Contents

1 GPU Compatibility Guide

1.1 The Issue: RTX 5090 with Compute Capability 12.0

Your NVIDIA GeForce RTX 5090 has **compute capability 12.0** (Blackwell architecture), which is very new (released late 2024). TensorFlow 2.21 was built with support for compute capabilities up to **9.0** (Ada/Hopper generation).

When TensorFlow tries to use the GPU, it attempts JIT (Just-In-Time) compilation from PTX intermediate representation, but this fails with: - `CUDA_ERROR_INVALID_PTX` - `CUDA_ERROR_INVALID_HANDLE` - `cuLaunchKernel` errors

1.2 Solutions

1.2.1 Solution 1: Automatic CPU Fallback (Implemented)

The code now automatically detects compute capability 12.0 and falls back to CPU mode. When you import `skol_classifier`, you should see:

```
GPU /physical_device:GPU:0: Compute capability 12.0
WARNING: Compute capability 12.0 may not be fully supported.
         Pre-compiled kernels not available, JIT compilation may fail.
         Falling back to CPU-only mode for stability.
Forced CPU-only mode due to GPU compatibility issues
```

If the auto-detection fails, you'll get a helpful error message.

1.2.2 Solution 2: Manual CPU Mode (Recommended for Jupyter)

If you're using Jupyter notebooks or the auto-detection doesn't work, **manually force CPU mode BEFORE any imports**:

```
import os
os.environ['CUDA_VISIBLE_DEVICES'] = ''  # Force CPU-only mode

# Now import skol_classifier
from skol_classifier import SkolClassifierV2
```

IMPORTANT: This MUST be set before TensorFlow is imported. If you've already imported TensorFlow in your session, you need to restart the kernel/Python session.

1.2.3 Solution 3: Run Scripts with CPU Mode

For standalone scripts, set the environment variable when running:

```
CUDA_VISIBLE_DEVICES='' python your_training_script.py
```

1.2.4 Solution 4: Wait for TensorFlow Update

Monitor TensorFlow releases for compute capability 12.0 support. As of TensorFlow 2.21 (December 2024), this is not yet supported. Future versions may add support for Blackwell architecture.

1.3 Testing

Run the test script to verify the auto-detection:

```
python test_gpu_fallback.py
```

1.4 Performance Note

CPU-only mode will be significantly slower than GPU training, but it will work correctly. For the RNN model: - GPU training: Minutes to hours (depending on dataset size) - CPU training: Hours to days (expect 10-50x slower)

Consider using smaller datasets or reducing model complexity when training on CPU: - Reduce `window_size` (e.g., from 50 to 20) - Reduce `hidden_size` (e.g., from 128 to 64) - Reduce `num_layers` (e.g., from 2 to 1) - Reduce `batch_size` (e.g., from 32 to 16)

1.5 Code Changes

The following files were updated to handle this issue:

1.5.1 skol_classifier/rnn_model.py

1. **Automatic GPU Capability Detection** (lines 32-61):
   - Detects compute capability before attempting to use GPU
   - Automatically falls back to CPU if compute capability >= 10.0
   - Provides clear warning messages
2. **Error Handling in Model Building** (lines 133-148):
   - Catches CUDA errors during model initialization
   - Provides helpful error message with solution
   - Suggests CPU-only mode if GPU fails

1.6 Debugging Commands

Check your GPU compute capability:

```
nvidia-smi --query-gpu=compute_cap --format=csv
```

Check TensorFlow build info:

```
import tensorflow as tf
print(tf.sysconfig.get_build_info())
```

Check if TensorFlow sees GPUs:

```
import tensorflow as tf
print(tf.config.list_physical_devices('GPU'))
```

1.7 Future-Proofing

When newer TensorFlow versions add compute capability 12.0 support, simply: 1. Update TensorFlow: `pip install --upgrade tensorflow[and-cuda]` 2. The auto-detection logic will recognize the support and use GPU

The threshold check (`if major >= 10`) can be adjusted in rnn_model.py:47 if needed.