Performance Tuning Guide

This guide covers performance optimization for Voyager Evolved on Linux.

System Requirements

Minimum

- 4 CPU cores
- 8GB RAM
- SSD storage
- Linux kernel 5.x+

Recommended

- 8+ CPU cores
- 16GB RAM
- NVMe SSD
- Linux kernel 6.x

Performance Profiles

Voyager Evolved includes three built-in performance profiles:

Fast Profile

```
from voyager.evolved.config import EvolvedConfig
config = EvolvedConfig.create_fast_profile()
```

Settings:
- Observation update: 2.0s (slower)
- Max tracked players: 5
- Pattern recognition: disabled
- Cache: larger (1000 entries)
- Batch size: 20
- Skill retrieval: 3 top results

Use when: Testing, lower-end systems, many agents

Balanced Profile (Default)

```
config = EvolvedConfig()  # Default
```

Settings:
- Observation update: 1.0s
- Max tracked players: 10
- Pattern recognition: enabled
- Cache: 500 entries, 50MB

- Batch size: 10
- Skill retrieval: 5 top results

Use when: Normal operation

Quality Profile

```
config = EvolvedConfig.create_quality_profile()
```

Settings:
- Observation update: 0.5s (faster)
- Max tracked players: 20
- Pattern recognition: enabled
- Cache: smaller (250 entries)
- Skill retrieval: 10 top results

Use when: Best behavior quality, powerful systems

Memory Optimization

LLM Response Cache

```yaml
performance:
  enable_llm_cache: true
  cache_max_entries: 500
  cache_max_size_mb: 50.0
```

Tuning tips:
- Increase `cache_max_entries` for repetitive tasks
- Decrease `cache_max_size_mb` on low-memory systems
- Monitor hit rate with `get_performance_report()`

Memory Limits

```yaml
performance:
  memory_limit_percent: 80.0
  cleanup_interval: 60.0
```

Tuning tips:
- Lower `memory_limit_percent` to 70% on shared systems
- Reduce observation memory: `observation.max_observation_memory: 500`

Linux Memory Settings

```
# Add to ~/.bashrc or /etc/environment
export MALLOC_ARENA_MAX=2

# For glibc malloc tuning
export MALLOC_MMAP_THRESHOLD_=131072
export MALLOC_TRIM_THRESHOLD_=131072
```

CPU Optimization

Async Workers

```yaml
performance:
  enable_async: true
  max_async_workers: 4
```

Tuning tips:

- Set `max_async_workers` to CPU cores / 2
- Reduce to 2 if running multiple agents

Batch Processing

```yaml
performance:
  batch_processing: true
  batch_size: 10
  batch_interval: 1.0
```

Tuning tips:

- Increase `batch_size` for high player counts
- Increase `batch_interval` to reduce CPU usage

Ollama Optimization

Model Selection

Model	RAM	Speed	Quality
`llama2`	4GB	Medium	Good
`mistral`	4GB	Fast	Good
`llama2:13b`	8GB	Slow	Better
`codellama`	4GB	Medium	Best for code

Request Timeout

```yaml
ollama:
  request_timeout: 120  # Increase for slow systems
  temperature: 0.7      # Lower for more consistent results
```

GPU Acceleration

If you have an NVIDIA GPU:

```
# Install CUDA support for Ollama
ollama pull llama2  # Will auto-detect GPU

# Verify GPU usage
nvidia-smi
```

Observation System Tuning

Player Tracking

```yaml
observation:
  update_frequency: 1.0      # Seconds between scans
  max_tracked_players: 10    # Reduce for performance
  observation_radius: 32     # Block radius
```

Pattern Recognition

```yaml
observation:
  enable_pattern_recognition: true  # Set false for speed
  pattern_clustering_eps: 0.5
  min_pattern_samples: 3
```

Memory Decay

```yaml
observation:
  max_observation_memory: 1000
  observation_decay_rate: 0.1  # Higher = faster forgetting
```

Goal System Tuning

Fitness Memory

```yaml
evolutionary_goals:
  fitness_memory_size: 100  # Reduce for speed
  adaptation_rate: 0.05
```

Goal Generation

```yaml
evolutionary_goals:
  goal_mutation_rate: 0.1  # Lower for stability
  max_chain_length: 5      # Reduce for simpler goals
```

Human Behavior Tuning

Disable for Performance

```yaml
human_behavior:
  enable_fatigue: false
  enable_attention_system: false
  enable_emotions: false
```

Light Human Behavior

```yaml
human_behavior:
  thinking_pause_chance: 0.05  # Reduce pauses
  look_around_frequency: 0.1   # Reduce looking
  mistake_chance: 0.02         # Reduce mistakes
```

Monitoring Performance

Real-time Metrics

```python
from voyager.evolved.performance import get_performance_report

report = get_performance_report()
print(f"Cache hit rate: {report['cache']['hit_rate']:.1%}")
print(f"Memory usage: {report['memory']['process_rss_mb']:.0f} MB")
```

Profiling

```yaml
debug:
  enable_profiling: true
  profile_interval: 60.0
```

Log Analysis

```bash
# Monitor log file
tail -f logs/voyager.log | grep -E "(time|cache|memory)"
```

Common Issues

High Memory Usage

1. Reduce cache size
2. Lower `max_observation_memory`
3. Disable pattern recognition
4. Use smaller Ollama model

Slow Response Times

1. Enable LLM caching
2. Use faster Ollama model
3. Increase batch size

4. Reduce skill retrieval count

CPU Spikes

1. Increase `update_frequency`
2. Reduce `max_async_workers`
3. Increase `batch_interval`
4. Disable fatigue/attention systems

Benchmark Your Setup

```python
import time
from voyager.evolved.performance import get_perf_monitor

monitor = get_perf_monitor()

# Run for a while, then check
stats = monitor.get_all_stats()
for name, metric in stats.items():
    print(f"{name}: avg={metric['avg']:.3f}s, max={metric['max']:.3f}s")
```

Architecture Diagram