📖 AI Orchestrator - Complete CLI Usage Guide

This comprehensive guide covers all CLI commands, options, and examples for the AI Orchestrator.

⚠️ CRITICAL: Virtual Environment Activation

Before running ANY command, you MUST activate the virtual environment:

```
# Navigate to the project directory
cd ~/ai-orchestrator  # or your installation directory

# Activate the virtual environment
source venv/bin/activate

# Your prompt should change, e.g.: (venv) user@machine:~$

# Now you can run commands
ai-orchestrator --help
```

Alternative: Use the quick-start script (auto-activates venv):

```
./quick-start.sh --help
./quick-start.sh status
```

📋 Table of Contents

Command Overview

Command	Purpose	Example
`run`	Execute task with orchestration	`ai-orchestrator run "Design an API"`
`ask`	Quick query to specific model	`ai-orchestrator ask -m anthropic "Hello"`
`status`	Check configuration	`ai-orchestrator status`
`analyze`	Preview routing without executing	`ai-orchestrator analyze "Build a web app"`
`test-api`	Test API connections	`ai-orchestrator test-api`
`list-models`	List available models	`ai-orchestrator list-models gemini`
`init`	Initialize .env file	`ai-orchestrator init`

Testing AI Models

Quick Model Tests

Test each AI model to verify your API keys are working:

```
# Activate venv first!
source venv/bin/activate

# Test OpenAI (ChatGPT)
ai-orchestrator ask -m openai "Say hello in one word"

# Test Anthropic (Claude)
ai-orchestrator ask -m anthropic "Say hello in one word"

# Test Google (Gemini)
ai-orchestrator ask -m gemini "Say hello in one word"

# Test Moonshot (Kimi)
ai-orchestrator ask -m moonshot "Say hello in one word"
```

Test All APIs at Once

```
ai-orchestrator test-api
```

Expected output:

```
Testing OPENAI...
  ✅ SUCCESS - Response: OK
Testing ANTHROPIC...
  ✅ SUCCESS - Response: OK
Testing GEMINI...
  ✅ SUCCESS - Response: OK
Testing MOONSHOT...
  ⏭️ Skipped (no API key configured)

Summary: 3 passed, 0 failed, 1 skipped
```

Test Specific Model

```
ai-orchestrator test-api -m openai
ai-orchestrator test-api -m anthropic
ai-orchestrator test-api -m gemini
ai-orchestrator test-api -m moonshot
```

Running Tasks

The `run` command is the main entry point for task execution.

Basic Usage

```
ai-orchestrator run "Your task description here"
```

Options

Option	Short	Description
`--model`	`-m`	Force specific model (openai/anthropic/gemini/moonshot)
`--output`	`-o`	Save output to file
`--quiet`	`-q`	Minimal output (response only)
`--debug`	`-d`	Show debug information
`--env`	`-e`	Path to custom .env file

Examples by Model

OpenAI (ChatGPT) - Architecture & System Design

```
# Auto-routed (detected as architecture task)
ai-orchestrator run "Design a microservices architecture for an e-commerce platform"

# Force OpenAI
ai-orchestrator run -m openai "Create a system design for a real-time chat applica-
tion"

# Save output to file
ai-orchestrator run -m openai "Design a REST API schema" -o api-design.md
```

Anthropic (Claude) - Coding & Implementation

```
# Auto-routed (detected as coding task)
ai-orchestrator run "Implement a rate limiter in Python with sliding window"

# Force Claude
ai-orchestrator run -m anthropic "Write a React component for a data table with sort-
ing"

# Quiet mode (just the code)
ai-orchestrator run -m anthropic -q "Write a binary search function in Python"
```

Google (Gemini) - Reasoning & Analysis

```
# Auto-routed (detected as reasoning task)
ai-orchestrator run "Analyze the trade-offs between SQL and NoSQL for this use case"

# Force Gemini
ai-orchestrator run -m gemini "Explain why this algorithm has O(n log n) complexity"

# With debug info
ai-orchestrator run -m gemini -d "Compare REST vs GraphQL"
```

Moonshot (Kimi) - Code Review

```
# Auto-routed (detected as review task)
ai-orchestrator run "Review this code for security vulnerabilities"

# Force Kimi
ai-orchestrator run -m moonshot "Audit this function for best practices"
```

Direct Model Queries (ask command)

The `ask` command provides quick, direct access to a specific model.

Syntax

```
ai-orchestrator ask -m <model> "Your prompt"
```

All Models Examples

```
# OpenAI
ai-orchestrator ask -m openai "Explain microservices in 2 sentences"
ai-orchestrator ask -m openai "What's the best way to structure a Python project?"

# Anthropic
ai-orchestrator ask -m anthropic "Write a Python function to reverse a string"
ai-orchestrator ask -m anthropic "How do I fix a memory leak in Node.js?"

# Gemini
ai-orchestrator ask -m gemini "What is 15% of 280?"
ai-orchestrator ask -m gemini "Explain the time complexity of quicksort"

# Moonshot
ai-orchestrator ask -m moonshot "Review this code: def add(a,b): return a+b"
```

Options

Option	Short	Description
`--model`	`-m`	Model to use (required)
`--quiet`	`-q`	Output only the response
`--debug`	`-d`	Show debug information
`--env`	`-e`	Path to custom .env file

Debug Mode Example

```
ai-orchestrator ask -m anthropic -d "Test prompt"
```

Output:

```
Debug: Config loaded from default locations
Debug: Prompt: Test prompt
Debug: Creating anthropic client...
Debug: Model name: claude-3-5-sonnet-20241022
Debug: Sending request to anthropic...
Debug: Response received
Debug: Success: True
Debug: Content length: 42
Debug: Tokens used: 15
```

Status and Configuration

Check Current Status

```
ai-orchestrator status
```

Output:

```
╔═══════════════════════════════════════════════════════════╗
║          🤖 AI ORCHESTRATOR v1.0.0 🤖                     ║
╚═══════════════════════════════════════════════════════════╝

┌──────────────────────────────────────────────────────────┐
│                Model Configuration Status                │
├──────────┬──────────────────────────┬──────────┬─────────┤
│ Provider │ Model                    │ Status   │ Specialization │
├──────────┼──────────────────────────┼──────────┼─────────┤
│ OpenAI   │ gpt-4o-mini              │ ✓ Configured │ Architecture │
│ Anthropic│ claude-3-5-sonnet...     │ ✓ Configured │ Coding       │
│ Gemini   │ gemini-2.5-flash         │ ✓ Configured │ Reasoning    │
│ Moonshot │ moonshot-v1-8k           │ ✗ Not configured │ Code Review │
└──────────┴──────────────────────────┴──────────┴─────────┘

✓ 3 model(s) ready for use
```

Initialize New Configuration

```
ai-orchestrator init
```

Creates a new `.env` file in the current directory.

Use Custom Configuration

```
ai-orchestrator status -e /path/to/custom.env
ai-orchestrator run -e /path/to/custom.env "Your task"
```

Task Analysis

Preview how a task would be routed without executing it.

Syntax

```
ai-orchestrator analyze "Your task description"
```

Examples

```
# Simple task
ai-orchestrator analyze "Write a Python function"

# Complex task
ai-orchestrator analyze "Design and implement a REST API with authentication"
```

Output:

```
╔═══════════════════════════════════════════════════════════╗
║ Task: Design and implement a REST API with authentication ║
╚═══════════════════════════════════════════════════════════╝

                  Routing Analysis
┌─────┬──────────────┬──────────┬──────────────────────┐
│  #  │ Task Type    │ Target   │ Provider             │
├─────┼──────────────┼──────────┼──────────────────────┤
│  1  │ architecture │ openai   │ OpenAI (ChatGPT)     │
│  2  │ coding       │ anthropic│ Anthropic (Claude)   │
└─────┴──────────────┴──────────┴──────────────────────┘
```

API Testing

Test All Configured Models

```
ai-orchestrator test-api
```

Test Specific Model

```
ai-orchestrator test-api -m openai
ai-orchestrator test-api -m anthropic
ai-orchestrator test-api -m gemini
ai-orchestrator test-api -m moonshot
```

Test All with Custom Config

```
ai-orchestrator test-api -e /path/to/custom.env
```

Model Listing

List available models dynamically from the API.

List Gemini Models

```
ai-orchestrator list-models gemini
```

Output:

```
┌──────────────────────────────────────────────────────────────┐
│                  Available Gemini Models                     │
├──────────────────────┬──────────────────┬──────────┬─────────┤
│ Model Name           │ Display Name     │ Input Tokens│ Output  │
├──────────────────────┼──────────────────┼──────────┼─────────┤
│ gemini-2.5-flash     │ Gemini 2.5 Flash│ 1048576  │ 8192    │
│ gemini-2.5-pro       │ Gemini 2.5 Pro   │ 2097152  │ 8192    │
│ gemini-1.5-pro       │ Gemini 1.5 Pro   │ 2097152  │ 8192    │
└──────────────────────┴──────────────────┴──────────┴─────────┘

✓ Found 3 model(s) available for your API key
Current configured model: gemini-2.5-flash

To use a different model:
  Add to your .env file: GEMINI_MODEL=<model-name>
```

Common Workflows

1. Quick Code Generation

```
source venv/bin/activate
ai-orchestrator ask -m anthropic "Write a function to validate email addresses in Python"
```

2. System Design with Output

```
source venv/bin/activate
ai-orchestrator run -m openai "Design a scalable notification system" -o notification-system.md
```

3. Code Review

```
source venv/bin/activate
ai-orchestrator run -m moonshot "Review this code for security: $(cat myfile.py)"
```

4. Multi-Model Task (Auto-Routed)

```
source venv/bin/activate
ai-orchestrator run "Design and implement a user authentication system with tests"
```

5. Batch Testing

```
source venv/bin/activate

# Test all APIs first
ai-orchestrator test-api

# Then run tasks
ai-orchestrator run "Build a todo app backend"
```

6. Debug Failed Requests

```
source venv/bin/activate
ai-orchestrator ask -m anthropic -d "Your prompt"
# Shows detailed debug info for troubleshooting
```

Tips and Tricks

1. Use Quotes for Complex Tasks

```
# Good
ai-orchestrator run "Design a REST API with user authentication"

# Also good (multi-line with quotes)
ai-orchestrator run "Design a REST API with:
- User authentication
- CRUD operations
- Rate limiting"
```

2. Pipe Output to Other Commands

```
# Save to file
ai-orchestrator ask -m anthropic -q "Write a bash script" > script.sh

# Pipe to clipboard (macOS)
ai-orchestrator ask -m anthropic -q "Write a function" | pbcopy
```

3. Use Quiet Mode for Scripts

```
# In a script, use -q for clean output
CODE=$(ai-orchestrator ask -m anthropic -q "Write a function")
echo "$CODE" > myfunction.py
```

4. Check Status Before Running

```
# Quick sanity check
ai-orchestrator status && ai-orchestrator run "Your task"
```

5. Use the Quick-Start Script

```
# Automatically handles venv activation
./quick-start.sh run "Your task"
./quick-start.sh ask -m openai "Hello"
./quick-start.sh status
```

6. Remember Model Specializations

For This...	Use This Model	Command
Architecture	OpenAI	`-m openai`
Coding	Anthropic	`-m anthropic`
Analysis	Gemini	`-m gemini`
Code Review	Moonshot	`-m moonshot`

Environment Variables

The CLI reads configuration from multiple locations (in order of priority):

1. `~/.config/ai-orchestrator/config.env` (recommended)
2. `~/ai-orchestrator/.env`
3. `./.env` (current directory)

Available Variables

```
# API Keys
OPENAI_API_KEY=sk-...
ANTHROPIC_API_KEY=sk-ant-...
GEMINI_API_KEY=AIza...
MOONSHOT_API_KEY=sk-...

# Model Names (optional overrides)
OPENAI_MODEL=gpt-4o-mini
ANTHROPIC_MODEL=claude-3-5-sonnet-20241022
GEMINI_MODEL=gemini-2.5-flash
MOONSHOT_MODEL=moonshot-v1-8k

# Development Settings
DEBUG=false
LOG_LEVEL=INFO
```

Troubleshooting

Command Not Found

```
# Solution: Activate venv
source venv/bin/activate
```

ModuleNotFoundError

```
# Solution: Activate venv and reinstall
source venv/bin/activate
pip install -e .
```

API Errors

```
# Run with debug flag
ai-orchestrator ask -m anthropic -d "Test"

# Test API connection
ai-orchestrator test-api -m anthropic

# Check status
ai-orchestrator status
```

Empty Response

```
# Try with different prompt
ai-orchestrator ask -m openai "Say hello"

# Check debug output
ai-orchestrator ask -m openai -d "Say hello"
```

See Also

- README.md (README.md) - Project overview
- MANUAL_SETUP.md (MANUAL_SETUP.md) - Manual installation guide
- QUICK_REFERENCE.md (QUICK_REFERENCE.md) - Quick copy-paste commands
- TROUBLESHOOTING.md (TROUBLESHOOTING.md) - Detailed troubleshooting
- MODELS.md (MODELS.md) - Available models documentation

Last Updated: February 2026