RTX 5090 sm_120 CUDA Compatibility Guide for Docker ML Workloads

Executive Summary

The NVIDIA RTX 5090 with Blackwell architecture (compute capability 12.0/sm_120) launched January 30, 2025, NVIDIA +3 presenting significant compatibility challenges for ML workloads in Docker containers. MIT HAN Lab +4 **Critical requirement: CUDA 12.8+ and specialized configurations are mandatory**, as current stable ML frameworks lack native sm_120 support. Super User This guide provides comprehensive solutions for running embedding models like BGE-M3 and Multilingual E5 Large effectively on the RTX 5090's 32GB GDDR7 memory. Markaicode +4

1. CUDA Toolkit Requirements for sm_120 Architecture

Minimum and Recommended Versions

- **Minimum**: CUDA Toolkit 12.8 GA (first version with native sm_120 support) GitHub +4
- **Recommended Production**: CUDA 12.9 Update 1 or CUDA 13.0 Update 1
- **Driver Requirements**:
  - Minimum: 570.xxx.xx GitHub +2
  - Current stable: 572.24 (includes critical fixes) Graphics Card Hub
  - Linux: ≥575.57.08 for CUDA 12.9 nvidia
  - Windows: ≥576.57 for CUDA 12.9 nvidia

Compilation Flags for sm_120

bash

```
-gencode=arch=compute_80,code=sm_80 \
-gencode=arch=compute_89,code=sm_89 \
-gencode=arch=compute_90,code=sm_90 \
-gencode=arch=compute_120,code=sm_120 \
-gencode=arch=compute_120,code=compute_120
```

Known Issues (January 2025)

- **PCIe Gen 5 compatibility**: Set motherboard to PCIe Gen 4 mode if experiencing issues (minimal 1% performance impact) Tom's Hardware Graphics Card Hub

- **Driver installation black screens**: Resolved with firmware updates in 572.24 PC Gamer Graphics Card Hub
- **GPU virtualization freezes**: Workaround requires disabling modesetting MIT HAN Lab VideoCardz

2. Docker Configuration Requirements

Essential Software Stack

- **Docker**: Version 19.03+ (for --gpus flag support) Stack Overflow NVIDIA
- **NVIDIA Container Toolkit**: Version 1.17.8+ NVIDIA (critical for sm_120)
- **Base OS**: Ubuntu 20.04+, Ubuntu 24.04 recommended

Installation and Configuration

bash

```
# Install NVIDIA Container Toolkit
curl -fsSL https://nvidia.github.io/libnvidia-container/gpgkey | \
  sudo gpg --dearmor -o /usr/share/keyrings/nvidia-container-toolkit-keyring.gpg

curl -s -L https://nvidia.github.io/libnvidia-container/stable/deb/nvidia-container-toolkit.list | \
  sed 's#deb https://#deb [signed-by=/usr/share/keyrings/nvidia-container-toolkit-keyring.gpg] https://#g' | \
  sudo tee /etc/apt/sources.list.d/nvidia-container-toolkit.list

sudo apt-get update
sudo apt-get install -y nvidia-container-toolkit=1.17.8-1

# Configure Docker runtime
sudo nvidia-ctk runtime configure --runtime=docker
sudo systemctl restart docker
```

Essential Docker Run Parameters

bash

```
docker run -it --rm \
  --gpus all \
  --ipc=host \
  --ulimit memlock=-1 \
  --ulimit stack=67108864 \
  --shm-size=32g \
  -e PYTORCH_CUDA_ALLOC_CONF="max_split_size_mb:512,garbage_collection_threshold:0.8" \
  -e TORCH_CUDA_ARCH_LIST="12.0" \
  -v $(pwd):/workspace \
  [base-image]
```

3. PyTorch and TensorFlow Compatibility

PyTorch Status

Current State: PyTorch 2.7.0 provides Linux support via CUDA 12.8 wheels; (GitHub) Windows lacks stable support (PyTorch) (SaladCloud)

Installation Options:

bash

```
# Linux (Stable)
pip install torch==2.7.0 --index-url https://download.pytorch.org/whl/cu128

# Nightly (partial Windows support)
pip install --pre torch --index-url https://download.pytorch.org/whl/nightly/cu128

# Compatibility Layer (immediate workaround)
pip install rtx50-compat
# In Python:
import rtx50_compat  # MUST import before PyTorch
import torch
```

Build from Source (if needed):

bash

```
export TORCH_CUDA_ARCH_LIST="12.0"
export FORCE_CUDA=1
export CUDA_HOME=/usr/local/cuda
export CUDACXX=$CUDA_HOME/bin/nvcc
python setup.py install
```

TensorFlow Status

Critical: No stable TensorFlow release supports sm_120. Nightly builds (2.20.0.dev) have significant issues including PTX compilation failures and runtime errors. GitHub +3 Not recommended for production use.

4. Transformer Model Issues and Solutions

Common Errors and Fixes

Error: "CUDA error: no kernel image is available for execution on the device" GitHub +7 **Solution**: Use PyTorch 2.7+ or rtx50-compat package GitHub GitHub

Flash Attention Compatibility

Community Build (Recommended):

- Source: loscrossos/lib_flashattention v2.7.4.post1_crossos00 GitHub github
- Supports all RTX 50-series cards GitHub
- Compatible with PyTorch 2.7.0, Python 3.12 github

Alternative: PyTorch native SDPA with PR #145602 adds sm_120 support Hugging Face GitHub

BGE-M3 and E5 Large Optimizations

```python
# Enable Flash Attention 2
model.config.use_flash_attention_2 = True

# Optimal batch sizes for BGE-M3 (567M params)
# Sequence length → Batch size
# 512 → 64
# 1024 → 32
# 2048 → 16
# 4096 → 8

# Memory requirements
# BGE-M3: 2-3GB VRAM (inference), 8-12GB (fine-tuning)
# E5 Large: Use proper prefixes ("query:", "passage:")
```

Mixed Precision Recommendations

- **Use BF16 over FP16** for numerical stability on Blackwell

- Enable TF32: `torch.backends.cuda.matmul.allow_tf32 = True` Hugging Face
- Leverage FP4 support for 2x performance over FP8 MIT HAN Lab +2

5. Optimal Docker Base Images

Recommended Pre-built Images

Best Option: `dconsorte/pytorch-tensorflow-gpu:latest`

- Ubuntu 24.04 + Python 3.11 GitHub github
- CUDA 12.8.1 + cuDNN 9.x GitHub github
- PyTorch 2.6.0.dev with Blackwell support GitHub github
- TensorFlow 2.19.0.dev (limited functionality) GitHub github
- Pre-configured Jupyter Lab GitHub github

Official CUDA Images (requires framework installation):

```bash
nvidia/cuda:12.8.1-base-ubuntu24.04
nvidia/cuda:12.8.1-runtime-ubuntu24.04
nvidia/cuda:12.8.1-devel-ubuntu24.04
```

Custom Dockerfile for Production

```dockerfile
```

```
FROM nvcr.io/nvidia/pytorch:25.04-py3

# Install CUDA 12.8+ for sm_120
RUN wget https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2404/x86_64/cuda-keyring_1.1-1_a
    dpkg -i cuda-keyring_1.1-1_all.deb && \
    apt-get update && \
    apt-get install -y cuda-toolkit-12-8

# Set RTX 5090 environment variables
ENV TORCH_CUDA_ARCH_LIST="12.0"
ENV FORCE_CUDA=1
ENV TORCH_USE_CUDA_DSA=1
ENV CUDA_HOME=/usr/local/cuda
ENV CUDACXX=$CUDA_HOME/bin/nvcc
ENV PATH=$CUDA_HOME/bin:$PATH

# Install PyTorch with CUDA 12.8
RUN pip install torch==2.7.0 torchvision torchaudio --index-url https://download.pytorch.org/whl/cu128

# Install ML libraries
RUN pip install transformers accelerate datasets flash-attn sentence-transformers

# Enable TF32 optimization
RUN echo "import torch; torch.backends.cuda.matmul.allow_tf32 = True; torch.backends.cudnn.allow_tf32 = True"

WORKDIR /workspace
```

6. Memory Management for 32GB VRAM

Optimal Configuration

bash

```
export PYTORCH_CUDA_ALLOC_CONF="max_split_size_mb:512,roundup_power2_divisions:16,garbage_collection_t
```

Markaicode

Python Memory Settings

python

```python
import torch

# Reserve 30GB for models (keep 2GB for system)
torch.cuda.set_per_process_memory_fraction(0.94)
torch.cuda.empty_cache()

# Enable gradient checkpointing for large models
model.gradient_checkpointing_enable()

# Dynamic batch size discovery
def find_optimal_batch_size(model, max_memory_gb=28):
    batch_sizes = [1, 2, 4, 8, 16, 32, 64, 128]
    for batch_size in batch_sizes:
        try:
            # Test forward pass
            dummy_input = torch.randn(batch_size, 512).cuda()
            _ = model(dummy_input)

            memory_used = torch.cuda.memory_allocated() / 1e9
            if memory_used > max_memory_gb:
                return batch_size // 2
            torch.cuda.empty_cache()
        except RuntimeError:
            return batch_size // 2
    return batch_sizes[-1]
```

Model Size Guidelines

Model Size	FP32	FP16	INT8	Recommended Batch Size (2048 tokens)
7B params	28GB	14GB	7GB	16
13B params	52GB	26GB	13GB	8
70B params	280GB	140GB	70GB	Requires quantization + offloading

Memory Profiling

python

```
# PyTorch profiler
with torch.profiler.profile(
    activities=[torch.profiler.ProfilerActivity.CUDA],
    profile_memory=True,
    record_shapes=True
) as prof:
    model(inputs)
prof.export_chrome_trace("memory_trace.json")

# Real-time monitoring
nvidia-smi -l 1 --query-gpu=memory.used,memory.free --format=csv
```

7. Latest NVIDIA Updates (January 2025)

Current Status

- **RTX 5090 Launch**: January 30, 2025 at $1,999 (NVIDIA +3)
- **Architecture**: Blackwell (GB202), 21,760 CUDA cores, (Vast.ai) 575W TDP (Runpod +4)
- **Driver Updates**:
  - 572.16: Launch driver
  - 572.24: Critical stability fixes (current recommended)

Resolution Timeline

- **PyTorch Windows stable**: Expected in 2.7.1 or 2.8.0 (Q1-Q2 2025)
- **TensorFlow stable**: Timeline unclear, significant development needed
- **Flash Attention 3**: Native Blackwell support in development (GitHub)

Performance Expectations

- **LLM Inference**: 180-250 tokens/s (8B model), 120-180 tokens/s (13B model) (HostKey) (GitHub)
- **Training**: ~2,000 samples/hour (7B FP16), ~1,200 samples/hour (13B FP16)
- **vs RTX 4090**: 20-50% improvement in transformer workloads (HostKey +4)

Quick Start Commands

1. Verify RTX 5090 Setup

```
bash
```

```
# Check GPU detection
nvidia-smi
docker run --rm --gpus all nvidia/cuda:12.8.1-base-ubuntu24.04 nvidia-smi

# Test PyTorch compatibility
docker run --rm --gpus all -it dconsorte/pytorch-tensorflow-gpu:latest python -c \
  "import rtx50_compat; import torch; print(f'CUDA available: {torch.cuda.is_available()}'); \
   print(f'Device: {torch.cuda.get_device_name(0)}')"
```

2. Run BGE-M3 or E5 Large

python

```
# In container with proper setup
import rtx50_compat  # If using compatibility layer
import torch
from sentence_transformers import SentenceTransformer

# Load model
model = SentenceTransformer('BAAI/bge-m3')
model = model.cuda()
model.config.use_flash_attention_2 = True

# Configure memory
torch.cuda.set_per_process_memory_fraction(0.94)
os.environ['PYTORCH_CUDA_ALLOC_CONF'] = 'max_split_size_mb:512,garbage_collection_threshold:0.8'

# Process embeddings
sentences = ["example text"] * 16  # Batch of 16
embeddings = model.encode(sentences, batch_size=16, convert_to_tensor=True)
```

3. Docker Compose Configuration

yaml

```yaml
version: '3.8'
services:
  ml-training:
    image: dconsorte/pytorch-tensorflow-gpu:latest
    deploy:
      resources:
        reservations:
          devices:
            - driver: nvidia
              count: 1
              capabilities: [gpu]
    environment:
      - CUDA_VISIBLE_DEVICES=0
      - TORCH_CUDA_ARCH_LIST=12.0
      - PYTORCH_CUDA_ALLOC_CONF=max_split_size_mb:512,garbage_collection_threshold:0.8
    volumes:
      - ./workspace:/workspace
      - ./models:/models
    shm_size: '32gb'
    ulimits:
      memlock: -1
      stack: 67108864
```

Critical Takeaways

1. **Always use CUDA 12.8+ and PyTorch 2.7+** for RTX 5090 support GitHub PyTorch
2. **Import rtx50-compat before PyTorch** as immediate workaround PyPI GitHub
3. **Configure Docker with 32GB shared memory** for large models
4. **Use BF16 mixed precision** for optimal performance/stability balance
5. **Install community Flash Attention builds** for transformer acceleration
6. **Set PCIe to Gen 4 mode** if experiencing compatibility issues
7. **Monitor memory continuously** and use gradient checkpointing for large models Hugging Face
8. **Expect 20-50% performance gains** over RTX 4090 once properly configured HostKey Computer Vision Lab

This configuration enables full utilization of the RTX 5090's capabilities for ML workloads while navigating current compatibility limitations. Markaicode GitHub Regular driver and framework updates throughout 2025 will progressively improve native support.