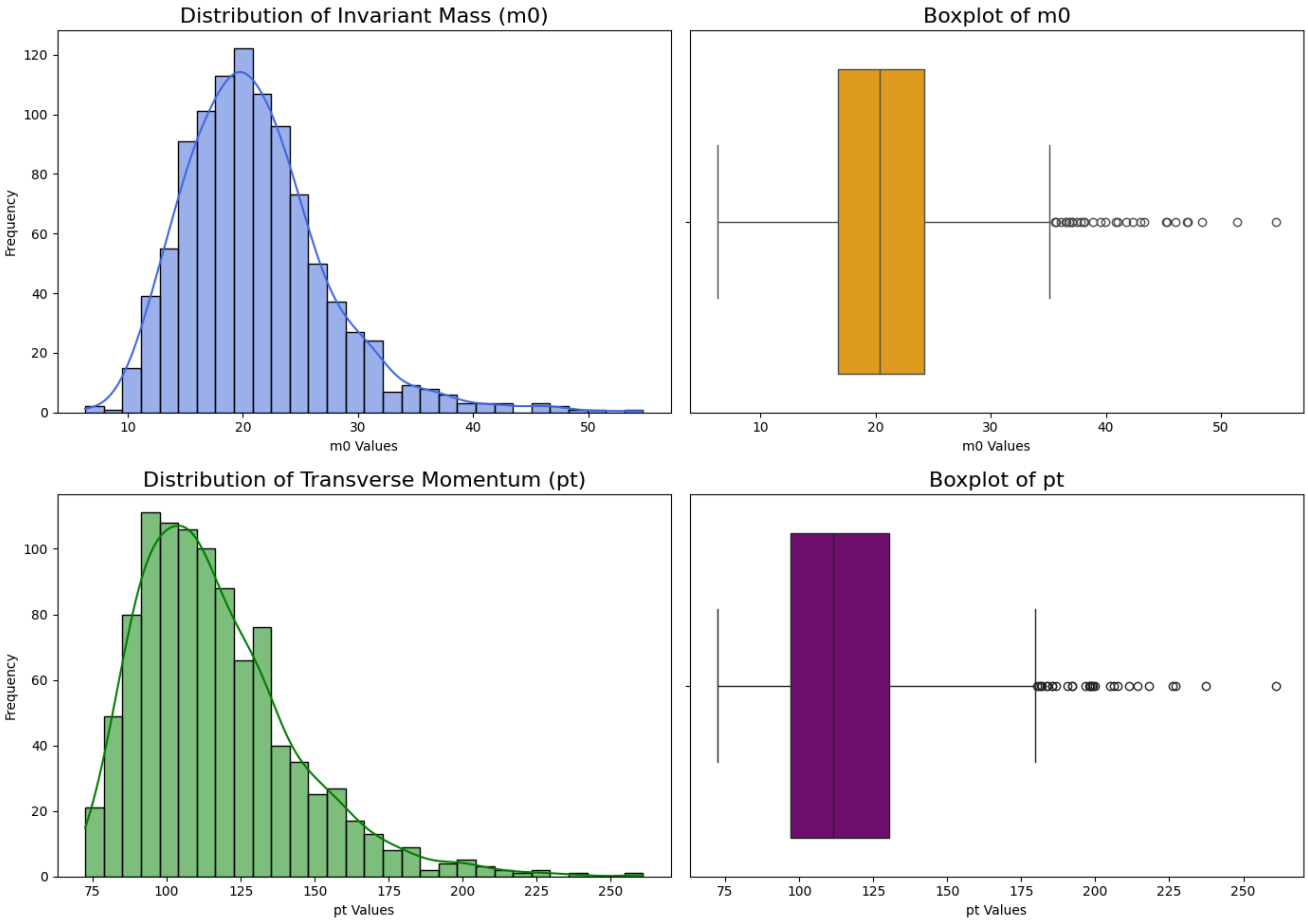
```
# !h5repack /content/quark-gluon_data-set_n139306.hdf5 /content/repaired_file.hdf5
```

```
# y
```

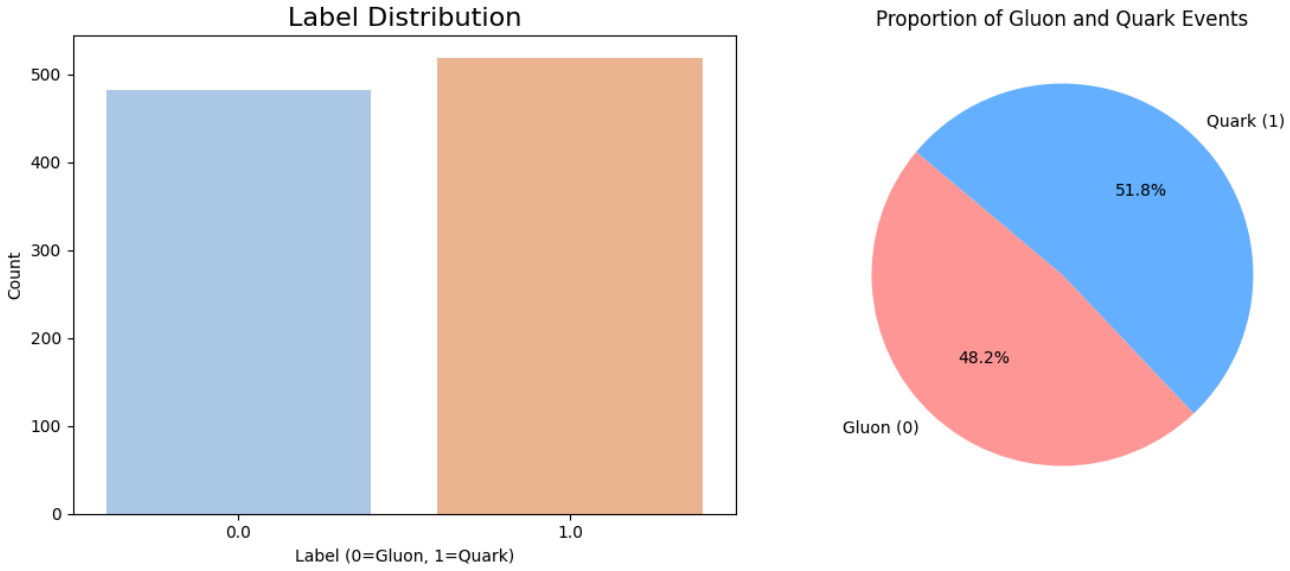
```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd

# ✅ Sample Data
# m0, pt, X_jets (1000, 125, 125, 3), and y
np.random.seed(42)  # For reproducibility

# 🔥 1. **Distribution of m0**
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Histogram with KDE
sns.histplot(m0, kde=True, color='royalblue', bins=30, ax=axes[0])
axes[0].set_title('Distribution of Invariant Mass (m0)', fontsize=16)
axes[0].set_xlabel('m0 Values')
axes[0].set_ylabel('Frequency')

# Boxplot
sns.boxplot(x=m0, color='orange', ax=axes[1])
axes[1].set_title('Boxplot of m0', fontsize=16)
axes[1].set_xlabel('m0 Values')

plt.tight_layout()
plt.show()
```

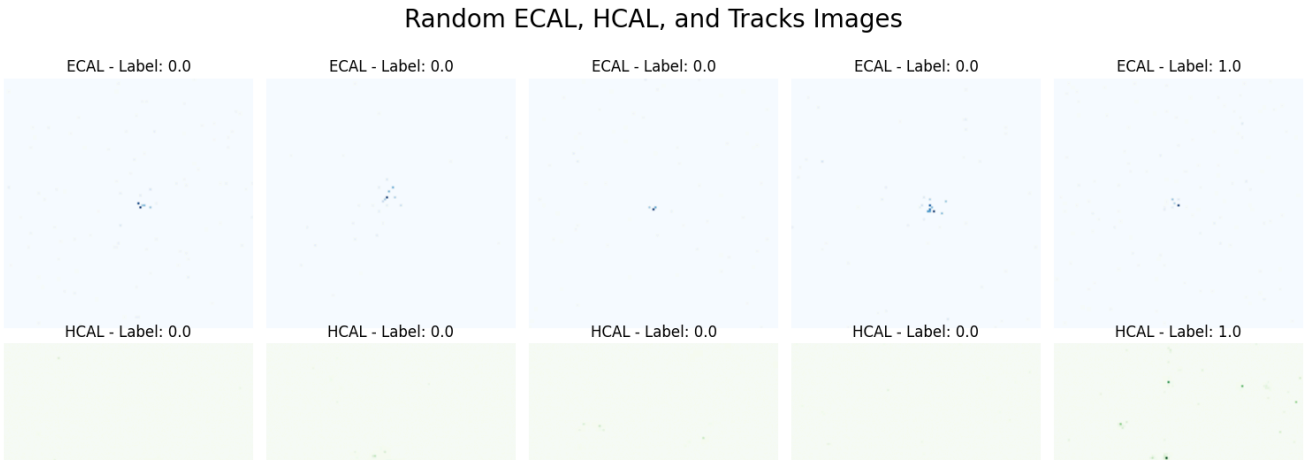
```
# 🔥 2. **Distribution of pt**
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Histogram with KDE
```

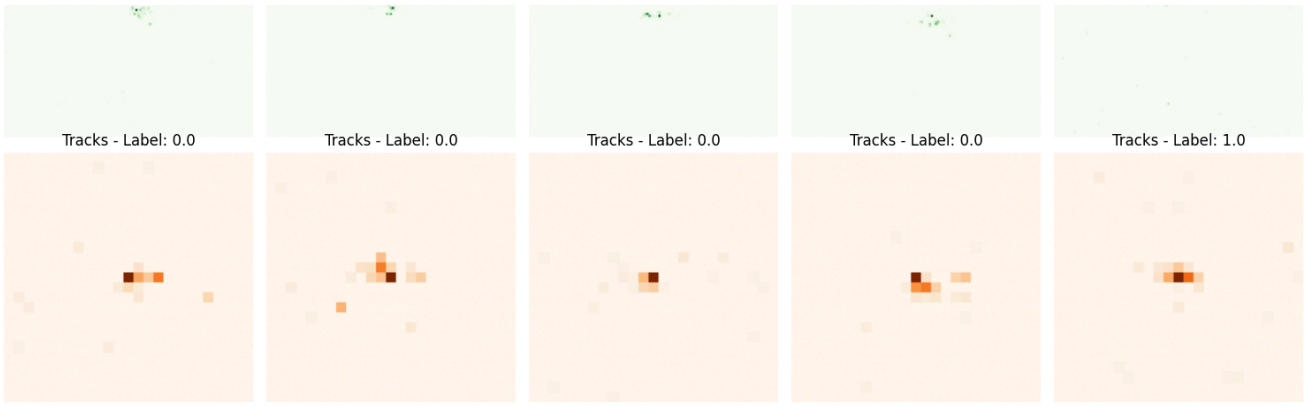
```
sns.histplot(pt, kde=True, color='green', bins=30, ax=axes[0])
axes[0].set_title('Distribution of Transverse Momentum (pt)', fontsize=16)
axes[0].set_xlabel('pt Values')
axes[0].set_ylabel('Frequency')

# Boxplot
sns.boxplot(x=pt, color='purple', ax=axes[1])
axes[1].set_title('Boxplot of pt', fontsize=16)
axes[1].set_xlabel('pt Values')

plt.tight_layout()
plt.show()


# 🔥 3. **Label Distribution (y)**
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Bar plot
sns.countplot(x=y, palette='pastel', ax=axes[0])
axes[0].set_title('Label Distribution', fontsize=16)
axes[0].set_xlabel('Label (0=Gluon, 1=Quark)')
axes[0].set_ylabel('Count')

# Pie chart
labels = ['Gluon (0)', 'Quark (1)']
sizes = [np.sum(y == 0), np.sum(y == 1)]
colors = ['#ff9999', '#66b3ff']
axes[1].pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140, colors=colors)
axes[1].set_title('Proportion of Gluon and Quark Events')

plt.tight_layout()
plt.show()


# 🔥 4. **Visualizing ECAL, HCAL, and Tracks**
fig, axes = plt.subplots(3, 5, figsize=(15, 10))
fig.suptitle('Random ECAL, HCAL, and Tracks Images', fontsize=20)

for i in range(5):
    idx = np.random.randint(0, len(X_jets))

    # ECAL (Channel 0)
    axes[0, i].imshow(X_jets[idx, :, :, 0], cmap='Blues')
    axes[0, i].set_title(f'ECAL - Label: {y[idx]}')
    axes[0, i].axis('off')

    # HCAL (Channel 1)
    axes[1, i].imshow(X_jets[idx, :, :, 1], cmap='Greens')
    axes[1, i].set_title(f'HCAL - Label: {y[idx]}')
    axes[1, i].axis('off')

    # Tracks (Channel 2)
    axes[2, i].imshow(X_jets[idx, :, :, 2], cmap='Oranges')
    axes[2, i].set_title(f'Tracks - Label: {y[idx]}')
    axes[2, i].axis('off')

plt.tight_layout(rect=[0, 0, 1, 0.97])
plt.show()


# 🔥 5. **Correlation Analysis**
# Create dataframe for correlation
data = {
    'm0': m0,
    'pt': pt,
    'y': y
}
df = pd.DataFrame(data)

# Heatmap of correlation matrix
plt.figure(figsize=(8, 6))
corr_matrix = df.corr()
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm', square=True)
plt.title('Correlation Matrix')
plt.show()
```

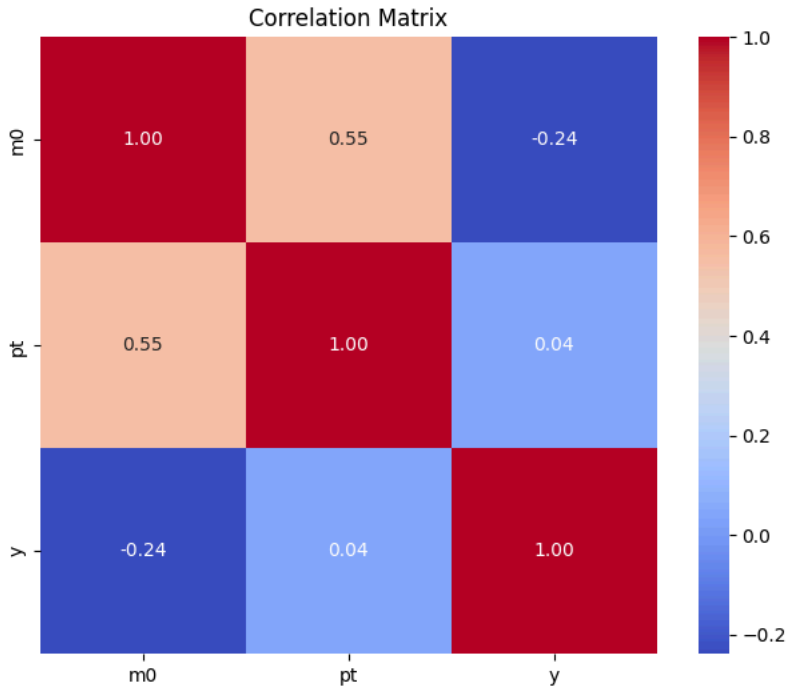



```
<ipython-input-9-47736b8d573b>:51: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.countplot(x=y, palette='pastel', ax=axes[0])
```






Tracks - Label: 0.0
Tracks - Label: 0.0
Tracks - Label: 0.0
Tracks - Label: 0.0
Tracks - Label: 1.0


Correlation Matrix
m0
pt
y
1.00
0.55
-0.24
0.55
1.00
0.04
-0.24
0.04
1.00
m0
pt
y
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

```
import os
import h5py
import numpy as np
import matplotlib.pyplot as plt

# ✅ Google Drive Mounting
from google.colab import drive
drive.mount('/content/drive')

# ✅ File Paths
file_path = '/content/drive/MyDrive/quark-gluon_data-set_n139306.hdf5'
save_dir = '/content/drive/MyDrive/QuarkGluonBatches/'  # Storage location

# ✅ Ensure save directory exists
os.makedirs(save_dir, exist_ok=True)

class QuarkGluonImageLoader:
    def __init__(self, file_path, batch_size=1000, save_dir=save_dir):
        """
        Initialize the dataset loader with lazy loading and Google Drive saving.
        """
        self.file_path = file_path
        self.batch_size = batch_size
        self.save_dir = save_dir
        self.total_samples = 0

        # Open the HDF5 file
        with h5py.File(file_path, 'r') as f:
            self.total_samples = f['X_jets'].shape[0]  # Number of samples
            print(f"Total samples: {self.total_samples}")

    def save_batch(self, batch_idx, X_batch):
        """
        Save the raw batch (unprocessed) as .npy files in Google Drive.
        """
        np.save(os.path.join(self.save_dir, f'X_batch_{batch_idx}.npy'), X_batch)
        print(f"✅ Saved Batch {batch_idx} to Google Drive")

    def get_last_saved_batch(self):
        """
        Check Google Drive for the last saved batch index.
        """
        existing_batches = [
            int(fname.split('_')[2].split('.')[0])
            for fname in os.listdir(self.save_dir) if fname.startswith('X_batch_')
        ]

        if existing_batches:
            last_batch = max(existing_batches)
            print(f"🔚 Resuming from batch {last_batch + 1}")
            return last_batch + 1
        else:
            print("🚀 Starting from the beginning.")
            return 0

    def __iter__(self):
        """
        Iterate through the dataset in batches with lazy loading and resume from last batch.
        """
        start_batch = self.get_last_saved_batch()

        with h5py.File(self.file_path, 'r') as f:
            for batch_idx, start in enumerate(range(0, self.total_samples, self.batch_size)):
                # Resume from the last saved batch
                if batch_idx < start_batch:
                    continue

                end = min(start + self.batch_size, self.total_samples)

                # Load raw batch (unprocessed images)
                raw_batch = f['X_jets'][start:end]  # No preprocessing

                # Save the raw batch to Google Drive
                self.save_batch(batch_idx, raw_batch)

                # Preprocess only for training
                preprocessed_batch = raw_batch / 255.0
                yield raw_batch, preprocessed_batch

# ✅ **Usage**
batch_size = 2000
```

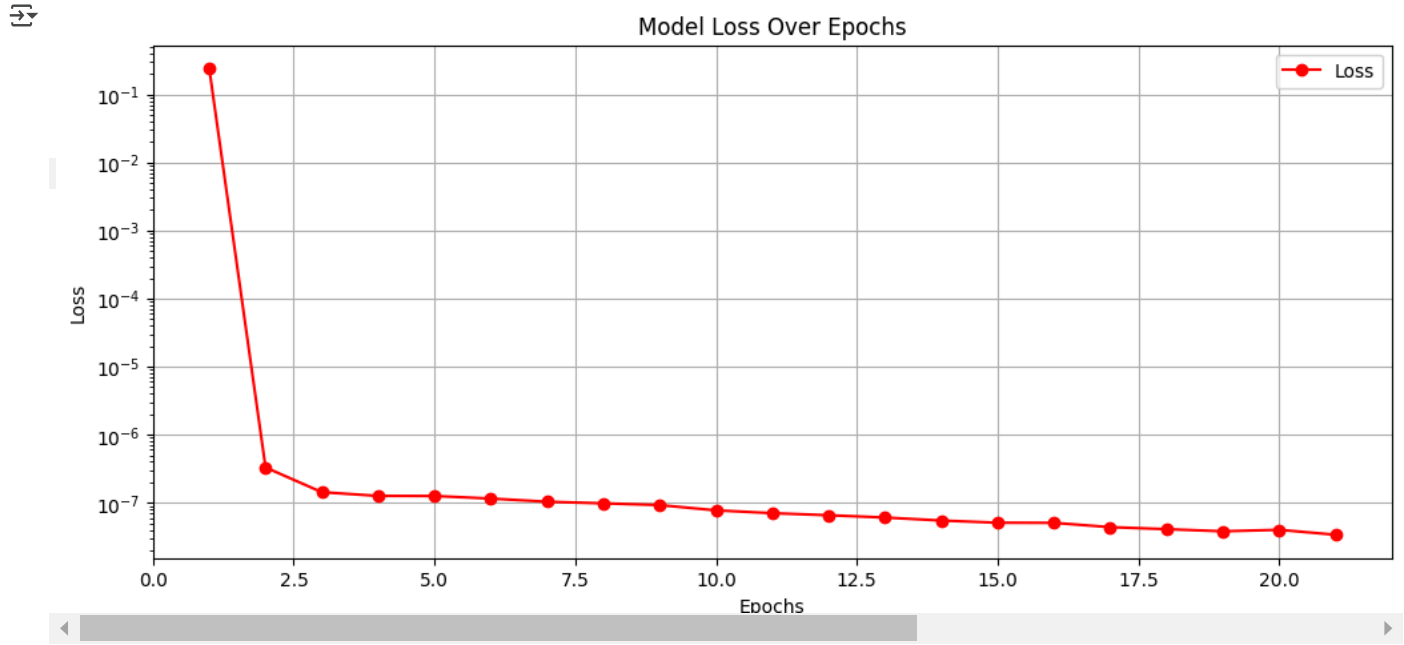
```
# Instantiate the loader
loader = QuarkGluonImageLoader(file_path, batch_size=batch_size)

# Iterate through batches with lazy loading and img/255.0 normalization
for i, (raw_batch, preprocessed_batch) in enumerate(loader):
    print(f"\nBatch {i+1}:")
    print(f"Raw batch shape: {raw_batch.shape}")
    print(f"Preprocessed batch shape: {preprocessed_batch.shape}")

    # Display raw and preprocessed images for the first batch
    if i == 0:
        fig, axes = plt.subplots(1, 2, figsize=(12, 6))

        # Display raw image
        axes[0].imshow(raw_batch[0] / 255.0)
        axes[0].set_title('Raw Image')
        axes[0].axis('off')

        # Display preprocessed image
        axes[1].imshow(preprocessed_batch[0])
        axes[1].set_title('Preprocessed (img/255.0)')
        axes[1].axis('off')

        plt.tight_layout()
        plt.show()


import os
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras import layers, models, callbacks
from tensorflow.keras.mixed_precision import set_global_policy

# ✅ Enable mixed precision training
set_global_policy('mixed_float16')

# ✅ Google Drive path
GDRIVE_PATH = '/content/drive/My Drive/QuarkGluonBatches/'

# ✅ Parameters
BATCH_SIZE = 256  # Reduced batch size to prevent memory exhaustion
IMG_SHAPE = (125, 125, 3)

# ✅ Remove double normalization
def create_dataset(folder_path, batch_size, img_shape):
    file_paths = [os.path.join(folder_path, f) for f in os.listdir(folder_path) if f.endswith('.npy')]

    def load_and_slice(file_path):
        large_batch = np.load(file_path)  # Load the entire batch
        num_slices = large_batch.shape[0] // batch_size

        for i in range(num_slices):
            batch_slice = large_batch[i * batch_size:(i + 1) * batch_size]

            # Remove second normalization (no need to divide by 255 again)
            preprocessed_batch = batch_slice / 255.0  # Normalize only once

            yield batch_slice, preprocessed_batch

    def generator():
        for path in file_paths:
            for raw, preprocessed in load_and_slice(path):
                yield raw, preprocessed

    output_signature = (
        tf.TensorSpec(shape=(batch_size, *img_shape), dtype=tf.float32),
        tf.TensorSpec(shape=(batch_size, *img_shape), dtype=tf.float32)
    )

    dataset = tf.data.Dataset.from_generator(generator, output_signature=output_signature)
    dataset = dataset.shuffle(len(file_paths))
    dataset = dataset.prefetch(tf.data.experimental.AUTOTUNE)

    return dataset


# ✅ Create the dataset
dataset = create_dataset(GDRIVE_PATH, BATCH_SIZE, IMG_SHAPE)

# ✅ Autoencoder Model with Bottleneck
def build_autoencoder(input_shape):
```

```
    encoder_input = layers.Input(shape=input_shape)

    # Encoder
    x = layers.Conv2D(32, (3, 3), strides=(2, 2), padding='same', activation='relu')(encoder_input)
    x = layers.Conv2D(64, (3, 3), strides=(2, 2), padding='same', activation='relu')(x)
    x = layers.Conv2D(128, (3, 3), strides=(2, 2), padding='same', activation='relu')(x)

    # Bottleneck
    x = layers.Flatten()(x)
    bottleneck = layers.Dense(256, activation='relu')(x)

    # Decoder
    x = layers.Dense(16 * 16 * 128, activation='relu')(bottleneck)
    x = layers.Reshape((16, 16, 128))(x)
    x = layers.Conv2DTranspose(128, (3, 3), strides=(2, 2), padding='same', activation='relu')(x)
    x = layers.Conv2DTranspose(64, (3, 3), strides=(2, 2), padding='same', activation='relu')(x)
    x = layers.Conv2DTranspose(32, (3, 3), strides=(2, 2), padding='same', activation='relu')(x)
    x = layers.Conv2D(3, (3, 3), padding='same', activation='sigmoid')(x)

    # Ensure output matches input size
    decoder_output = layers.Cropping2D(cropping=((1, 2), (1, 2)))(x)

    autoencoder = models.Model(inputs=encoder_input, outputs=decoder_output)
    optimizer = tf.keras.optimizers.Adam(learning_rate=0.0001, clipnorm=1.0)
    autoencoder.compile(optimizer=optimizer, loss='mse')
    return autoencoder

# ✅ Instantiate the autoencoder
autoencoder = build_autoencoder(IMG_SHAPE)


# ✅ Use model.fit() for efficient memory usage
steps_per_epoch = len(os.listdir(GDRIVE_PATH)) * (2000 // BATCH_SIZE) // 4

# ✅ Callbacks for early stopping and model checkpointing
early_stopping = callbacks.EarlyStopping(monitor='loss', patience=4, restore_best_weights=True)
model_checkpoint = callbacks.ModelCheckpoint(
    os.path.join(GDRIVE_PATH, 'best_autoencoder_model2.keras'),
    monitor='loss',
    save_best_only=True
)

# ✅ Train the model with callbacks
autoencoder.fit(
    dataset,
    epochs=30,
    steps_per_epoch=steps_per_epoch,
    callbacks=[early_stopping, model_checkpoint]
)

# ✅ Save the final model
model_save_path = os.path.join(GDRIVE_PATH, 'autoencoder_model_final2.keras')
autoencoder.save(model_save_path)
print(f"Final model saved at {model_save_path}")
```

```
Epoch 1/30
127/127 ━━━━━━━━━━━━━━━━━━━━ 402s 2s/step - loss: 0.2393
Epoch 2/30
127/127 ━━━━━━━━━━━━━━━━━━━━ 289s 2s/step - loss: 3.3077e-07
Epoch 3/30
127/127 ━━━━━━━━━━━━━━━━━━━━ 391s 3s/step - loss: 1.4318e-07
Epoch 4/30
107/127 ━━━━━━━━━━━━━━━━━━── 16s 806ms/step - loss: 1.2635e-07/usr/local/lib/python3.11/dist-packages/keras/src/trainers/epoch_iterat
  self._interrupted_warning()
127/127 ━━━━━━━━━━━━━━━━━━━━ 88s 698ms/step - loss: 1.2593e-07
Epoch 5/30
127/127 ━━━━━━━━━━━━━━━━━━━━ 386s 2s/step - loss: 1.1543e-07
Epoch 6/30
127/127 ━━━━━━━━━━━━━━━━━━━━ 383s 3s/step - loss: 1.0367e-07
Epoch 7/30
127/127 ━━━━━━━━━━━━━━━━━━━━ 325s 3s/step - loss: 9.8274e-08
Epoch 8/30
127/127 ━━━━━━━━━━━━━━━━━━━━ 106s 840ms/step - loss: 9.2539e-08
Epoch 9/30
127/127 ━━━━━━━━━━━━━━━━━━━━ 403s 2s/step - loss: 7.7486e-08
Epoch 10/30
127/127 ━━━━━━━━━━━━━━━━━━━━ 383s 3s/step - loss: 7.0428e-08
Epoch 11/30
127/127 ━━━━━━━━━━━━━━━━━━━━ 366s 3s/step - loss: 6.5624e-08
Epoch 12/30
127/127 ━━━━━━━━━━━━━━━━━━━━ 112s 884ms/step - loss: 6.0784e-08
Epoch 13/30
127/127 ━━━━━━━━━━━━━━━━━━━━ 485s 3s/step - loss: 5.4898e-08
Epoch 14/30
127/127 ━━━━━━━━━━━━━━━━━━━━ 383s 3s/step - loss: 5.1004e-08
```

```
Epoch 15/30
127/127 ━━━━━━━━━━━━━━━━━━━━ 388s 3s/step - loss: 5.0746e-08
Epoch 16/30
127/127 ━━━━━━━━━━━━━━━━━━━━ 94s 742ms/step - loss: 4.3905e-08
Epoch 17/30
127/127 ━━━━━━━━━━━━━━━━━━━━ 490s 3s/step - loss: 4.1032e-08
Epoch 18/30
127/127 ━━━━━━━━━━━━━━━━━━━━ 316s 3s/step - loss: 3.8150e-08
Epoch 19/30
127/127 ━━━━━━━━━━━━━━━━━━━━ 389s 3s/step - loss: 4.0114e-08
Epoch 20/30
 33/127 ━━━━━━━━━━━━━━━━━━━━ 3:55 3s/step - loss: 3.3961e-08
```

```
import matplotlib.pyplot as plt

# ✅ Epochs
epochs = list(range(1, len(loss) + 1))

# ✅ Plotting
plt.figure(figsize=(12, 5))
plt.plot(epochs, loss, marker='o', color='red', label='Loss')

# ✅ Labels and Title
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Model Loss Over Epochs')
plt.yscale('log')
plt.legend()
plt.grid(True)

# ✅ Show Plot
plt.show()
```




```
# ✅ Recreate the dataset before visualization if running separately
GDRIVE_PATH = '/content/drive/My Drive/QuarkGluonBatches/'
BATCH_SIZE = 256
IMG_SHAPE = (125, 125, 3)

# ✅ Create the dataset
dataset = create_dataset(GDRIVE_PATH, BATCH_SIZE, IMG_SHAPE)
```

```
import os
import numpy as np
import tensorflow as tf
from PIL import Image

# ✅ Path Configurations
MODEL_PATH = '/content/drive/My Drive/QuarkGluonBatches/best_autoencoder_model2.keras'
OUTPUT_DIR = "/content/reconstructed_images"  # Output folder for reconstructed images
os.makedirs(OUTPUT_DIR, exist_ok=True)

# ✅ Load the final model
if os.path.exists(MODEL_PATH):
    print(f"🔥 Loading saved model from {MODEL_PATH}")
```

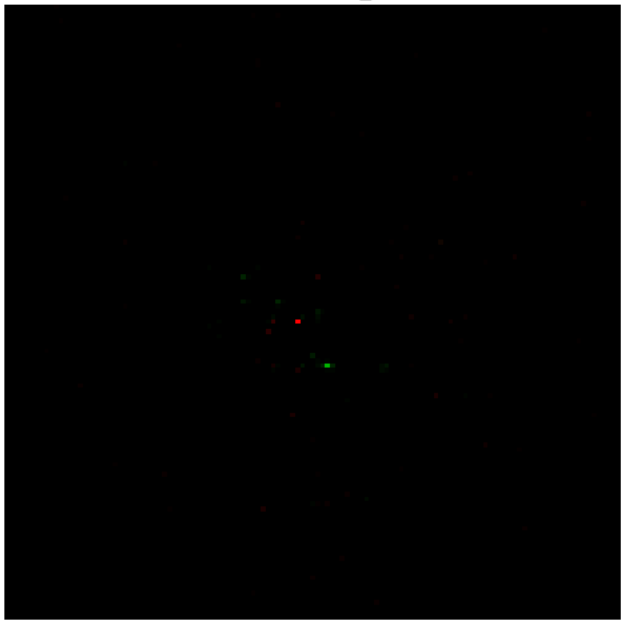
```
    autoencoder = tf.keras.models.load_model(MODEL_PATH)
else:
    raise FileNotFoundError("❌ No saved model found!")

# ✅ Image Prediction and Saving
def predict_and_save(model, dataset, output_dir):
    img_index = 1

    for _, preprocessed_batch in dataset:  # Use preprocessed images for reconstruction
        # ✅ Predict reconstructed images
        reconstructed = model.predict(preprocessed_batch)

        # ✅ Rescale and save each reconstructed image
        reconstructed_rescaled = (reconstructed / np.max(reconstructed)) * 255.0  # Rescale
        reconstructed_rescaled = reconstructed_rescaled.astype(np.uint8)

        for i in range(reconstructed_rescaled.shape[0]):
            img = Image.fromarray(reconstructed_rescaled[i])
            img.save(os.path.join(output_dir, f"gen_{img_index}.jpg"))
            print(f"✅ Saved: gen_{img_index}.jpg")
            img_index += 1

# ✅ Call the prediction and save function
print("🔥 Generating and saving reconstructed images...")
predict_and_save(autoencoder, dataset, OUTPUT_DIR)
print("🎯 All reconstructed images saved successfully!")
```

```
🔥 Loading saved model from /content/drive/My Drive/QuarkGluonBatches/best_autoencoder_model2.keras
✅ Model loaded successfully!

🔥 Generating and saving reconstructed images...
✅ Saved: gen_1.jpg
✅ Saved: gen_2.jpg
✅ Saved: gen_3.jpg
✅ Saved: gen_4.jpg
✅ Saved: gen_5.jpg
✅ Saved: gen_6.jpg
✅ Saved: gen_7.jpg
✅ Saved: gen_8.jpg
✅ Saved: gen_9.jpg
✅ Saved: gen_10.jpg

🎯 All reconstructed images saved successfully!
🗂️ Output directory: /content/
```

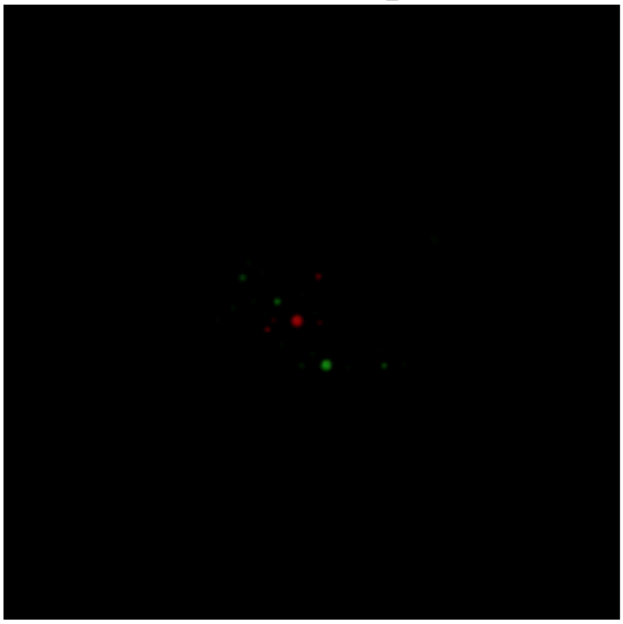
```
import os
import matplotlib.pyplot as plt
from PIL import Image

# ✅ Path Configuration
img_dir = "/content/"  # Single directory containing both original and reconstructed images

# ✅ Collect and Pair Image Files
og_files = sorted([f for f in os.listdir(img_dir) if f.startswith("image_") and f.endswith(('.png', '.jpg'))])
recon_files = {f.replace("image_", "gen_").split('.')[0]: f for f in os.listdir(img_dir) if f.startswith("gen_")}

# ✅ Ensure matching pairs exist
paired_files = [(og, recon_files.get(og.split('.')[0].replace("image_", "gen_"))) for og in og_files]

# ✅ Filter out any missing pairs
paired_files = [(og, recon) for og, recon in paired_files if recon is not None]

# ✅ Plot Configuration
num_images = len(paired_files)
cols = 2
rows = num_images

fig, axes = plt.subplots(rows, cols, figsize=(10, 5 * rows))

# ✅ Display images side by side
for i, (og_file, recon_file) in enumerate(paired_files):
    og_path = os.path.join(img_dir, og_file)
    recon_path = os.path.join(img_dir, recon_file)

    # Load images
    og_img = Image.open(og_path)
    recon_img = Image.open(recon_path)

    # Plot original
    axes[i, 0].imshow(og_img)
    axes[i, 0].set_title(f"Original: {og_file}")
    axes[i, 0].axis('off')
```

```
        # Plot reconstructed
        axes[i, 1].imshow(recon_img)
        axes[i, 1].set_title(f"Reconstructed: {recon_file}")
        axes[i, 1].axis('off')

    plt.tight_layout()
    plt.show()
```

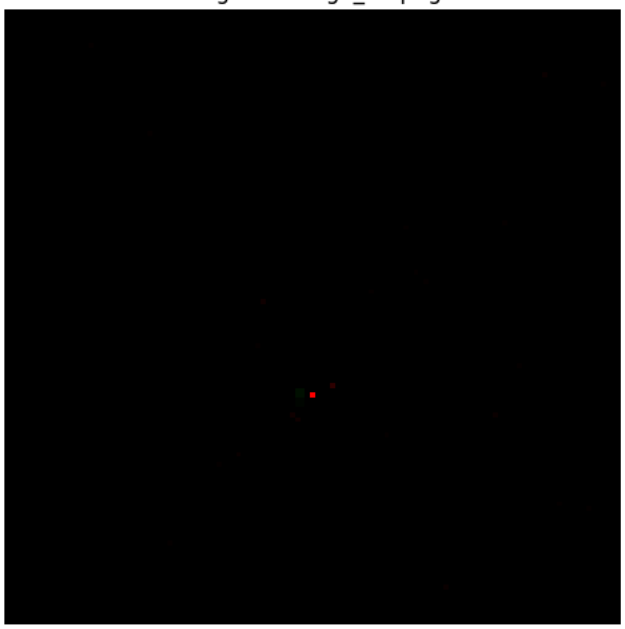
Original: image_1.png



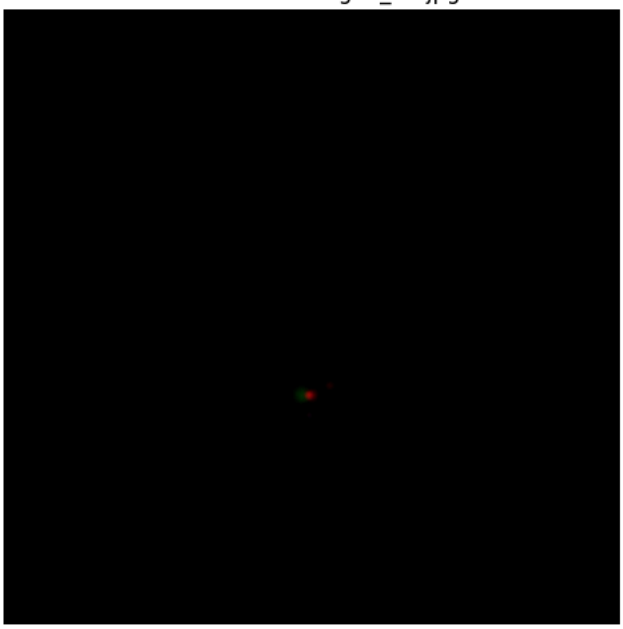
Reconstructed: gen_1.jpg



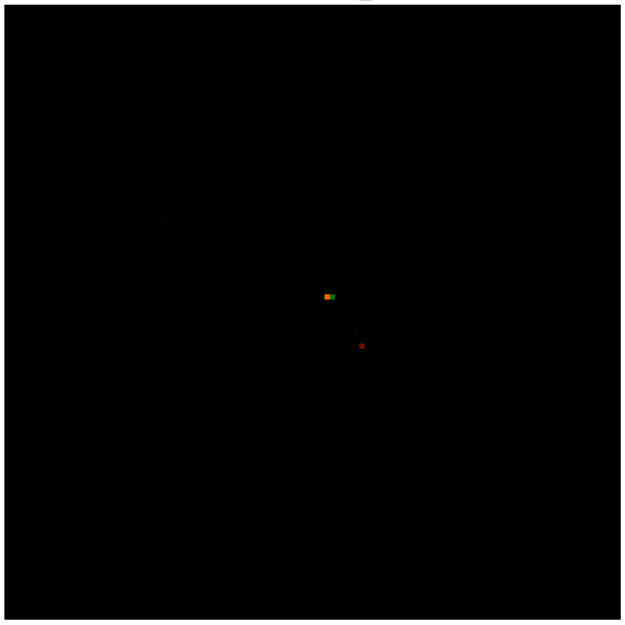
Original: image_10.png



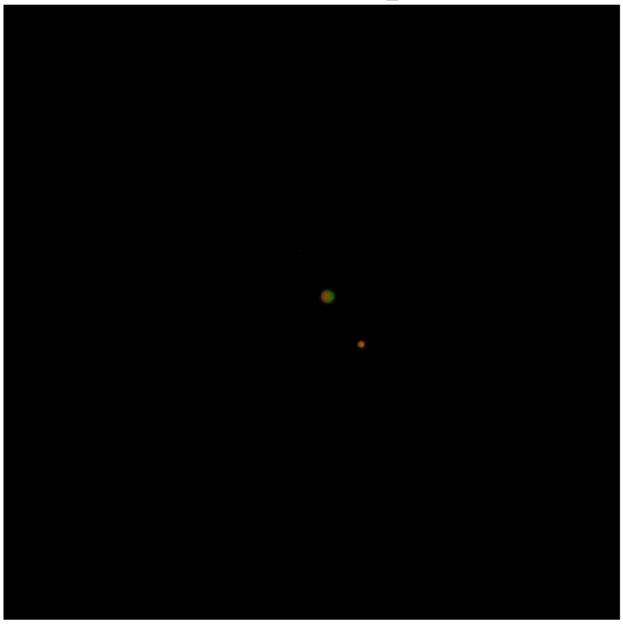
Reconstructed: gen_10.jpg



Original: image_2.png



Reconstructed: gen_2.jpg



Original: image_3.png

Reconstructed: gen_3.jpg