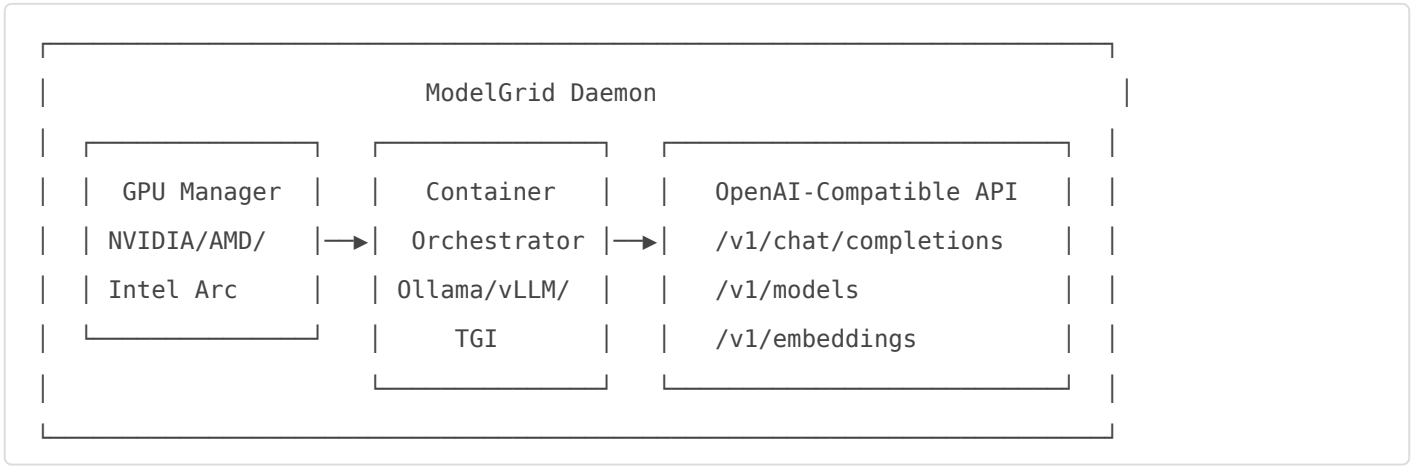
readme.md for @modelgrid.com/modelgrid

□□ ModelGrid

GPU infrastructure management daemon with OpenAI-compatible API for serving AI models in containers.

ModelGrid is a root-level daemon that transforms any GPU-equipped machine into a production-ready AI inference server. It manages Docker containers (Ollama, vLLM, TGI) across NVIDIA, AMD, and Intel GPUs, exposing a unified **OpenAI-compatible API** that works as a drop-in replacement for existing tools.



Issue Reporting and Security

For reporting bugs, issues, or security vulnerabilities, please visit community.foss.global/. This is the central community hub for all issue reporting. Developers who sign and comply with our contribution agreement and go through identification can also get a code.foss.global/ account to submit Pull Requests directly.

□ Features

- □□ **OpenAI-Compatible API** — Drop-in replacement for OpenAI's API. Works with existing tools, SDKs, and applications
- □□ **Multi-GPU Support** — Auto-detect and manage NVIDIA (CUDA), AMD (ROCm), and Intel Arc (oneAPI) GPUs
- □□ **Container Orchestration** — Seamlessly manage Ollama, vLLM, and TGI containers with GPU passthrough
- □□ **Greenlit Models** — Controlled model auto-pulling with VRAM validation for secure deployments
- ⚡ **Streaming Support** — Real-time token streaming via Server-Sent Events
- □□ **Auto-Recovery** — Health monitoring with automatic container restart
- □□ **Docker Native** — Full Docker/Podman integration with isolated networking
- □□ **Prometheus Metrics** — Built-in `/metrics` endpoint for monitoring
- □□ **Cross-Platform** — Pre-compiled binaries for Linux, macOS, and Windows

□□ Installation

Via npm (Recommended)

```
npm install -g @modelgrid.com/modelgrid
```

Via Installer Script

```
curl -sSL https://code.foss.global/modelgrid.com/modelgrid/raw/branch/main/install.sh | sudo bash
```

Manual Binary Download

Download the appropriate binary for your platform from releases:

Platform	Binary
Linux x64	`modelgrid-linux-x64`
Linux ARM64	`modelgrid-linux-arm64`
macOS Intel	`modelgrid-macos-x64`

Platform	Binary
macOS Apple Silicon	`modelgrid-macos-arm64`
Windows x64	`modelgrid-windows-x64.exe`

```
chmod +x modelgrid-linux-x64
sudo mv modelgrid-linux-x64 /usr/local/bin/modelgrid
```

□□ Quick Start

```
# 1. Check your GPUs
sudo modelgrid gpu list

# 2. Initialize configuration
sudo modelgrid config init

# 3. Add an API key
sudo modelgrid config apikey add

# 4. Add a container (interactive)
sudo modelgrid container add

# 5. Enable and start the service
sudo modelgrid service enable
sudo modelgrid service start

# 6. Test the API
curl http://localhost:8080/v1/models \
  -H "Authorization: Bearer YOUR_API_KEY"
```

That's it! Your GPU server is now serving AI models with an OpenAI-compatible API. □□

□□ API Reference

ModelGrid exposes a fully OpenAI-compatible API on port `8080` (configurable).

Authentication

All API endpoints require Bearer token authentication:

```
curl -H "Authorization: Bearer YOUR_API_KEY" http://localhost:8080/v1/models
```

Chat Completions

```
POST /v1/chat/completions
```

```
curl -X POST http://localhost:8080/v1/chat/completions \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{
    "model": "llama3:8b",
    "messages": [
      {"role": "system", "content": "You are a helpful assistant."},
      {"role": "user", "content": "What is machine learning?"}
    ],
    "temperature": 0.7,
    "max_tokens": 1024,
    "stream": false
  }'
```

Streaming Response:

```
curl -X POST http://localhost:8080/v1/chat/completions \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{
    "model": "llama3:8b",
    "messages": [{"role": "user", "content": "Write a poem about AI"}],
    "stream": true
  }' --no-buffer
```

List Models

```
GET /v1/models
```

Returns all available models across all containers:

```
{
  "object": "list",
  "data": [
    {
      "id": "llama3:8b",
      "object": "model",
      "owned_by": "modelgrid",
      "created": 1706745600
    }
  ]
}
```

Embeddings

```
POST /v1/embeddings
```

```
curl -X POST http://localhost:8080/v1/embeddings \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{
    "model": "llama3:8b",
    "input": "Hello, world!"
  }'
```

Health Check (No Auth Required)

```
GET /health
```

```
{
  "status": "ok",
  "uptime": 3600,
  "containers": { "total": 2, "running": 2 },
  "models": 5,
```

```
  "gpus": 2
}
```

Prometheus Metrics (No Auth Required)

```
GET /metrics
```

```
# HELP modelgrid_uptime_seconds Server uptime in seconds
modelgrid_uptime_seconds 3600
# HELP modelgrid_containers_total Total configured containers
modelgrid_containers_total 2
# HELP modelgrid_containers_running Running containers
modelgrid_containers_running 2
```

□□ CLI Commands

Service Management

```
modelgrid service enable       # Install and enable systemd service
modelgrid service disable      # Stop and disable service
modelgrid service start        # Start the daemon
modelgrid service stop         # Stop the daemon
modelgrid service restart      # Restart the daemon
modelgrid service status       # Show service status with GPU/container info
modelgrid service logs         # Tail live service logs
```

GPU Management

```
modelgrid gpu list             # List all detected GPUs with VRAM info
modelgrid gpu status           # Show real-time GPU utilization
modelgrid gpu drivers          # Check driver status for all GPUs
modelgrid gpu install          # Install GPU drivers (interactive)
```

Example output:

```
GPU Devices (2):

┌──────────────┬──────────────────────────┬─────────┬────────────────┬──────────────┐
│ ID           │ Model                    │ VRAM    │ Driver         │ Status       │
├──────────────┼──────────────────────────┼─────────┼────────────────┼──────────────┤
│ nvidia-0     │ NVIDIA RTX 4090          │ 24 GB   │ 535.154.05     │ Ready        │
│ nvidia-1     │ NVIDIA RTX 4090          │ 24 GB   │ 535.154.05     │ In Use       │
└──────────────┴──────────────────────────┴─────────┴────────────────┴──────────────┘
```

Container Management

```
modelgrid container list       # List all configured containers
modelgrid container add        # Interactive container setup wizard
modelgrid container remove ID  # Remove a container
modelgrid container start [ID] # Start container(s)
modelgrid container stop [ID]  # Stop container(s)
modelgrid container logs ID    # Show container logs
```

Model Management

```
modelgrid model list           # List available/loaded models
modelgrid model pull NAME      # Pull a model (must be greenlit)
modelgrid model remove NAME    # Remove a model from container
modelgrid model status         # Show model recommendations with VRAM analysis
modelgrid model refresh        # Refresh greenlist cache
```

Configuration

```
modelgrid config show          # Display current configuration
modelgrid config init          # Initialize default configuration
modelgrid config apikey list   # List configured API keys
modelgrid config apikey add    # Generate and add new API key
modelgrid config apikey remove # Remove an API key
```

Global Options

```
--debug, -d    # Enable debug mode (verbose logging)
--version, -v  # Show version information
--help, -h     # Show help message
```

□□ Supported Containers

Ollama

Best for general-purpose model serving with easy model management.

```
# Add via CLI
sudo modelgrid container add
# Select: ollama

# Or configure directly
{
  "id": "ollama-1",
  "type": "ollama",
  "name": "Ollama Server",
  "gpuIds": ["nvidia-0"],
  "port": 11434
}
```

Supported models: llama3, mistral, codellama, phi, gemma, and 100+ more

vLLM

High-performance inference with PagedAttention and continuous batching.

```
{
  "id": "vllm-1",
  "type": "vllm",
  "name": "vLLM Server",
  "gpuIds": ["nvidia-0", "nvidia-1"],  # Tensor parallelism
  "port": 8000,
  "env": {
    "HF_TOKEN": "your-huggingface-token"  # For gated models
```

```
  }
}
```

Best for: Production workloads, high throughput, multi-GPU setups

TGI (Text Generation Inference)

HuggingFace's production-ready inference server with quantization support.

```
{
  "id": "tgi-1",
  "type": "tgi",
  "name": "TGI Server",
  "gpuIds": ["nvidia-0"],
  "port": 8080,
  "env": {
    "QUANTIZE": "gptq"  # Or: awq, bitsandbytes
  }
}
```

Best for: Quantized models, Flash Attention, HuggingFace ecosystem

□□ GPU Support

NVIDIA (CUDA)

Requirements:

- NVIDIA Driver 470+
- CUDA Toolkit 11.0+
- NVIDIA Container Toolkit (`nvidia-docker2`)

```
# Check status
modelgrid gpu drivers

# Install (Ubuntu/Debian)
sudo apt install nvidia-driver-535 nvidia-container-toolkit
sudo systemctl restart docker
```

AMD (ROCm)

Requirements:

- ROCm 5.0+
- AMD GPU with ROCm support (RX 6000+, MI series)

```
# Install ROCm
wget https://repo.radeon.com/amdgpu-install/latest/ubuntu/jammy/amdgpu-install_6.0.60000-1_all.deb
sudo apt install ./amdgpu-install_6.0.60000-1_all.deb
sudo amdgpu-install --usecase=rocm
```

Intel Arc (oneAPI)

Requirements:

- Intel oneAPI Base Toolkit
- Intel Arc A-series GPU (A770, A750, A380)

```
# Install oneAPI
wget -O- https://apt.repos.intel.com/intel-gpg-keys/GPG-PUB-KEY-INTEL-SW-PRODUCTS.PUB | sudo gpg --dearmor -o /usr/share/keyrings/intel-oneapi-archive-keyring.gpg
echo "deb [signed-by=/usr/share/keyrings/intel-oneapi-archive-keyring.gpg] https://apt.repos.intel.com/oneapi all main" | sudo tee /etc/apt/sources.list.d/intel-oneapi.list
sudo apt update && sudo apt install intel-basekit
```

⚙ Configuration

Configuration is stored at `/etc/modelgrid/config.json`:

```
{
  "version": "1.0",
  "api": {
    "port": 8080,
    "host": "0.0.0.0",
    "apiKeys": ["sk-your-api-key-here"],
```

```
    "cors": false,
    "corsOrigins": ["*"],
    "rateLimit": 60
  },
  "docker": {
    "networkName": "modelgrid",
    "runtime": "docker",
    "socketPath": "/var/run/docker.sock"
  },
  "gpus": {
    "autoDetect": true,
    "assignments": {}
  },
  "containers": [
    {
      "id": "ollama-1",
      "type": "ollama",
      "name": "Primary Ollama",
      "image": "ollama/ollama:latest",
      "gpuIds": ["nvidia-0"],
      "port": 11434,
      "models": [],
      "env": {},
      "volumes": []
    }
  ],
  "models": {
    "greenlistUrl":
"https://code.foss.global/modelgrid.com/model_lists/raw/branch/main/greenlit.json",
    "autoPull": true,
    "defaultContainer": "ollama",
    "autoLoad": ["llama3:8b"]
  },
  "checkInterval": 30000
}
```

Configuration Options

Option	Description	Default
`api.port`	API server port	`8080`
`api.host`	Bind address	`0.0.0.0`
`api.apiKeys`	Valid API keys	`[]`
`api.rateLimit`	Requests per minute	`60`
`docker.runtime`	Container runtime	`docker`
`gpus.autoDetect`	Auto-detect GPUs	`true`
`models.autoPull`	Auto-pull greenlit models	`true`
`models.autoLoad`	Models to preload on start	`[]`
`checkInterval`	Health check interval (ms)	`30000`

□□ Greenlit Models

ModelGrid uses a **greenlist system** for security. Only pre-approved models can be auto-pulled, preventing arbitrary downloads.

Default greenlist includes:

- `llama3.2:1b` (4GB VRAM)
- `llama3.2:3b` (6GB VRAM)
- `llama3:8b` (8GB VRAM)
- `mistral:7b` (8GB VRAM)
- `codellama:7b` (8GB VRAM)

Custom greenlist:

```
// greenlit.json
{
  "version": "1.0",
  "lastUpdated": "2026-01-30",
  "models": [
    { "name": "llama3:8b", "container": "ollama", "minVram": 8 },
    { "name": "llama3:70b", "container": "vllm", "minVram": 48 },
    { "name": "mistral:7b-instruct", "container": "ollama", "minVram": 8 }
  ]
}
```

Configure with:

```
{
  "models": {
    "greenlistUrl": "https://your-server.com/greenlit.json"
  }
}
```

□□ Development

Building from Source

```
# Clone repository
git clone https://code.foss.global/modelgrid.com/modelgrid.git
cd modelgrid

# Run directly with Deno
deno run --allow-all mod.ts help

# Run tests
deno task test

# Type check
deno task check

# Compile for current platform
deno compile --allow-all --output modelgrid mod.ts

# Compile for all platforms
deno task compile
```

Project Structure

```
modelgrid/
├── mod.ts                    # Deno entry point
├── ts/
│   ├── cli.ts                # CLI command routing
```

```
│   ├── modelgrid.ts          # Main coordinator class
│   ├── daemon.ts             # Background daemon process
│   ├── systemd.ts            # Systemd service integration
│   ├── constants.ts          # Configuration constants
│   ├── logger.ts             # Logging utilities
│   ├── interfaces/           # TypeScript interfaces
│   ├── hardware/             # GPU detection (NVIDIA/AMD/Intel)
│   ├── drivers/              # Driver management
│   ├── docker/               # Docker management
│   ├── containers/           # Container orchestration
│   │   ├── ollama.ts         # Ollama implementation
│   │   ├── vllm.ts           # vLLM implementation
│   │   └── tgi.ts            # TGI implementation
│   ├── api/                  # OpenAI-compatible API
│   │   ├── server.ts         # HTTP server
│   │   ├── router.ts         # Request routing
│   │   └── handlers/         # Endpoint handlers
│   ├── models/               # Model management
│   └── cli/                  # CLI handlers
├── test/                     # Test files
├── scripts/                  # Build scripts
└── bin/                      # npm wrapper
```

□□ Uninstallation

```
# Stop and remove service
sudo modelgrid service disable

# Uninstall via script
sudo modelgrid uninstall

# Or manual removal
sudo rm /usr/local/bin/modelgrid
sudo rm -rf /etc/modelgrid
sudo rm -rf /opt/modelgrid
sudo rm /etc/systemd/system/modelgrid.service
sudo systemctl daemon-reload
```

Resources

- **Repository:** https://code.foss.global/modelgrid.com/modelgrid
- **Issues:** https://community.foss.global/
- **Releases:** https://code.foss.global/modelgrid.com/modelgrid/releases

License and Legal Information

This repository contains open-source code licensed under the MIT License. A copy of the license can be found in the LICENSE file.

Please note: The MIT License does not grant permission to use the trade names, trademarks, service marks, or product names of the project, except as required for reasonable and customary use in describing the origin of the work and reproducing the content of the NOTICE file.

Trademarks

This project is owned and maintained by Task Venture Capital GmbH. The names and logos associated with Task Venture Capital GmbH and any related products or services are trademarks of Task Venture Capital GmbH or third parties, and are not included within the scope of the MIT license granted herein.

Use of these trademarks must comply with Task Venture Capital GmbH's Trademark Guidelines or the guidelines of the respective third-party owners, and any usage must be approved in writing. Third-party trademarks used herein are the property of their respective owners and used only in a descriptive manner, e.g. for an implementation of an API or similar.

Company Information

Task Venture Capital GmbH Registered at District Court Bremen HRB 35230 HB, Germany

For any legal inquiries or further information, please contact us via email at hello@task.vc.

By using this repository, you acknowledge that you have read this section, agree to comply with its terms, and understand that the licensing of the code does not imply endorsement by Task Venture Capital GmbH of any derivative works.

Revision #3
Created 2026-03-28 11:09:46 UTC by foss.global Team
Updated 2026-03-28 12:16:23 UTC by foss.global Team