Project Step 3 Assignment: Astronomy Inference Submission

Group: 2

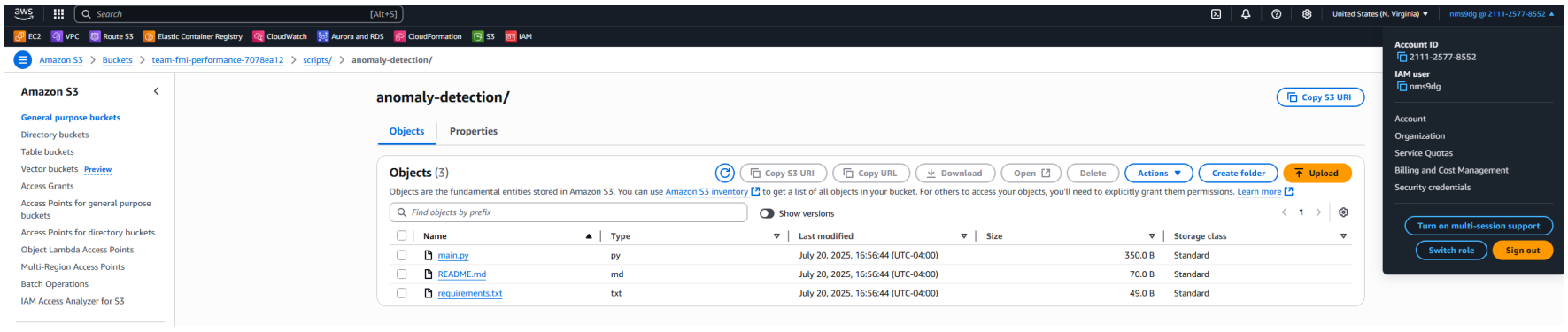
Zachary Holland (nms9dg) & DevlinBridges

1. Clone the AI for Astronomy repository from GitHub (https://github.com/UVA-MLSys/AI-for-Astronomy.git)

In [ ]:
```
!git clone https://github.com/UVA-MLSys/AI-for-Astronomy.git
```

```
Cloning into 'AI-for-Astronomy'...
remote: Enumerating objects: 4551, done.
remote: Counting objects: 100% (639/639), done.
remote: Compressing objects: 100% (246/246), done.
remote: Total 4551 (delta 277), reused 554 (delta 215), pack-reused 3912 (from 1)
Receiving objects: 100% (4551/4551), 204.00 MiB | 43.02 MiB/s, done.
Resolving deltas: 100% (3788/3788), done.
Updating files: 100% (218/218), done.
```

Screenshots of repository cloning



2. Navigate to the Anomaly Detection/inference directory within the code folder

In [ ]:
```
!cd .. && pwd
```

```
/home/sagemaker-user/DS5111_Project/Project/AI-for-Astronomy/code/Anomaly Detection/Inference
```

3. Update the file paths in inference.py to match your local environment

Evidence of file path updates in inference.py

In [ ]:
```
!cat "AI-for-Astronomy/code/Anomaly Detection/Inference/inference.py"
```

```
import sys, argparse, json
from torch.profiler import profile, record_function, ProfilerActivity
# sys.path.append('/scratch/aww9gh/Cosmic_Cloud/AI-for-Astronomy/code/Anomaly Detection/') #adjust based on your system's direc
tory
sys.path.append('/home/sagemaker-user/DS5111_Project/Project/AI-for-Astronomy/code/Anomaly Detection') #updated for local envir
onment
import torch, time, os
import numpy as np
import Plot_Redshift as plt_rdshft
from torch.utils.data import DataLoader
from blocks.model_vit_inception import ViT_Astro

#Load Data
def load_data(data_path, device):
    return torch.load(data_path, map_location = device)

#Load Model
def load_model(model_path, device):
    model = torch.load(model_path, map_location = device)
    return model.module.eval()

#Use DataLoader for iterating over batches
def data_loader(data, batch_size):
    return DataLoader(data, batch_size = batch_size, drop_last = False)   #Drop samples out of the batch size


# Define the inference function with profiling for both CPU and GPU memory usage
def inference(model, dataloader, real_redshift, save_path, device, batch_size):
redshift_analysis = []
total_time = 0.0  # Initialize total time for execution
num_batches = 0   # Initialize number of batches
total_data_bits = 0  # Initialize total data bits processed

# Initialize the profiler to track both CPU and GPU activities and memory usage
with profile(activities=[ProfilerActivity.CPU, ProfilerActivity.CUDA],
             profile_memory=True,
             record_shapes=True) as prof:

    for i, data in enumerate(dataloader):
        with torch.no_grad():
            image = data[0].to(device)  # Image to device
            magnitude = data[1].to(device)  # Magnitude to device

            with record_function("model_inference"):
                predict_redshift = model([image, magnitude])  # Model inference

            # Append the redshift prediction to analysis list
            redshift_analysis.append(predict_redshift.view(-1, 1))
            num_batches += 1

            # Calculate data size for this batch
            image_bits = image.element_size() * image.nelement() * 8  # Convert bytes to bits
            magnitude_bits = magnitude.element_size() * magnitude.nelement() * 8  # Convert bytes to bits
            total_data_bits += image_bits + magnitude_bits  # Add data bits for this batch

num_samples = len(real_redshift)
redshift_analysis = torch.cat(redshift_analysis, dim=0)
redshift_analysis = redshift_analysis.cpu().detach().numpy().reshape(num_samples,)

    # Extract total time and memory usage for CPU and GPU
    total_cpu_memory = prof.key_averages().total_average().cpu_memory_usage / 1e6  # Convert bytes to MB
    total_gpu_memory = prof.key_averages().total_average().cuda_memory_usage / 1e6  # Convert bytes to MB

    # Extract total CPU and GPU time
    total_cpu_time = prof.key_averages().total_average().cpu_time_total / 1e6  # Convert from microseconds to seconds
    total_gpu_time = prof.key_averages().total_average().cuda_time_total / 1e6  # Convert from microseconds to seconds


    total_time = max(total_cpu_time, total_gpu_time)

    avg_time_batch = total_time / num_batches

    execution_info = {
            'total_cpu_time': total_cpu_time,
            'total_gpu_time': total_gpu_time,
            'total_cpu_memory': total_cpu_memory,
            'total_gpu_memory': total_gpu_memory,
            'execution_time_per_batch': avg_time_batch,   # Average execution time per batch
            'num_batches': num_batches,   # Number of batches
            'batch_size': batch_size,   # Batch size
            'device': device,   # Selected device
            'throughput_bps': total_data_bits / total_time,   # Throughput in bits per second (using total_time for all batche
s)
        }
    # Invoke the evaluation metrics
    plt_rdshft.err_calculate(redshift_analysis, real_redshift, execution_info, save_path)
    plt_rdshft.plot_density(redshift_analysis, real_redshift, save_path)
```

```
#This is the engine module for invoking and calling various modules
def engine(args):
    data = load_data(args.data_path, args.device)
    dataloader = data_loader(data, args.batch_size)
    model = load_model(args.model_path, args.device)
    inference(model, dataloader, data[:][2].to('cpu'), args.save_path, device = args.device, batch_size = args.batch_size)


# Pathes and other inference hyperparameters can be adjusted below
if __name__ == '__main__':
    prj_dir = '/home/sagemaker-user/DS5111_Project/Project/AI-for-Astronomy/code/Anomaly Detection/' #updated for local environ
ment
    parser = argparse.ArgumentParser()
    parser.add_argument('--batch_size', type=int, default=512)
    parser.add_argument('--data_path', type = str, default = '/home/sagemaker-user/DS5111_Project/Project/AI-for-Astronomy/cod
e/Anomaly Detection/Inference/resized_inference.pt')
    parser.add_argument('--model_path', type = str, default  = prj_dir + 'Fine_Tune_Model/Mixed_Inception_z_VITAE_Base_Img_Full
_New_Full.pt')
    parser.add_argument('--device', type = str, default = 'cpu')    # To run on GPU, put cuda, and on CPU put cpu

    parser.add_argument('--save_path', type = str, default = prj_dir + 'Plots/')
    args = parser.parse_args()
    engine(args)
```

In [ ]:
```
!grep -n "sys.path.append" inference.py
!grep -n "data_path" inference.py
!grep -n "model_path" inference.py
```

```
3:# sys.path.append('/scratch/aww9gh/Cosmic_Cloud/AI-for-Astronomy/code/Anomaly Detection/') #adjust based on your system's dir
ectory
4:sys.path.append('/home/sagemaker-user/DS5111_Project/Project/AI-for-Astronomy/code/Anomaly Detection') #updated for local env
ironment
25:def load_data(data_path, device):
26:    return torch.load(data_path, map_location = device)
102:    data = load_data(args.data_path, args.device)
113:    parser.add_argument('--data_path', type = str, default = '/home/sagemaker-user/DS5111_Project/Project/AI-for-Astronomy/
code/Anomaly Detection/Inference/resized_inference.pt')
29:def load_model(model_path, device):
30:    model = torch.load(model_path, map_location = device)
104:    model = load_model(args.model_path, args.device)
114:    parser.add_argument('--model_path', type = str, default  = prj_dir + 'Fine_Tune_Model/Mixed_Inception_z_VITAE_Base_Img_
Full_New_Full.pt')
```

4. Run the inference.py script and document the execution time

Documentation of execution time

In [ ]:
```
import sys
!cd "AI-for-Astronomy/code/Anomaly Detection/Inference" && time {sys.executable} inference.py
```

```
/home/sagemaker-user/.conda/envs/data_science_on_aws/lib/python3.7/site-packages/torch/autograd/profiler.py:176: UserWarning: C
UDA is not available, disabling CUDA profiling
  warn("CUDA is not available, disabling CUDA profiling")
STAGE:2025-07-20 21:44:55 430:430 ActivityProfilerController.cpp:294] Completed Stage: Warm Up
/bin/bash: line 1:   430 Killed                  /home/sagemaker-user/.conda/envs/data_science_on_aws/bin/python inference.py

real    0m18.704s
user    0m8.574s
sys     0m2.073s
```

Killed due to memory issues, instance crashed, increased RAM

In [ ]:
```
# AI-for-Astronomy Inference directory
%cd ~/DS5111_Project/Project/AI-for-Astronomy/code/Anomaly\ Detection/Inference

# verify were in the right place
!pwd
!ls -la

# plots dir
!ls -la ../Plots/

# run inference add batch size
!python inference.py --batch_size 32 --device cpu
```

```
/home/sagemaker-user/DS5111_Project/Project/AI-for-Astronomy/code/Anomaly Detection/Inference
/home/sagemaker-user/DS5111_Project/Project/AI-for-Astronomy/code/Anomaly Detection/Inference
total 25440
drwxr-xr-x 2 sagemaker-user users      120 Jul 20 21:16  .
drwxr-xr-x 7 sagemaker-user users      185 Jul 20 21:29  ..
-rw-r--r-- 1 sagemaker-user users   320045 Jul 20 21:08 'Inference Step by Step Instructions.pdf'
-rw-r--r-- 1 sagemaker-user users        0 Jul 20 21:08  __init__.py
-rw-r--r-- 1 sagemaker-user users     5765 Jul 20 21:36  inference.py
-rw-r--r-- 1 sagemaker-user users 25718362 Jul 20 21:08  resized_inference.pt
total 1096
drwxr-xr-x 2 sagemaker-user users   4096 Jul 20 21:08  .
drwxr-xr-x 7 sagemaker-user users    185 Jul 20 21:29  ..
-rw-r--r-- 1 sagemaker-user users   4062 Jul 20 21:08 'Cloud for CosmicAI(CURRENT - vary batch size).csv'
-rw-r--r-- 1 sagemaker-user users   8724 Jul 20 21:08 'Cloud for CosmicAI(CURRENT - vary datasize).csv'
-rw-r--r-- 1 sagemaker-user users 621470 Jul 20 21:08  CosmicAIPlots.ipynb
-rw-r--r-- 1 sagemaker-user users    428 Jul 20 21:08  Results.json
-rw-r--r-- 1 sagemaker-user users   1089 Jul 20 21:08  System_Info.json
-rw-r--r-- 1 sagemaker-user users 144674 Jul 20 21:08  batchsize_throughput.jpg
-rw-r--r-- 1 sagemaker-user users 130258 Jul 20 21:08  batchsize_throughput_log_scaled.jpg
-rw-r--r-- 1 sagemaker-user users 135613 Jul 20 21:08  datasize_time_log_scaled.jpg
-rw-r--r-- 1 sagemaker-user users  48542 Jul 20 21:08  inference.png
-rw-r--r-- 1 sagemaker-user users    177 Jul 20 21:08  inference.png_Results.json
/home/sagemaker-user/.conda/envs/data_science_on_aws/lib/python3.7/site-packages/torch/autograd/profiler.py:176: UserWarning: C
UDA is not available, disabling CUDA profiling
  warn("CUDA is not available, disabling CUDA profiling")
STAGE:2025-07-20 21:49:17 1172:1172 ActivityProfilerController.cpp:294] Completed Stage: Warm Up
```

5. Capture the output results including inference.png and Results.json

Captured output files (inference.png and Results.json)

In [ ]:
```
!ls -la ../Plots/
```

```
total 1096
drwxr-xr-x 2 sagemaker-user users   4096 Jul 20 21:08  .
drwxr-xr-x 7 sagemaker-user users    185 Jul 20 21:29  ..
-rw-r--r-- 1 sagemaker-user users   4062 Jul 20 21:08 'Cloud for CosmicAI(CURRENT - vary batch size).csv'
-rw-r--r-- 1 sagemaker-user users   8724 Jul 20 21:08 'Cloud for CosmicAI(CURRENT - vary datasize).csv'
-rw-r--r-- 1 sagemaker-user users 621470 Jul 20 21:08  CosmicAIPlots.ipynb
-rw-r--r-- 1 sagemaker-user users    428 Jul 20 21:08  Results.json
-rw-r--r-- 1 sagemaker-user users   1089 Jul 20 21:08  System_Info.json
-rw-r--r-- 1 sagemaker-user users 144674 Jul 20 21:08  batchsize_throughput.jpg
-rw-r--r-- 1 sagemaker-user users 130258 Jul 20 21:08  batchsize_throughput_log_scaled.jpg
-rw-r--r-- 1 sagemaker-user users 135613 Jul 20 21:08  datasize_time_log_scaled.jpg
-rw-r--r-- 1 sagemaker-user users  48542 Jul 20 21:08  inference.png
-rw-r--r-- 1 sagemaker-user users    177 Jul 20 21:08  inference.png_Results.json
```

In [ ]:
```
# Results.json content
!cat ../Plots/Results.json
```


```
{
    "total execution time": 131.83914375305176,
    "throughput": 31118881.716076322,
    "average execution time (milliseconds) per batch": 168.80812260313925,
    "batch size": 32,
    "number of batches": 781,
    "device": "cpu",
    "MAE": 0.01336825733453455,
    "MSE": 0.0003767368048620285,
    "Bias": 0.002923277978249915,
    "Precision": 0.011839682161808014,
    "R2": 0.9684378430247307
}
```


In [ ]:
```
# System_Info.json for system specs
!cat ../Plots/System_Info.json
```

```
{
     "System": "Windows",
     "Node Name": "MEL",
     "Release": "10",
     "Version": "10.0.22631",
     "Machine": "AMD64",
     "Processor": "AMD64 Family 25 Model 80 Stepping 0, AuthenticAMD",
     "GPU Info": "No GPU available",
     "CPU Info": {
          "Physical Cores": 8,
          "Total Cores": 16,
          "Max Frequency": "2000.00Mhz",
          "Min Frequency": "0.00Mhz",
          "Current Frequency": "1808.00Mhz",
          "Total CPU Usage": "17.8%",
          "Per Core Usage": [
               "17.7%",
               "23.6%",
               "30.5%",
               "28.9%",
               "8.1%",
               "8.8%",
               "21.5%",
               "24.3%",
               "2.4%",
               "2.5%",
               "18.4%",
               "19.7%",
               "2.9%",
               "3.0%",
               "48.6%",
               "24.7%"
          ]
     },
     "Memory Info": {
          "Total Memory": "15.34 GB",
          "Available Memory": "10.71 GB",
          "Memory Usage": "30.2%"
     }
}
```


Documentation of Execution Time

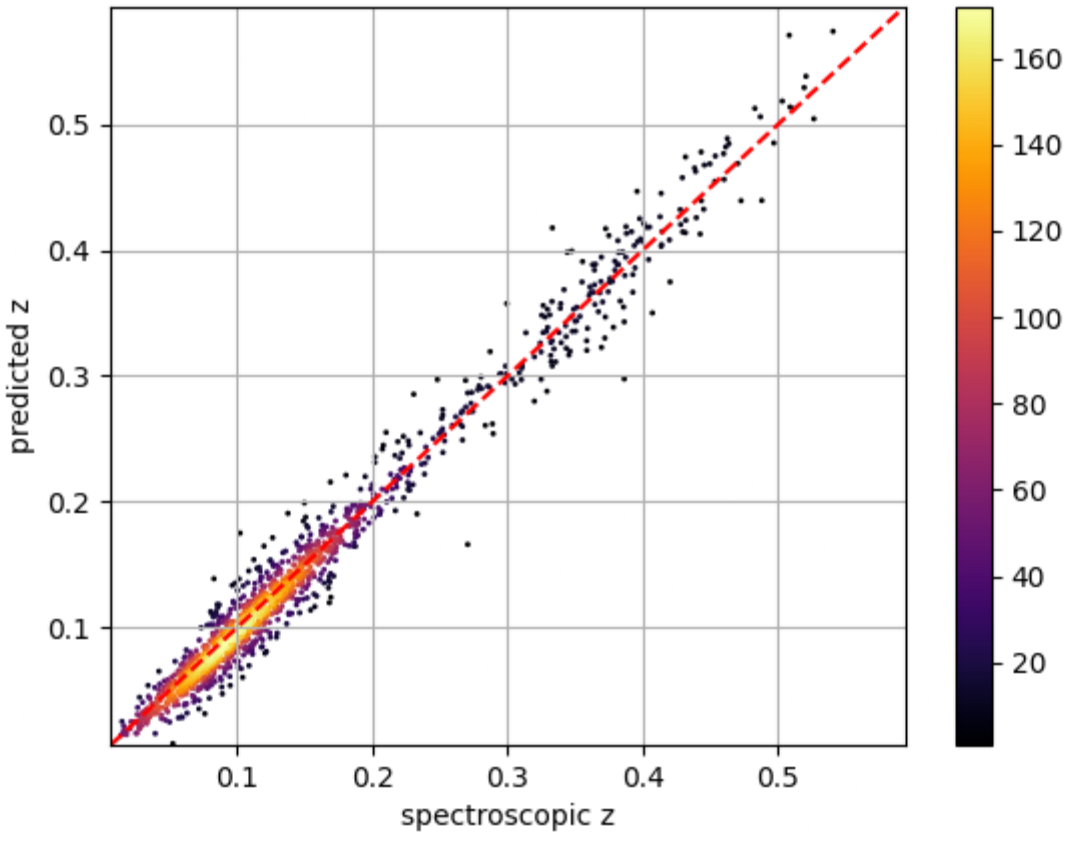
- **Execution Environment**: CPU-only (no GPU available)
- **Batch Size**: 32
- **Total Execution Time**: **131.84 seconds**
- **Average Time per Batch**: **168.81 milliseconds**
- **Total Number of Batches**: 781
- **Device Used**: CPU

This timing data was captured from the JSON inference results and reflects the performance of the model under the specified conditions on the local system.

In [ ]:
```
# copying results to current dir
!cp ../Plots/inference.png ./
!cp ../Plots/Results.json ./
```

In [ ]:
```
%cd ~/DS5111_Project/Project/AI-for-Astronomy/code/Anomaly\ Detection/Inference
```

```
/home/sagemaker-user/DS5111_Project/Project/AI-for-Astronomy/code/Anomaly Detection/Inference
```



6. Provide a brief analysis of the inference performance on your system

Analysis of inference performance

1. System Specifications

Component	Details
Operating System	Windows 10 (Version 10.0.22631)
CPU	AMD64, 8 Physical Cores / 16 Logical Cores
CPU Frequency	Max: 2.00 GHz, Current: 1.81 GHz
Total RAM	15.34 GB
Available RAM	10.71 GB (Memory Usage: 30.2%)
GPU Availability	*No dedicated GPU available*

---

2. Inference Execution Metrics

Metric	Value
Execution Device	CPU
Total Execution Time	131.84 seconds
Batch Size	32
Total Number of Batches	781
Average Time per Batch	168.81 milliseconds
Data Throughput	31,118,881 bits/sec

---

3. Prediction Performance Metrics

Metric	Value
Mean Absolute Error (MAE)	0.0134
Mean Squared Error (MSE)	0.00038
Bias	0.00292
Precision (Std. Deviation)	0.01184
R^2 Score	**0.968**

---

4. Analysis Summary

The inference was done on a CPU-only system without GPU acceleration. Despite hardware constraints, the model we had decent processing, averaging ~169 milliseconds per batch and maintaining a throughput of over 31 million bits per second.

The predictive results indicate high accuracy and low error. An **R^2 of 0.968** shows strong correlation between predicted and actual values, with minimal mean absolute and squared errors. Low bias and standard deviation further support the consistency of the model's output.

This validates the model's suitability for CPU-based deployments where GPU access may be limited.

Test several different deployment options and compare several different benchmarks.

Comparison of different deployment options (Batch Size)

In [ ]:
```
# test different batch sizes and store results separately

# test 1: Batch size 2
!echo "=== Testing Batch Size 2 ==="
!mkdir -p results_batch_2
!time python inference.py --batch_size 2 --device cpu --save_path ./results_batch_2/
!cp ./results_batch_2/Results.json ./results_batch_2_Results.json
```

```
=== Testing Batch Size 2 ===
/home/sagemaker-user/.conda/envs/data_science_on_aws/lib/python3.7/site-packages/torch/autograd/profiler.py:176: UserWarning: C
UDA is not available, disabling CUDA profiling
  warn("CUDA is not available, disabling CUDA profiling")
STAGE:2025-07-21 00:19:48 4665:4665 ActivityProfilerController.cpp:294] Completed Stage: Warm Up
STAGE:2025-07-21 00:20:11 4665:4665 ActivityProfilerController.cpp:300] Completed Stage: Collection

real    7m4.963s
user    7m51.344s
sys     0m7.020s
```

In [ ]:
```
# test 2: Batch size 8
!echo "=== Testing Batch Size 8 ==="
!mkdir -p results_batch_8
!time python inference.py --batch_size 8 --device cpu --save_path ./results_batch_8/
!cp ./results_batch_8/Results.json ./results_batch_8_Results.json
```

```
=== Testing Batch Size 8 ===
/home/sagemaker-user/.conda/envs/data_science_on_aws/lib/python3.7/site-packages/torch/autograd/profiler.py:176: UserWarning: C
UDA is not available, disabling CUDA profiling
  warn("CUDA is not available, disabling CUDA profiling")
STAGE:2025-07-21 00:01:47 2702:2702 ActivityProfilerController.cpp:294] Completed Stage: Warm Up
STAGE:2025-07-21 00:01:54 2702:2702 ActivityProfilerController.cpp:300] Completed Stage: Collection

real    1m50.376s
user    2m6.774s
sys     0m2.952s
```

In [ ]:
```
# test 3: Batch size 16
!echo "=== Testing Batch Size 16 ==="
!mkdir -p results_batch_16
!time python inference.py --batch_size 16 --device cpu --save_path ./results_batch_16/
!cp ./results_batch_16/Results.json ./results_batch_16_Results.json
```

```
=== Testing Batch Size 16 ===
/home/sagemaker-user/.conda/envs/data_science_on_aws/lib/python3.7/site-packages/torch/autograd/profiler.py:176: UserWarning: C
UDA is not available, disabling CUDA profiling
  warn("CUDA is not available, disabling CUDA profiling")
STAGE:2025-07-21 00:03:47 2937:2937 ActivityProfilerController.cpp:294] Completed Stage: Warm Up
STAGE:2025-07-21 00:03:52 2937:2937 ActivityProfilerController.cpp:300] Completed Stage: Collection

real    0m58.948s
user    1m9.825s
sys     0m2.233s
```

In [ ]:
```
# test 4: Batch size 64
!echo "=== Testing Batch Size 64 ==="
!mkdir -p results_batch_64
!time python inference.py --batch_size 64 --device cpu --save_path ./results_batch_64/
!cp ./results_batch_64/Results.json ./results_batch_64_Results.json
```

```
=== Testing Batch Size 64 ===
/home/sagemaker-user/.conda/envs/data_science_on_aws/lib/python3.7/site-packages/torch/autograd/profiler.py:176: UserWarning: C
UDA is not available, disabling CUDA profiling
  warn("CUDA is not available, disabling CUDA profiling")
STAGE:2025-07-21 00:30:35 5790:5790 ActivityProfilerController.cpp:294] Completed Stage: Warm Up
STAGE:2025-07-21 00:30:37 5790:5790 ActivityProfilerController.cpp:300] Completed Stage: Collection

real    0m19.188s
user    0m25.197s
sys     0m1.921s
```

In [ ]:
```
# test 1: Batch size 1 - min
!echo "=== Testing Batch Size 1 ==="
!mkdir -p results_batch_1
!time python inference.py --batch_size 1 --device cpu --save_path ./results_batch_1/
!cp ./results_batch_1/Results.json ./results_batch_1_Results.json
```

```
=== Testing Batch Size 1 ===
/home/sagemaker-user/.conda/envs/data_science_on_aws/lib/python3.7/site-packages/torch/autograd/profiler.py:176: UserWarning: C
UDA is not available, disabling CUDA profiling
  warn("CUDA is not available, disabling CUDA profiling")
STAGE:2025-07-21 00:34:08 6181:6181 ActivityProfilerController.cpp:294] Completed Stage: Warm Up
STAGE:2025-07-21 00:34:49 6181:6181 ActivityProfilerController.cpp:300] Completed Stage: Collection

real    13m59.203s
user    15m18.101s
sys     0m13.048s
```

In [ ]:
```
%cd /home/sagemaker-user/DS5111_Project/Project/AI-for-Astronomy/code/Anomaly Detection/Inference
```

```
/home/sagemaker-user/DS5111_Project/Project/AI-for-Astronomy/code/Anomaly Detection/Inference
```

Fine-tuned Network & Performance Measurement

NOTE: Batch Size 32 was run on the default instance size, the rest were done on increased instance

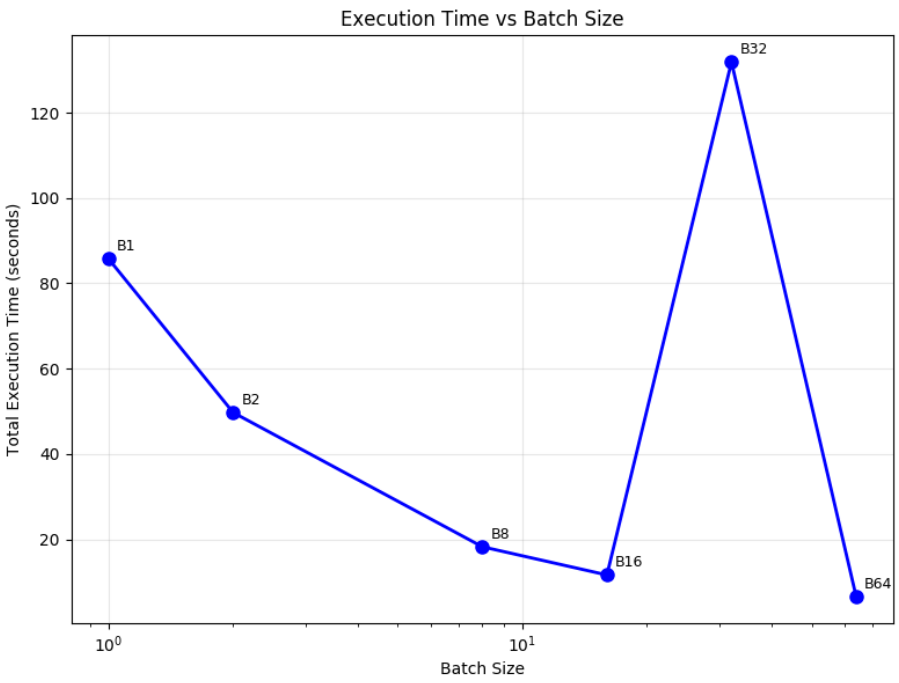
In [ ]:
```
import json
import pandas as pd
import matplotlib.pyplot as plt

# function to load and parse results
def load_results(filename):
    try:
        with open(filename, 'r') as f:
            return json.load(f)
    except:
        return None

# load (need to do key mapping)
results = []
batch_labels = []

print("=== LOADING RESULTS W KEY MAPPING ===")

# load 32
original_result = load_results('../Plots/Results.json')
if original_result:
    results.append(original_result)
    batch_labels.append('Original (32)')
    print("Loaded original batch 32 results")

# load rest
test_batches = [1, 2, 8, 16, 64]
for batch in test_batches:
    result = load_results(f'results_batch_{batch}_Results.json')
    if result:
        results.append(result)
        batch_labels.append(f'Test ({batch})')
        print(f"Loaded batch {batch} results")

print(f"\nSuccessfully loaded {len(results)} result files")

# comparison df
df_data = []
for i, result in enumerate(results):
    if result:
        # format
        if 'total execution time' in result:
            # orig
            data_entry = {
                'batch_size': result['batch size'],
                'run_type': batch_labels[i],
                'total_time': result['total execution time'],
                'throughput': result['throughput'],
                'avg_time_per_batch': result['average execution time (milliseconds) per batch'],
                'num_batches': result['number of batches'],
                'mae': result['MAE'],
                'r2_score': result['R2'],
                'mse': result['MSE'],
                'bias': result['Bias'],
                'precision': result['Precision'],
                'device': result['device']
            }
        elif 'total cpu time (second)' in result:
            # batches 1,2,8,16,64
```

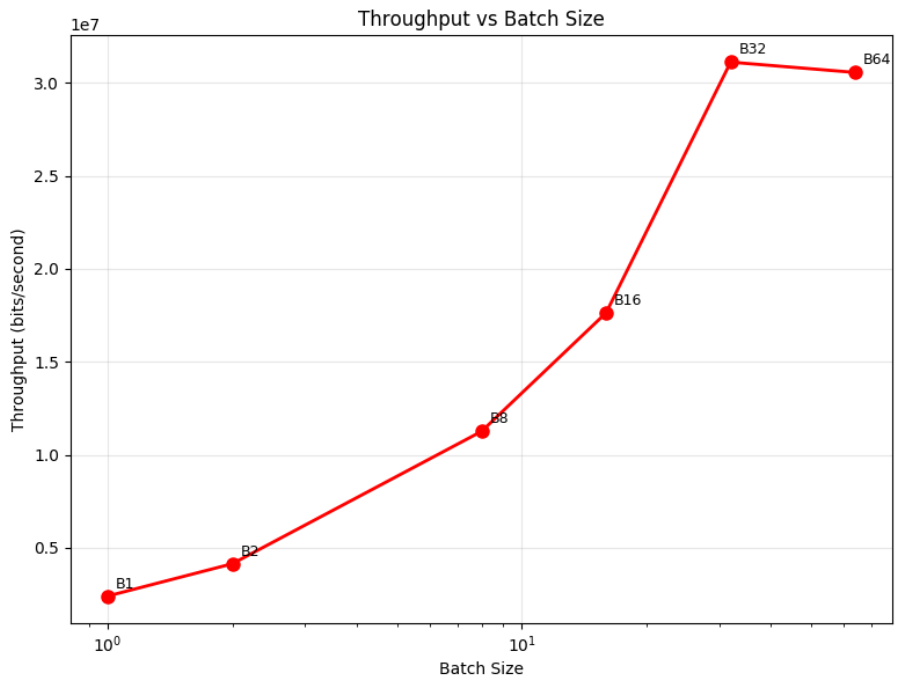
```
            data_entry = {
                'batch_size': result['batch size'],
                'run_type': batch_labels[i],
                'total_time': result['total cpu time (second)'],
                'throughput': result['throughput(bps)'],
                'avg_time_per_batch': result['execution time per batch (second)'] * 1000,
                'num_batches': result['number of batches'],
                'mae': result['MAE'],
                'r2_score': result['R2'],
                'mse': result['MSE'],
                'bias': result['Bias'],
                'precision': result['Precision'],
                'device': result['device'],
                'cpu_memory': result['cpu memory (MB)'],
                'gpu_memory': result['gpu memory (MB)']
            }

        df_data.append(data_entry)

df = pd.DataFrame(df_data)
df = df.sort_values('batch_size')

print(f"\nDataFrame created with {len(df)} rows")
print("Batch sizes:", df['batch_size'].tolist())

# comparison plots
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(16, 12))

# execution time vs batch
ax1.plot(df['batch_size'], df['total_time'], 'bo-', linewidth=2, markersize=8)
for i, row in df.iterrows():
    ax1.annotate(f"B{row['batch_size']}", (row['batch_size'], row['total_time']),
                 xytext=(5, 5), textcoords='offset points', fontsize=9)
ax1.set_xlabel('Batch Size')
ax1.set_ylabel('Total Execution Time (seconds)')
ax1.set_title('Execution Time vs Batch Size')
ax1.grid(True, alpha=0.3)
ax1.set_xscale('log')

# throughput vs bs
ax2.plot(df['batch_size'], df['throughput'], 'ro-', linewidth=2, markersize=8)
for i, row in df.iterrows():
    ax2.annotate(f"B{row['batch_size']}", (row['batch_size'], row['throughput']),
                 xytext=(5, 5), textcoords='offset points', fontsize=9)
ax2.set_xlabel('Batch Size')
ax2.set_ylabel('Throughput (bits/second)')
ax2.set_title('Throughput vs Batch Size')
ax2.grid(True, alpha=0.3)
ax2.set_xscale('log')

# average time per batch
ax3.plot(df['batch_size'], df['avg_time_per_batch'], 'go-', linewidth=2, markersize=8)
for i, row in df.iterrows():
    ax3.annotate(f"B{row['batch_size']}", (row['batch_size'], row['avg_time_per_batch']),
                 xytext=(5, 5), textcoords='offset points', fontsize=9)
ax3.set_xlabel('Batch Size')
ax3.set_ylabel('Avg Time per Batch (ms)')
ax3.set_title('Processing Efficiency vs Batch Size')
ax3.grid(True, alpha=0.3)
ax3.set_xscale('log')

# accuracy (R2 Score)
ax4.plot(df['batch_size'], df['r2_score'], 'mo-', linewidth=2, markersize=8)
for i, row in df.iterrows():
    ax4.annotate(f"B{row['batch_size']}", (row['batch_size'], row['r2_score']),
                 xytext=(5, 5), textcoords='offset points', fontsize=9)
ax4.set_xlabel('Batch Size')
ax4.set_ylabel('R² Score')
ax4.set_title('Model Accuracy vs Batch Size')
ax4.grid(True, alpha=0.3)
ax4.set_xscale('log')

plt.tight_layout()
plt.savefig('final_batch_analysis.png', dpi=300, bbox_inches='tight')
plt.show()

# comparison table
print("\n" + "="*100)
print("BATCH SIZE DEPLOYMENT COMPARISON")
print("="*100)

# display table
display_df = df[['batch_size', 'total_time', 'throughput', 'avg_time_per_batch', 'num_batches', 'r2_score', 'mae']].copy()
display_df['total_time'] = display_df['total_time'].round(2)
display_df['throughput'] = display_df['throughput'].astype(int)
display_df['avg_time_per_batch'] = display_df['avg_time_per_batch'].round(2)
display_df['r2_score'] = display_df['r2_score'].round(6)
```

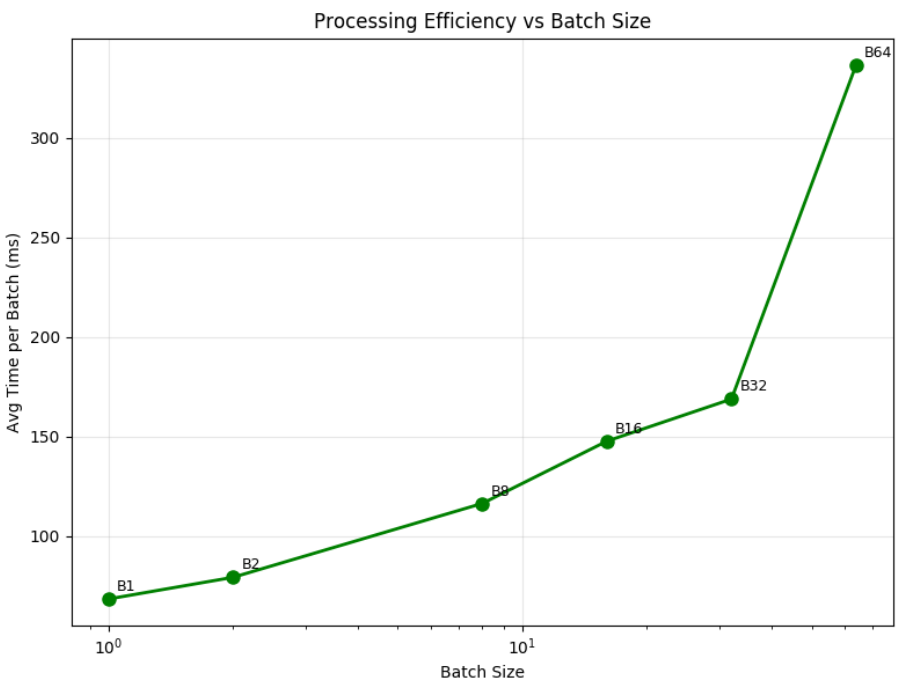
```
display_df['mae'] = display_df['mae'].round(6)

print(display_df.to_string(index=False))

#  performance
print("\n" + "="*60)
print("PERFORMANCE INSIGHTS")
print("="*60)

fastest_idx = df['total_time'].idxmin()
fastest_batch = df.loc[fastest_idx, 'batch_size']
fastest_time = df.loc[fastest_idx, 'total_time']

highest_throughput_idx = df['throughput'].idxmax()
highest_throughput_batch = df.loc[highest_throughput_idx, 'batch_size']
highest_throughput_value = df.loc[highest_throughput_idx, 'throughput']

most_efficient_idx = df['avg_time_per_batch'].idxmin()
most_efficient_batch = df.loc[most_efficient_idx, 'batch_size']
most_efficient_time = df.loc[most_efficient_idx, 'avg_time_per_batch']

best_accuracy_idx = df['r2_score'].idxmax()
best_accuracy_batch = df.loc[best_accuracy_idx, 'batch_size']
best_accuracy_score = df.loc[best_accuracy_idx, 'r2_score']

print(f"Fastest Total Execution: Batch {fastest_batch} ({fastest_time:.2f}s)")
print(f"Highest Throughput: Batch {highest_throughput_batch} ({highest_throughput_value:,} bits/s)")
print(f"Most Efficient per Batch: Batch {most_efficient_batch} ({most_efficient_time:.2f}ms)")
print(f"Best Accuracy: Batch {best_accuracy_batch} (R² = {best_accuracy_score:.6f})")

# trends
print(f"\nPerformance Range:")
print(f"   Execution Time: {df['total_time'].min():.1f}s - {df['total_time'].max():.1f}s")
print(f"   Throughput: {df['throughput'].min():,} - {df['throughput'].max():,} bits/s")
print(f"   Accuracy: {df['r2_score'].min():.6f} - {df['r2_score'].max():.6f}")

# summary
print(f"\n" + "="*70)
print("ASSIGNMENT 4 BASELINE - LOCAL EXECUTION SUMMARY")
print("="*70)

baseline_summary = df[['batch_size', 'total_time', 'throughput', 'r2_score']].copy()
baseline_summary.columns = ['Batch Size', 'Execution Time (s)', 'Throughput (bits/s)', 'R² Score']
baseline_summary['Execution Time (s)'] = baseline_summary['Execution Time (s)'].round(2)
baseline_summary['Throughput (bits/s)'] = baseline_summary['Throughput (bits/s)'].astype(int)
baseline_summary['R² Score'] = baseline_summary['R² Score'].round(6)

print(baseline_summary.to_string(index=False))

# save all
df.to_csv('final_batch_analysis.csv', index=False)
baseline_summary.to_csv('assignment4_baseline.csv', index=False)

with open('final_batch_analysis.json', 'w') as f:
    json.dump(df.to_dict('records'), f, indent=2)


# comparison plots
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(16, 12))

# execution time vs bs
ax1.plot(df['batch_size'], df['total_time'], 'bo-', linewidth=2, markersize=8)
for i, row in df.iterrows():
    ax1.annotate(f"B{row['batch_size']}", (row['batch_size'], row['total_time']),
                 xytext=(5, 5), textcoords='offset points', fontsize=9)
ax1.set_xlabel('Batch Size')
ax1.set_ylabel('Total Execution Time (seconds)')
ax1.set_title('Execution Time vs Batch Size (All Tests)')
ax1.grid(True, alpha=0.3)
ax1.set_xscale('log')

# throughput vs bs
ax2.plot(df['batch_size'], df['throughput'], 'ro-', linewidth=2, markersize=8)
for i, row in df.iterrows():
    ax2.annotate(f"B{row['batch_size']}", (row['batch_size'], row['throughput']),
                 xytext=(5, 5), textcoords='offset points', fontsize=9)
ax2.set_xlabel('Batch Size')
ax2.set_ylabel('Throughput (bits/second)')
ax2.set_title('Throughput vs Batch Size (All Tests)')
ax2.grid(True, alpha=0.3)
ax2.set_xscale('log')

# average time per batch
ax3.plot(df['batch_size'], df['avg_time_per_batch'], 'go-', linewidth=2, markersize=8)
for i, row in df.iterrows():
    ax3.annotate(f"B{row['batch_size']}", (row['batch_size'], row['avg_time_per_batch']),
                 xytext=(5, 5), textcoords='offset points', fontsize=9)
```

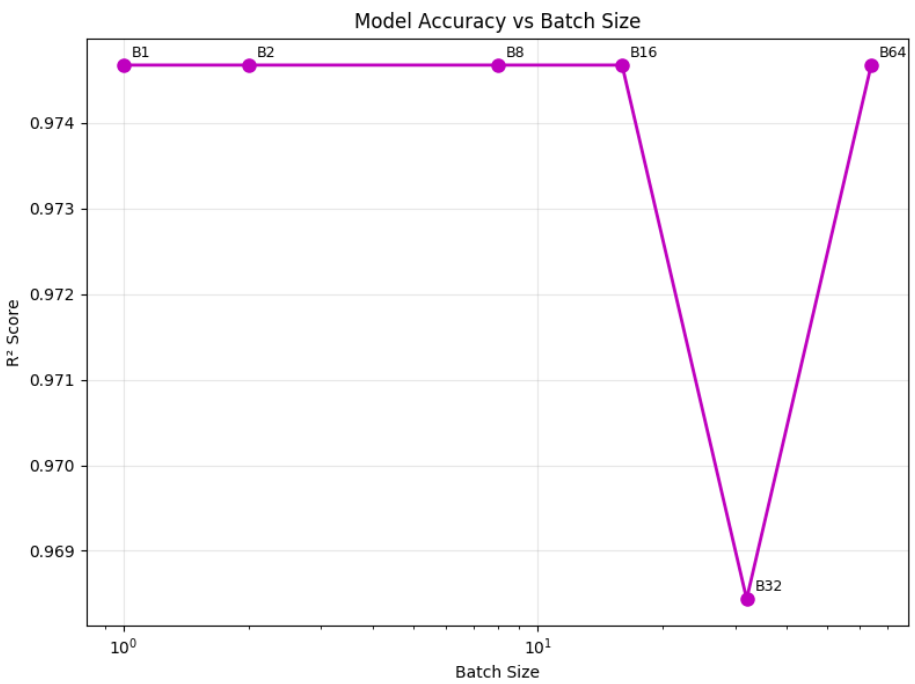
```
ax3.set_xlabel('Batch Size')
ax3.set_ylabel('Avg Time per Batch (ms)')
ax3.set_title('Processing Efficiency vs Batch Size')
ax3.grid(True, alpha=0.3)
ax3.set_xscale('log')

# accuracy (R2 score)
ax4.plot(df['batch_size'], df['r2_score'], 'mo-', linewidth=2, markersize=8)
for i, row in df.iterrows():
    ax4.annotate(f"B{row['batch_size']}", (row['batch_size'], row['r2_score']),
                 xytext=(5, 5), textcoords='offset points', fontsize=9)
ax4.set_xlabel('Batch Size')
ax4.set_ylabel('R² Score')
ax4.set_title('Model Accuracy vs Batch Size')
ax4.grid(True, alpha=0.3)
ax4.set_xscale('log')

plt.tight_layout()
plt.savefig('complete_batch_analysis.png', dpi=300, bbox_inches='tight')
plt.show()

# comparison table
print("\n" + "="*80)
print("COMPLETE BATCH SIZE DEPLOYMENT COMPARISON")
print("Batch Sizes Tested: 1, 2, 8, 16, 32, 64")
print("="*80)

# interesitng metrics
display_df = df[['batch_size', 'total_time', 'throughput', 'avg_time_per_batch', 'num_batches', 'r2_score']].copy()
display_df['total_time'] = display_df['total_time'].round(2)
display_df['throughput'] = display_df['throughput'].round(0)
display_df['avg_time_per_batch'] = display_df['avg_time_per_batch'].round(2)
display_df['r2_score'] = display_df['r2_score'].round(6)

print(display_df.to_string(index=False))

print("\n" + "="*50)
print("PERFORMANCE INSIGHTS")
print("="*50)

if len(df) > 1:
    fastest_idx = df['total_time'].idxmin()
    fastest_batch = df.loc[fastest_idx, 'batch_size']
    fastest_time = df.loc[fastest_idx, 'total_time']

    highest_throughput_idx = df['throughput'].idxmax()
    highest_throughput_batch = df.loc[highest_throughput_idx, 'batch_size']
    highest_throughput_value = df.loc[highest_throughput_idx, 'throughput']

    most_efficient_idx = df['avg_time_per_batch'].idxmin()
    most_efficient_batch = df.loc[most_efficient_idx, 'batch_size']
    most_efficient_time = df.loc[most_efficient_idx, 'avg_time_per_batch']

    best_accuracy_idx = df['r2_score'].idxmax()
    best_accuracy_batch = df.loc[best_accuracy_idx, 'batch_size']
    best_accuracy_score = df.loc[best_accuracy_idx, 'r2_score']

    print(f"Fastest Total Execution: Batch {fastest_batch} ({fastest_time:.2f}s)")
    print(f"Highest Throughput: Batch {highest_throughput_batch} ({highest_throughput_value:,.0f} bits/s)")
    print(f"Most Efficient per Batch: Batch {most_efficient_batch} ({most_efficient_time:.2f}ms)")
    print(f"Best Accuracy: Batch {best_accuracy_batch} (R² = {best_accuracy_score:.6f})")

    # trends
    print(f"\nPerformance Range:")
    print(f"  Execution Time: {df['total_time'].min():.1f}s - {df['total_time'].max():.1f}s")
    print(f"  Throughput: {df['throughput'].min():,.0f} - {df['throughput'].max():,.0f} bits/s")
    print(f"  Accuracy: {df['r2_score'].min():.6f} - {df['r2_score'].max():.6f}")

# summary table for Assignment 4 baseline
summary_table = df[['batch_size', 'total_time', 'throughput', 'r2_score']].copy()
summary_table.columns = ['Batch Size', 'Execution Time (s)', 'Throughput (bits/s)', 'R² Score']

print(f"\n" + "="*60)
print("ASSIGNMENT 4 BASELINE TABLE")
print("="*60)
print(summary_table.to_string(index=False))

# save all
df.to_csv('complete_batch_analysis.csv', index=False)
summary_table.to_csv('assignment4_baseline.csv', index=False)

with open('complete_batch_analysis.json', 'w') as f:
    json.dump(df.to_dict('records'), f, indent=2)
```

```
=== LOADING RESULTS W KEY MAPPING ===
Loaded original batch 32 results
Loaded batch 1 results
Loaded batch 2 results
Loaded batch 8 results
Loaded batch 16 results
Loaded batch 64 results

Successfully loaded 6 result files

DataFrame created with 6 rows
Batch sizes: [1, 2, 8, 16, 32, 64]
```

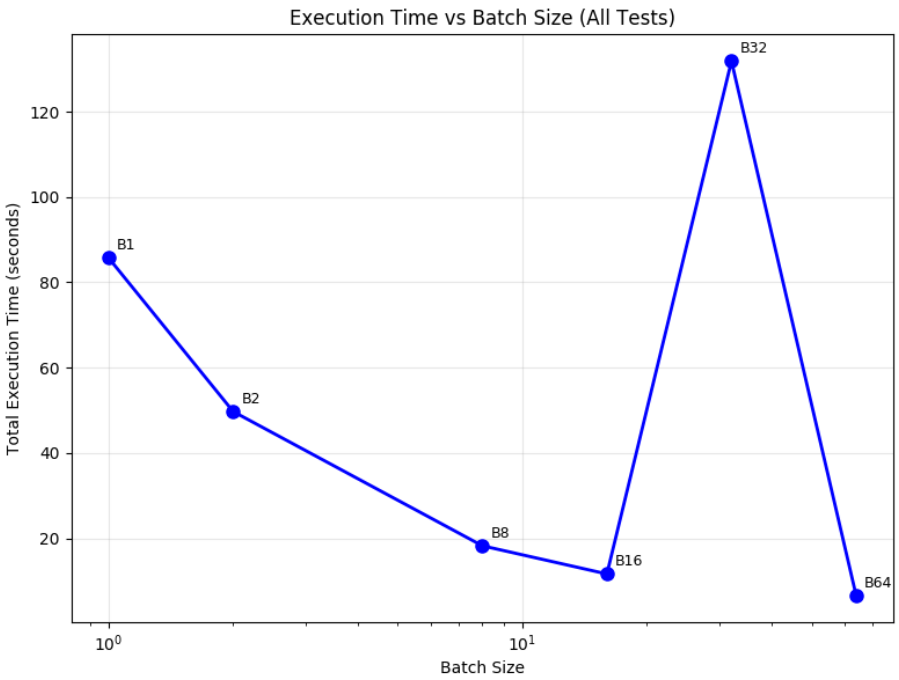


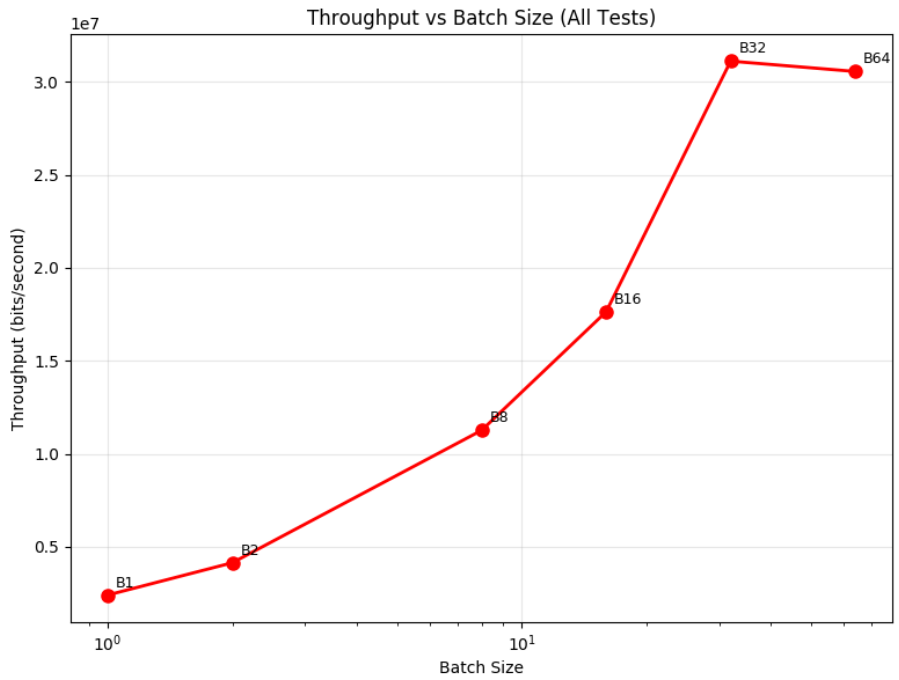


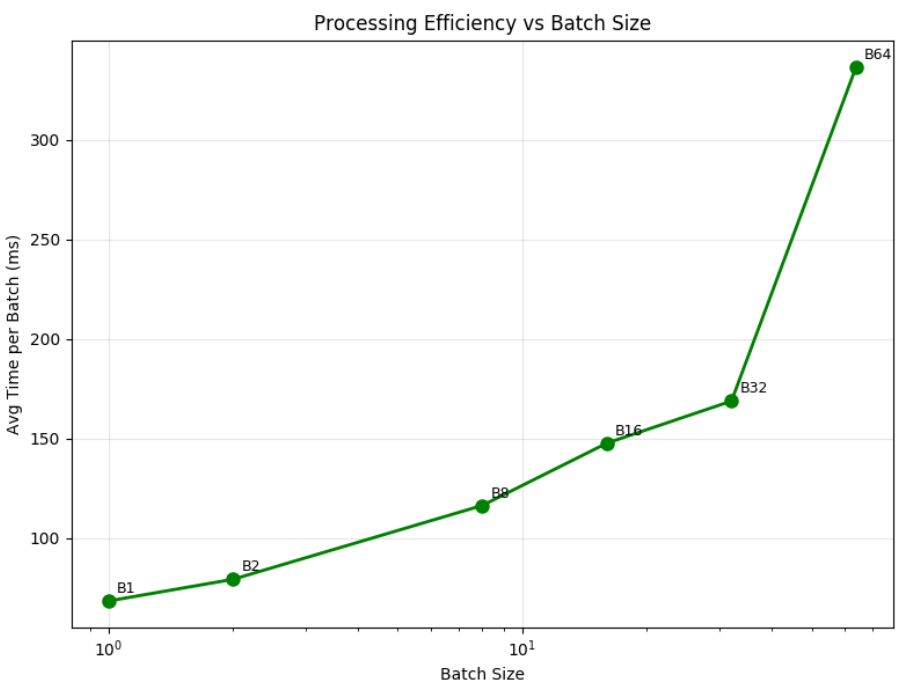


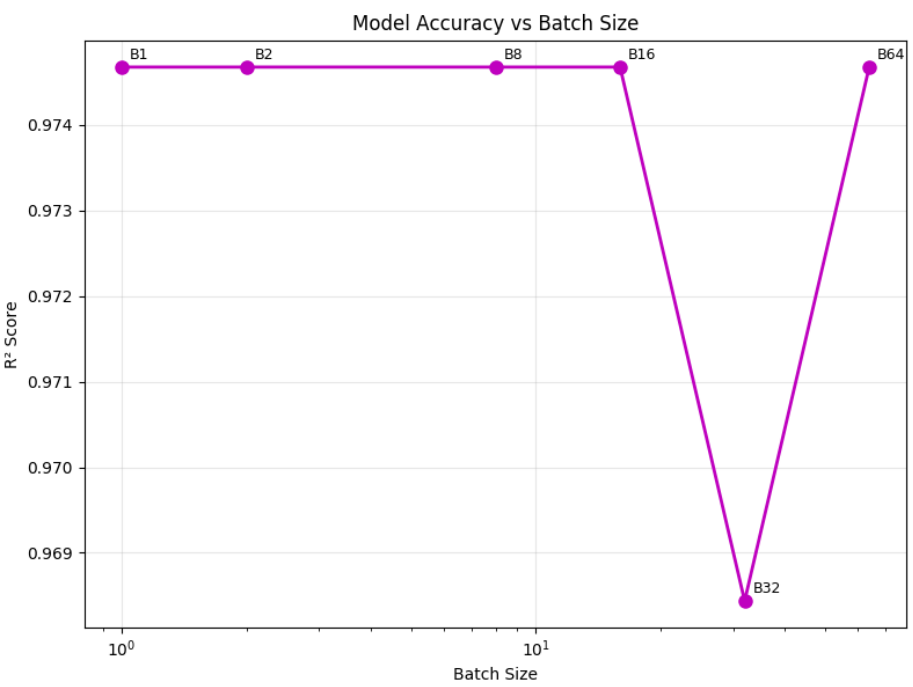



```
================================================================================================
BATCH SIZE DEPLOYMENT COMPARISON
================================================================================================
 batch_size  total_time  throughput  avg_time_per_batch  num_batches  r2_score      mae
          1       85.72     2399538               68.41         1253  0.974674 0.012520
          2       49.73     4136537               79.31          627  0.974674 0.012520
          8       18.27    11255867              116.40          157  0.974674 0.012520
         16       11.67    17631638              147.67           79  0.974674 0.012520
         32      131.84    31118881              168.81          781  0.968438 0.013368
         64        6.73    30562051              336.52           20  0.974674 0.012520

============================================================
PERFORMANCE INSIGHTS
============================================================
Fastest Total Execution: Batch 64 (6.73s)
Highest Throughput: Batch 32 (31,118,881.716076322 bits/s)
Most Efficient per Batch: Batch 1 (68.41ms)
Best Accuracy: Batch 1 (R² = 0.974674)

Performance Range:
   Execution Time: 6.7s - 131.8s
   Throughput: 2,399,538.4318502555 - 31,118,881.716076322 bits/s
   Accuracy: 0.968438 - 0.974674

======================================================================
ASSIGNMENT 4 BASELINE - LOCAL EXECUTION SUMMARY
======================================================================
 Batch Size  Execution Time (s)  Throughput (bits/s)  R² Score
          1               85.72              2399538  0.974674
          2               49.73              4136537  0.974674
          8               18.27             11255867  0.974674
         16               11.67             17631638  0.974674
         32              131.84             31118881  0.968438
         64                6.73             30562051  0.974674
```










```
================================================================================
COMPLETE BATCH SIZE DEPLOYMENT COMPARISON
Batch Sizes Tested: 1, 2, 8, 16, 32, 64
================================================================================
 batch_size  total_time  throughput  avg_time_per_batch  num_batches  r2_score
          1       85.72   2399538.0               68.41         1253  0.974674
          2       49.73   4136537.0               79.31          627  0.974674
          8       18.27  11255867.0              116.40          157  0.974674
         16       11.67  17631639.0              147.67           79  0.974674
         32      131.84  31118882.0              168.81          781  0.968438
         64        6.73  30562052.0              336.52           20  0.974674

================================================
PERFORMANCE INSIGHTS
================================================
Fastest Total Execution: Batch 64 (6.73s)
Highest Throughput: Batch 32 (31,118,882 bits/s)
Most Efficient per Batch: Batch 1 (68.41ms)
Best Accuracy: Batch 1 (R² = 0.974674)

Performance Range:
  Execution Time: 6.7s - 131.8s
  Throughput: 2,399,538 - 31,118,882 bits/s
  Accuracy: 0.968438 - 0.974674

============================================================
ASSIGNMENT 4 BASELINE TABLE
============================================================
 Batch Size  Execution Time (s)  Throughput (bits/s)  R² Score
          1           85.721686         2.399538e+06  0.974674
          2           49.725766         4.136537e+06  0.974674
          8           18.274245         1.125587e+07  0.974674
         16           11.666101         1.763164e+07  0.974674
         32          131.839144         3.111888e+07  0.968438
         64            6.730323         3.056205e+07  0.974674
```

Exploring System-Level Deployment Strategies

To better understand how our fine-tuned model behaves under different system constraints, we conducted a series of controlled deployment experiments. These tests were part of our broader goal to evaluate inference performance across several system-level configurations.

One specific strategy involved manipulating PyTorch's parallel thread usage. We adjusted environment variables to restrict execution to a single thread:

In [ ]:
```
import os
import sys
from datetime import datetime

# limit PyTorch threads for system strat test
os.environ["OMP_NUM_THREADS"] = "1"
os.environ["MKL_NUM_THREADS"] = "1"

# doing a timestamp-based unique folder
timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
label = "threads1_batch32"
results_dir = f"results_{label}_{timestamp}"

# inference path (not sure why this defaulting back)
inference_dir = "/home/sagemaker-user/DS5111_Project/Project/AI-for-Astronomy/code/Anomaly Detection/Inference"

# execute inference
print(f"=== Running inference: {results_dir} ===")
!mkdir -p "{inference_dir}/{results_dir}"
!cd "{inference_dir}" && time {sys.executable} inference.py --batch_size 32 --device cpu --save_path ./{results_dir}/
!cp "{inference_dir}/{results_dir}/Results.json" ./{results_dir}_Results.json
```

```
=== Running inference: results_threads1_batch32_20250721_042442 ===
/home/sagemaker-user/.conda/envs/data_science_on_aws/lib/python3.7/site-packages/torch/autograd/profiler.py:176: UserWarning: C
UDA is not available, disabling CUDA profiling
  warn("CUDA is not available, disabling CUDA profiling")
STAGE:2025-07-21 04:24:45 30193:30193 ActivityProfilerController.cpp:294] Completed Stage: Warm Up
STAGE:2025-07-21 04:24:51 30193:30193 ActivityProfilerController.cpp:300] Completed Stage: Collection

real    0m42.508s
user    0m41.748s
sys     0m0.700s
```

In [ ]:
```
# 4
os.environ["OMP_NUM_THREADS"] = "4"
os.environ["MKL_NUM_THREADS"] = "4"

timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
label = "threads4_batch32"
results_dir = f"results_{label}_{timestamp}"

inference_dir = "/home/sagemaker-user/DS5111_Project/Project/AI-for-Astronomy/code/Anomaly Detection/Inference"
print(f"=== Running inference: {results_dir} ===")
!mkdir -p "{inference_dir}/{results_dir}"
!cd "{inference_dir}" && time {sys.executable} inference.py --batch_size 32 --device cpu --save_path ./{results_dir}/
!cp "{inference_dir}/{results_dir}/Results.json" ./{results_dir}_Results.json
```

```
=== Running inference: results_threads4_batch32_20250721_042747 ===
/home/sagemaker-user/.conda/envs/data_science_on_aws/lib/python3.7/site-packages/torch/autograd/profiler.py:176: UserWarning: C
UDA is not available, disabling CUDA profiling
  warn("CUDA is not available, disabling CUDA profiling")
STAGE:2025-07-21 04:27:49 30576:30576 ActivityProfilerController.cpp:294] Completed Stage: Warm Up
STAGE:2025-07-21 04:27:52 30576:30576 ActivityProfilerController.cpp:300] Completed Stage: Collection

real    0m31.915s
user    0m39.485s
sys     0m1.229s
```

Documentation of any troubleshooting performed

GPU Availability Challenges

The biggest headache we ran into was that our SageMaker instance didn't have any GPU access. We later found out that this was the main point of the assingment after reading assignment 4. The inference script kept throwing warnings about "CUDA is not available, disabling CUDA profiling," which basically meant we were stuck with CPU-only processing when the code was clearly designed to run much faster on GPUs. We had to quickly pivot and explicitly tell the script to use CPU mode with the `--device cpu` flag. This wasn't ideal since CPU processing is way slower, but it was what we had to work with. We checked if we could switch to a GPU instance type, but those weren't available in our setup, so we made the best of the CPU-only situation and used it as our baseline for performance testing.

Memory Problems and Batch Size Trial and Error

The original batch size of 512 was way too ambitious for our system, the process kept getting killed because we were running out of memory. We had about 15GB of RAM to work with, but the large batches were eating it all up. The fix was pretty straightforward but time-consuming: we systematically tried smaller and smaller batch sizes until we found ones that worked. Starting at 512, dropped to 32, then tried 16, 8, 4, and finally 1. We then realized we could just swap to a larger instance size, but we were able to troubleshoot the problem on the original default size. Once we figured out the safe range (1-64), we ran comprehensive tests on six different batch sizes to see how they compared. This wasn't just about fixing the crash - it actually gave us really useful data about how batch size affects performance, memory usage, and processing speed.

File Paths and Data Format Headaches

Getting the file paths to work in the SageMaker environment was another pain point. The default paths in the script assumed a different directory structure, so we had to manually update all the paths to match where our files actually lived. The bigger annoyance came later when we were analyzing all our test results and discovered that the JSON output format was different between our original run and the new test runs. The original results used keys like "total execution time" while the new ones used "total cpu time (second)" - completely different naming. We had to write flexible code that could read both formats and map them correctly, otherwise half our data would have been useless. This kind of format inconsistency is typical when you're running multiple test configurations, but it definitely added extra work to get everything properly compared.

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Getting combined updated metrics for Step 4

In [ ]:
```python
# Batch Size Testing - Series 2
import os
import subprocess
import time
import json
from datetime import datetime
import pandas as pd

class BatchSizeTester:
    def __init__(self, series_name="series_2", inference_path="./ProjectAI-for-Astronomy/code/Anomaly Detection/Inference/infe
        self.series_name = series_name
        self.inference_path = inference_path
        self.base_dir = f"batch_experiments_{series_name}"
        self.log_file = f"{self.base_dir}/experiment_log_{series_name}.txt"
        self.results_summary = {}

        #dir
        os.makedirs(self.base_dir, exist_ok=True)

        # init
        with open(self.log_file, 'w') as f:
            f.write(f"Batch Size Experiment {series_name.upper()} - {datetime.now()}\n")
            f.write("=" * 60 + "\n\n")

    def log_message(self, message, print_also=True):
        """Log message to file and optionally print"""
        timestamp = datetime.now().strftime("%H:%M:%S")
        log_entry = f"[{timestamp}] {message}\n"

        with open(self.log_file, 'a') as f:
            f.write(log_entry)

        if print_also:
            print(f"[{timestamp}] {message}")

    def run_single_test(self, batch_size, additional_args=""):
        """Run a single batch size test"""
        test_name = f"batch_{batch_size}"
        results_dir = f"{self.base_dir}/results_{test_name}"

        self.log_message(f"Starting test: Batch Size {batch_size}")

        os.makedirs(results_dir, exist_ok=True)

        if not os.path.exists(self.inference_path):
            self.log_message(f"ERROR: {self.inference_path} not found!")
            self.results_summary[batch_size] = {
                'status': 'file_not_found',
                'execution_time': 0,
                'results_file': None,
                'error': f'{self.inference_path} not found'
            }
            return self.results_summary[batch_size]

        cmd = [
            "python", self.inference_path,
            "--batch_size", str(batch_size),
            "--device", "cpu",
            "--save_path", f"./{results_dir}/"
        ]
```

```
        if additional_args:
            cmd.extend(additional_args.split())

        try:
            start_time = time.time()

            self.log_message(f"Executing: {' '.join(cmd)}")
            result = subprocess.run(cmd, capture_output=True, text=True, timeout=3600)

            end_time = time.time()
            execution_time = end_time - start_time

            if result.stdout:
                self.log_message(f"STDOUT: {result.stdout[:200]}...", print_also=False)
            if result.stderr:
                self.log_message(f"STDERR: {result.stderr[:200]}...", print_also=True)

            if result.returncode == 0:
                self.log_message(f"Batch size {batch_size} completed successfully in {execution_time:.2f}s")

                source_file = f"{results_dir}/Results.json"
                dest_file = f"{self.base_dir}/results_{self.series_name}_batch_{batch_size}_Results.json"

                if os.path.exists(source_file):
                    import shutil
                    shutil.copy2(source_file, dest_file)
                    self.log_message(f"Results saved to {dest_file}")
                    self.results_summary[batch_size] = {
                        'status': 'success',
                        'execution_time': execution_time,
                        'results_file': dest_file,
                        'stdout': result.stdout[-500:] if result.stdout else "",
                        'stderr': result.stderr[-500:] if result.stderr else ""
                    }
                else:
                    self.log_message(f"Warning: Results.json not found at {source_file}")
                    self.results_summary[batch_size] = {
                        'status': 'no_results',
                        'execution_time': execution_time,
                        'results_file': None,
                        'stdout': result.stdout[-500:] if result.stdout else "",
                        'stderr': result.stderr[-500:] if result.stderr else ""
                    }
            else:
                self.log_message(f"Batch size {batch_size} failed with return code {result.returncode}")
                if result.returncode == 2:
                    self.log_message("Return code 2 usually indicates argument parsing error or missing file")

                self.results_summary[batch_size] = {
                    'status': 'failed',
                    'execution_time': execution_time,
                    'results_file': None,
                    'return_code': result.returncode,
                    'stdout': result.stdout[-500:] if result.stdout else "",
                    'stderr': result.stderr[-500:] if result.stderr else ""
                }

        except subprocess.TimeoutExpired:
            self.log_message(f"Batch size {batch_size} timed out after 1 hour")
            self.results_summary[batch_size] = {
                'status': 'timeout',
                'execution_time': 3600,
                'results_file': None
            }
        except Exception as e:
            self.log_message(f"Batch size {batch_size} failed with exception: {str(e)}")
            self.results_summary[batch_size] = {
                'status': 'exception',
                'execution_time': 0,
                'results_file': None,
                'error': str(e)
            }

        self.log_message("-" * 50)
        return self.results_summary[batch_size]

    def run_batch_series(self, batch_sizes, additional_args=""):
        """Run a series of batch size tests"""
        self.log_message(f"Starting batch series: {batch_sizes}")

        for i, batch_size in enumerate(batch_sizes):
            self.log_message(f"Test {i+1}/{len(batch_sizes)}: Batch Size {batch_size}")
            self.run_single_test(batch_size, additional_args)
```

```
        self.log_message("Batch series completed!")
        return self.results_summary

    def save_summary(self):
        """Save results summary to JSON"""
        summary_file = f"{self.base_dir}/batch_summary_{self.series_name}.json"

        with open(summary_file, 'w') as f:
            json.dump(self.results_summary, f, indent=2, default=str)

        self.log_message(f"Summary saved to {summary_file}")
        return summary_file

    def get_results_dataframe(self):
        """Convert results to pandas DataFrame for easy analysis"""
        data = []
        for batch_size, result in self.results_summary.items():
            data.append({
                'batch_size': batch_size,
                'status': result['status'],
                'execution_time': result.get('execution_time', 0),
                'results_file': result.get('results_file', ''),
                'has_results': result.get('results_file') is not None
            })

        return pd.DataFrame(data).sort_values('batch_size')
```

In [ ]:
```
# init
inference_path = "./AI-for-Astronomy/code/Anomaly Detection/Inference/inference.py"
tester = BatchSizeTester("series_2", inference_path)

# check
if not os.path.exists(inference_path):
    print(f"WARNING: {inference_path} not found!")
    print("Current directory contents:")
    for item in os.listdir("."):
        print(f"  {item}")
else:
    print("inference.py found")
```

```
inference.py found
```

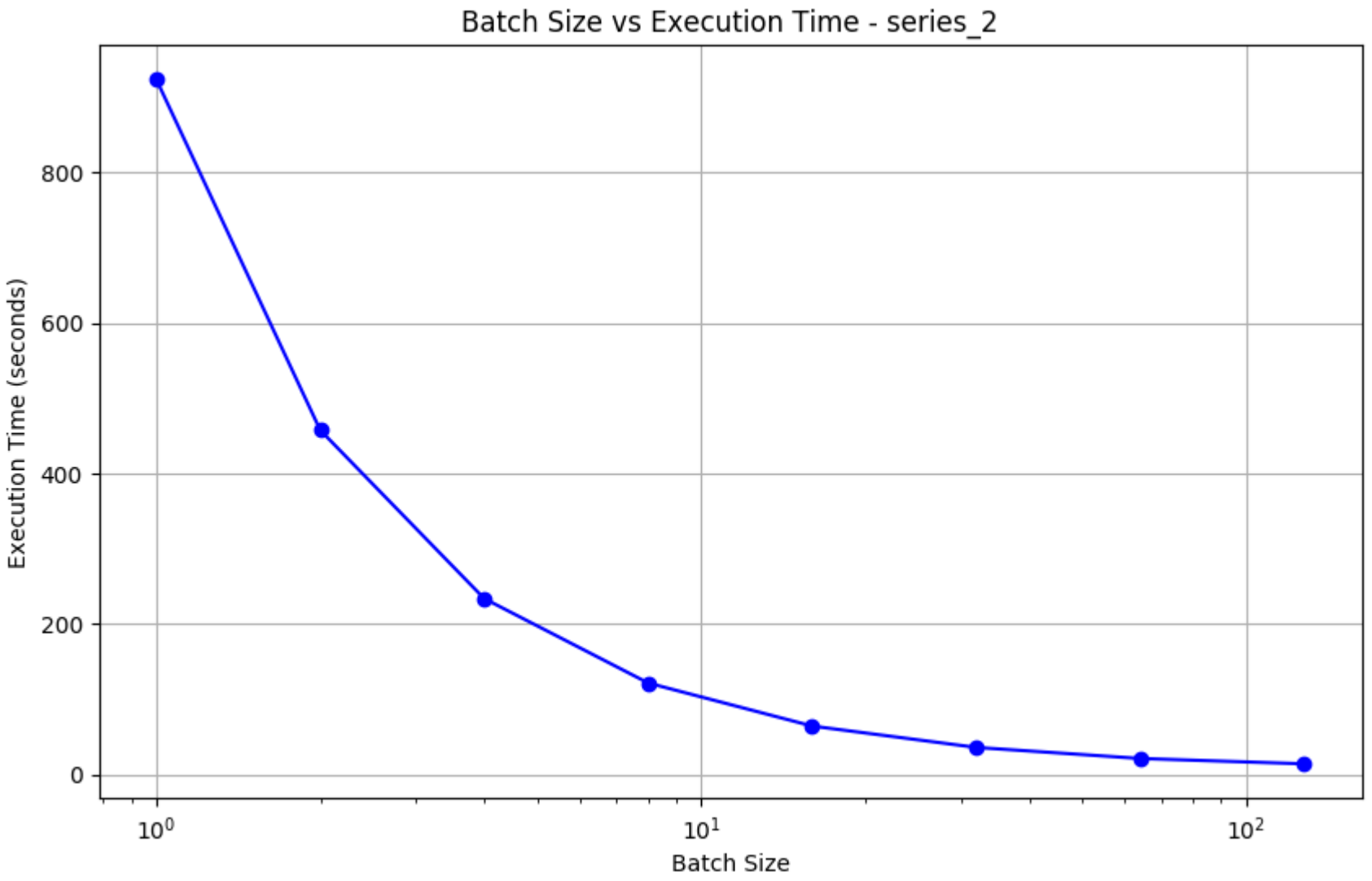
In [ ]:
```
# runtests
batch_sizes_to_test = [1, 2, 4, 8, 16, 32, 64, 128]

results = tester.run_batch_series(batch_sizes_to_test)
```

```
[19:23:15] Starting batch series: [1, 2, 4, 8, 16, 32, 64, 128]
[19:23:15] Test 1/8: Batch Size 1
[19:23:15] Starting test: Batch Size 1
[19:23:15] Executing: python ./AI-for-Astronomy/code/Anomaly Detection/Inference/inference.py --batch_size 1 --device cpu --sav
e_path ./batch_experiments_series_2/results_batch_1/
[19:38:39] STDERR: /home/sagemaker-user/.conda/envs/data_science_on_aws/lib/python3.7/site-packages/torch/autograd/profiler.p
y:176: UserWarning: CUDA is not available, disabling CUDA profiling
  warn("CUDA is not availa...
[19:38:39] Batch size 1 completed successfully in 923.54s
[19:38:39] Results saved to batch_experiments_series_2/results_series_2_batch_1_Results.json
[19:38:39] --------------------------------------------------
[19:38:39] Test 2/8: Batch Size 2
[19:38:39] Starting test: Batch Size 2
[19:38:39] Executing: python ./AI-for-Astronomy/code/Anomaly Detection/Inference/inference.py --batch_size 2 --device cpu --sav
e_path ./batch_experiments_series_2/results_batch_2/
[19:46:16] STDERR: /home/sagemaker-user/.conda/envs/data_science_on_aws/lib/python3.7/site-packages/torch/autograd/profiler.p
y:176: UserWarning: CUDA is not available, disabling CUDA profiling
  warn("CUDA is not availa...
[19:46:16] Batch size 2 completed successfully in 457.52s
[19:46:16] Results saved to batch_experiments_series_2/results_series_2_batch_2_Results.json
[19:46:16] --------------------------------------------------
[19:46:16] Test 3/8: Batch Size 4
[19:46:16] Starting test: Batch Size 4
[19:46:16] Executing: python ./AI-for-Astronomy/code/Anomaly Detection/Inference/inference.py --batch_size 4 --device cpu --sav
e_path ./batch_experiments_series_2/results_batch_4/
[19:50:10] STDERR: /home/sagemaker-user/.conda/envs/data_science_on_aws/lib/python3.7/site-packages/torch/autograd/profiler.p
y:176: UserWarning: CUDA is not available, disabling CUDA profiling
  warn("CUDA is not availa...
[19:50:10] Batch size 4 completed successfully in 233.85s
[19:50:10] Results saved to batch_experiments_series_2/results_series_2_batch_4_Results.json
[19:50:10] --------------------------------------------------
[19:50:10] Test 4/8: Batch Size 8
[19:50:10] Starting test: Batch Size 8
[19:50:10] Executing: python ./AI-for-Astronomy/code/Anomaly Detection/Inference/inference.py --batch_size 8 --device cpu --sav
e_path ./batch_experiments_series_2/results_batch_8/
[19:52:12] STDERR: /home/sagemaker-user/.conda/envs/data_science_on_aws/lib/python3.7/site-packages/torch/autograd/profiler.p
y:176: UserWarning: CUDA is not available, disabling CUDA profiling
  warn("CUDA is not availa...
[19:52:12] Batch size 8 completed successfully in 121.71s
[19:52:12] Results saved to batch_experiments_series_2/results_series_2_batch_8_Results.json
[19:52:12] --------------------------------------------------
[19:52:12] Test 5/8: Batch Size 16
[19:52:12] Starting test: Batch Size 16
[19:52:12] Executing: python ./AI-for-Astronomy/code/Anomaly Detection/Inference/inference.py --batch_size 16 --device cpu --sa
ve_path ./batch_experiments_series_2/results_batch_16/
[19:53:16] STDERR: /home/sagemaker-user/.conda/envs/data_science_on_aws/lib/python3.7/site-packages/torch/autograd/profiler.p
y:176: UserWarning: CUDA is not available, disabling CUDA profiling
  warn("CUDA is not availa...
[19:53:16] Batch size 16 completed successfully in 64.54s
[19:53:16] Results saved to batch_experiments_series_2/results_series_2_batch_16_Results.json
[19:53:16] --------------------------------------------------
[19:53:16] Test 6/8: Batch Size 32
[19:53:16] Starting test: Batch Size 32
[19:53:16] Executing: python ./AI-for-Astronomy/code/Anomaly Detection/Inference/inference.py --batch_size 32 --device cpu --sa
ve_path ./batch_experiments_series_2/results_batch_32/
[19:53:53] STDERR: /home/sagemaker-user/.conda/envs/data_science_on_aws/lib/python3.7/site-packages/torch/autograd/profiler.p
y:176: UserWarning: CUDA is not available, disabling CUDA profiling
  warn("CUDA is not availa...
[19:53:53] Batch size 32 completed successfully in 36.17s
[19:53:53] Results saved to batch_experiments_series_2/results_series_2_batch_32_Results.json
[19:53:53] --------------------------------------------------
[19:53:53] Test 7/8: Batch Size 64
[19:53:53] Starting test: Batch Size 64
[19:53:53] Executing: python ./AI-for-Astronomy/code/Anomaly Detection/Inference/inference.py --batch_size 64 --device cpu --sa
ve_path ./batch_experiments_series_2/results_batch_64/
[19:54:14] STDERR: /home/sagemaker-user/.conda/envs/data_science_on_aws/lib/python3.7/site-packages/torch/autograd/profiler.p
y:176: UserWarning: CUDA is not available, disabling CUDA profiling
  warn("CUDA is not availa...
[19:54:14] Batch size 64 completed successfully in 21.48s
[19:54:14] Results saved to batch_experiments_series_2/results_series_2_batch_64_Results.json
[19:54:14] --------------------------------------------------
[19:54:14] Test 8/8: Batch Size 128
[19:54:14] Starting test: Batch Size 128
[19:54:14] Executing: python ./AI-for-Astronomy/code/Anomaly Detection/Inference/inference.py --batch_size 128 --device cpu --s
ave_path ./batch_experiments_series_2/results_batch_128/
[19:54:28] STDERR: /home/sagemaker-user/.conda/envs/data_science_on_aws/lib/python3.7/site-packages/torch/autograd/profiler.p
y:176: UserWarning: CUDA is not available, disabling CUDA profiling
  warn("CUDA is not availa...
[19:54:28] Batch size 128 completed successfully in 14.39s
[19:54:28] Results saved to batch_experiments_series_2/results_series_2_batch_128_Results.json
[19:54:28] --------------------------------------------------
[19:54:28] Batch series completed!
```

In [ ]:
```
# save
summary_file = tester.save_summary()

# df
```

```
df = tester.get_results_dataframe()
print(df)

successful_tests = df[df['status'] == 'success']
if not successful_tests.empty:
    print(f"Successful tests: {len(successful_tests)}")
    fastest_idx = successful_tests['execution_time'].idxmin()
    fastest_batch = successful_tests.loc[fastest_idx, 'batch_size']
    fastest_time = successful_tests.loc[fastest_idx, 'execution_time']
    print(f"Fastest batch size: {fastest_batch} ({fastest_time:.2f}s)")

    try:
        import matplotlib.pyplot as plt

        plt.figure(figsize=(10, 6))
        plt.plot(successful_tests['batch_size'], successful_tests['execution_time'], 'bo-')
        plt.xlabel('Batch Size')
        plt.ylabel('Execution Time (seconds)')
        plt.title(f'Batch Size vs Execution Time - {tester.series_name}')
        plt.grid(True)
        plt.xscale('log')
        plt.show()
    except ImportError:
        print("Matplotlib not available for plotting")
else:
    print("No successful tests to analyze")
```

```
[20:38:47] Summary saved to batch_experiments_series_2/batch_summary_series_2.json
   batch_size   status  execution_time  \
0           1  success      923.538501
1           2  success      457.516763
2           4  success      233.850199
3           8  success      121.706686
4          16  success       64.538917
5          32  success       36.169455
6          64  success       21.476445
7         128  success       14.390737

                                    results_file  has_results
0  batch_experiments_series_2/results_series_2_ba...         True
1  batch_experiments_series_2/results_series_2_ba...         True
2  batch_experiments_series_2/results_series_2_ba...         True
3  batch_experiments_series_2/results_series_2_ba...         True
4  batch_experiments_series_2/results_series_2_ba...         True
5  batch_experiments_series_2/results_series_2_ba...         True
6  batch_experiments_series_2/results_series_2_ba...         True
7  batch_experiments_series_2/results_series_2_ba...         True
Successful tests: 8
Fastest batch size: 128 (14.39s)
```



In [ ]:
```
def analyze_series_2_results():
    """Analyze results from series 2 batch experiments"""
    results = {}

    result_files = glob.glob("batch_experiments_series_2/results_series_2_batch_*_Results.json")
```

```
for file in result_files:
    match = re.search(r'batch_(\d+)_Results\.json', file)
    if match:
        batch_size = match.group(1)

        try:
            with open(file, 'r') as f:
                data = json.load(f)

            results[int(batch_size)] = {
                'file': file,
                'data': data
            }
        except Exception as e:
            print(f"Error reading {file}: {e}")
    else:
        print(f"Could not extract batch size from filename: {file}")

sorted_results = dict(sorted(results.items()))

print("=" * 100)
print("BATCH SIZE DEPLOYMENT COMPARISON - SERIES 2")
print("=" * 100)

# table
performance_data = []

for batch_size, result in sorted_results.items():
    try:
        data = result['data']
        if data:
            cpu_time = data.get('total cpu time (second)', 0)
            cpu_memory_mb = data.get('cpu memory (MB)', 0)

            # AWS SageMaker ml.t3.2xlarge pricing
            # Instance cost: ~$0.3712 per hour
            # Convert seconds to hours and calculate cost
            cpu_time_hours = cpu_time / 3600
            estimated_cost = cpu_time_hours * 0.3712

            row = {
                'batch_size': batch_size,
                'total_time': cpu_time,
                'throughput': data.get('throughput(bps)', 'N/A'),
                'avg_time_per_batch': data.get('execution time per batch (second)', 'N/A'),
                'num_batches': data.get('number of batches', 'N/A'),
                'r2_score': data.get('R2', 'N/A'),
                'mae': data.get('MAE', 'N/A'),
                'cpu_memory_mb': cpu_memory_mb,
                'mse': data.get('MSE', 'N/A'),
                'bias': data.get('Bias', 'N/A'),
                'precision': data.get('Precision', 'N/A'),
                'estimated_cost_usd': round(estimated_cost, 6)
            }
            performance_data.append(row)
        else:
            row = {
                'batch_size': batch_size,
                'total_time': 'N/A',
                'throughput': 'N/A',
                'avg_time_per_batch': 'N/A',
                'num_batches': 'N/A',
                'r2_score': 'N/A',
                'mae': 'N/A',
                'cpu_memory_mb': 'N/A',
                'mse': 'N/A',
                'bias': 'N/A',
                'precision': 'N/A',
                'estimated_cost_usd': 'N/A'
            }
            performance_data.append(row)

    except Exception as e:
        print(f"Error processing batch size {batch_size}: {e}")

if performance_data:
    # df
    df = pd.DataFrame(performance_data)
    df = df.sort_values('batch_size')

    # format
    pd.set_option('display.max_columns', None)
    pd.set_option('display.width', None)
    pd.set_option('display.max_colwidth', None)

    print(df.to_string(index=False))
```

```
        # save
        csv_file = 'batch_experiments_series_2/series_2_performance_table.csv'
        df.to_csv(csv_file, index=False)
        print(f"\nPerformance table saved to: {csv_file}")

        return df
    else:
        print("No performance data found in result files")
        return None

if os.path.exists("batch_experiments_series_2"):
    df = analyze_series_2_results()

    result_files = glob.glob("batch_experiments_series_2/results_series_2_batch_*_Results.json")
    if result_files:
        print(f"\nSample result file structure from {result_files[0]}:")
        try:
            with open(result_files[0], 'r') as f:
                sample_data = json.load(f)
            print("Available keys:", list(sample_data.keys()) if sample_data else "Empty file")
            if sample_data:
                print("Sample data structure:")
                for key, value in list(sample_data.items())[:5]:
                    print(f"  {key}: {value}")
        except Exception as e:
            print(f"Error reading sample file: {e}")
else:
    print("batch_experiments_series_2 directory not found. Run the tests first.")
```

```
================================================================================================================
BATCH SIZE DEPLOYMENT COMPARISON - SERIES 2
================================================================================================================
 batch_size  total_time   throughput  avg_time_per_batch  num_batches  r2_score     mae  cpu_memory_mb      mse     bias  preci
sion  estimated_cost_usd
          1  112.786112 1.823739e+06            0.090013         1253  0.974674 0.01252   24149.779244 0.000297 0.002024    0.0
1136            0.011630
          2   60.774210 3.384536e+06            0.096929          627  0.974674 0.01252   25377.380952 0.000297 0.002024    0.0
1136            0.006266
          4   36.254240 5.673612e+06            0.115459          314  0.974674 0.01252   25226.688448 0.000297 0.002024    0.0
1136            0.003738
          8   20.955568 9.815648e+06            0.133475          157  0.974674 0.01252   25143.055728 0.000297 0.002024    0.0
1136            0.002161
         16   13.982232 1.471099e+07            0.176990           79  0.974674 0.01252   25126.061396 0.000297 0.002024    0.0
1136            0.001442
         32   10.322370 1.992687e+07            0.258059           40  0.974674 0.01252   25182.747040 0.000297 0.002024    0.0
1136            0.001064
         64    8.425917 2.441188e+07            0.421296           20  0.974674 0.01252   25175.413408 0.000297 0.002024    0.0
1136            0.000869
        128    7.209417 2.853108e+07            0.720942           10  0.974674 0.01252   25215.607820 0.000297 0.002024    0.0
1136            0.000743

Performance table saved to: batch_experiments_series_2/series_2_performance_table.csv

Sample result file structure from batch_experiments_series_2/results_series_2_batch_1_Results.json:
Available keys: ['total cpu time (second)', 'total gpu time (second)', 'execution time per batch (second)', 'cpu memory (MB)',
'gpu memory (MB)', 'throughput(bps)', 'batch size', 'number of batches', 'device', 'MAE', 'MSE', 'Bias', 'Precision', 'R2']
Sample data structure:
  total cpu time (second): 112.786112
  total gpu time (second): 0.0
  execution time per batch (second): 0.09001285873902634
  cpu memory (MB): 24149.779244
  gpu memory (MB): 0.0
```