System-Level Optimizations for Faster Model Loading

Storage Optimizations

1. NVMe SSD Storage

- **Impact**: 50-80% faster loading from disk
- Models load from storage → VRAM
- NVMe: ~3GB/s vs SATA SSD: ~500MB/s

2. RAM Caching

```
# Increase system cache for model files
echo 'vm.vfs_cache_pressure=50' >> /etc/sysctl.conf
echo 'vm.swappiness=10' >> /etc/sysctl.conf
```

3. Model File Location

- Store models on fastest drive
- Use tmpfs for frequently switched models:

```
# Create 6GB tmpfs for model caching
sudo mount -t tmpfs -o size=6G tmpfs /tmp/ollama_cache
# Symlink model directory
ln -s /tmp/ollama_cache ~/.ollama/models
```

Ollama Configuration Optimizations

1. Memory Settings

```
# Increase Ollama's memory limits
export OLLAMA_MAX_LOADED_MODELS=1  # Force unload when switching
export OLLAMA_MAX_QUEUE=1          # Don't queue requests
export OLLAMA_NUM_PARALLEL=1       # Single threaded for memory
```

2. Keep-Alive Tuning

```
# Reduce keep-alive to free memory faster
export OLLAMA_KEEP_ALIVE=30s  # Default is 5m
```

3. GPU Memory Management

```
# Force immediate GPU memory cleanup
export CUDA_CACHE_DISABLE=1
export PYTORCH_CUDA_ALLOC_CONF=max_split_size_mb:512
```

Hardware Tricks

1. GPU Memory Optimization

- Use MSI Afterburner to optimize VRAM timing
- Increase power limit for faster memory transfers
- Clean GPU drivers (older versions sometimes faster)

2. System RAM as VRAM Buffer

- Increase shared GPU memory in BIOS
- Use GPU memory compression if available

3. Background Process Management

```
# Stop unnecessary GPU processes
nvidia-smi --compute-mode=EXCLUSIVE_PROCESS
# Kill background GPU processes
sudo fuser -v /dev/nvidia*
```

Quantization Strategies

1. Use Smaller Quantizations

- `Q4_K_M` (current) vs `Q3_K_M` (30% smaller, 10% quality loss)
- `Q2_K` for ultra-fast switching (50% smaller, 20% quality loss)

2. Create Custom Quantizations

```
# Create Q3_K_M versions for faster loading
ollama create gemma3n:e2b-fast -f Modelfile.q3
ollama create gemma3n:e4b-fast -f Modelfile.q3
```

Predictive Loading

1. Pattern-Based Preloading

- Analyze conversation patterns
- Preload likely next model based on context
- Use conversation history to predict model needs

2. Time-Based Optimization

- Load complex model during low-activity periods
- Switch to simple model during high-frequency interactions
- Background model warming during idle time