Axiomeer Product Guide

AI Marketplace Platform - Complete User Documentation

Axiomeer

February 2026 - Version 6.0

Contents

1 Executive Summary

Axiomeer is an intelligent marketplace platform that enables AI agents to discover, evaluate, and consume tools, datasets, APIs, and AI models through a unified protocol. The platform provides automated product ranking, execution validation, citation verification, and comprehensive audit trails.

This guide provides complete documentation for deploying, configuring, and using Axiomeer in production environments.

Key capabilities:

- **Automated product discovery**: AI agents search for capabilities using natural language
- **Intelligent ranking**: Products are scored using weighted algorithms considering capability match, trust, latency, cost, and semantic relevance
- **Execution validation**: Enforces citation requirements and provenance timestamps
- **Audit trails**: Every transaction is logged with full provenance
- **Trust scoring**: Provider reliability is tracked through success rates and citation compliance
- **Semantic search**: Vector embeddings enable intelligent product matching beyond keyword search

Current deployment:

- 60 integrated products across 9 categories
- 52 provider endpoints operational
- 100% free API infrastructure (no paid services required)
- 90.48% benchmark success rate
- Average latency: 1.19 seconds for external API calls

2 Product Overview

2.1 What is Axiomeer

Axiomeer is an open marketplace where AI agents can find and use external resources without manual integration. Instead of hardcoding API connections, agents query the marketplace, receive ranked recommendations, and execute the best match automatically.

The platform handles:

- **Discovery**: Natural language queries are matched to available products
- **Selection**: Weighted ranking considers multiple factors to find the best option
- **Execution**: The marketplace calls provider endpoints and returns results
- **Validation**: Outputs are verified for citations and timestamps
- **Logging**: Complete execution records with provenance data

2.2 Architecture

The system consists of five main components:

2.2.1 1. API Server (FastAPI)

The REST API provides all marketplace operations:

- Product registration and catalog management
- Search and recommendation endpoints
- Execution and validation
- Trust score calculation
- Receipt storage and retrieval

2.2.2 2. Recommendation Engine

The router component ranks products using configurable weights:

- Capability match (0.70 weight)
- Latency penalty (0.20 weight)
- Cost consideration (0.10 weight)
- Trust score (0.15 weight)
- Relevance score (0.25 weight)

Weights are normalized and can be adjusted via environment variables.

2.2.3 3. Semantic Search (Optional)

Vector-based product discovery using:

- sentence-transformers (all-MiniLM-L6-v2 model)
- FAISS vector indexing
- Hybrid scoring (70% semantic + 30% TF-IDF)
- Automatic fallback to keyword search

2.2.4 4. Validation Layer

Output verification checks:

- Citation presence and format
- Timestamp presence and validity
- Evidence quality assessment
- Deterministic trust scoring

2.2.5 5. Storage Layer

SQLite database stores:

- Product catalog (AppListing table)
- Execution receipts (Run table)
- Trust scores and performance metrics

2.3 Technical Specifications

Runtime requirements:

- Python 3.10 or later
- Ollama (for LLM inference)
- Optional: FAISS for semantic search

Database:

- SQLite (development)
- PostgreSQL compatible (production)

API framework:

- FastAPI 0.109+
- Uvicorn ASGI server
- Pydantic v2 for validation

LLM integration:

- Ollama local inference
- Default model: phi3.5:3.8b
- Configurable model selection

Performance:

- Sub-50ms semantic search
- 1-2 second end-to-end query processing
- Concurrent request support
- Response caching (configurable TTL)

3 Getting Started

3.1 Prerequisites

Before installing Axiomeer, ensure your system meets these requirements:

3.1.1 System Requirements

Operating System:

- macOS 10.15 or later
- Linux (Ubuntu 20.04+, Debian 11+, RHEL 8+)
- Windows 10/11 (with WSL2 recommended)

Hardware:

- CPU: 2+ cores recommended
- RAM: 4GB minimum, 8GB recommended
- Storage: 2GB free space
- Network: Internet connection for API calls

3.1.2 Required Software

Python 3.10+

Check your version:

```
python3 --version
```

If Python is not installed, download from https://www.python.org/downloads/

Ollama

Install Ollama for LLM features:

```
# macOS
brew install ollama

# Linux
curl -fsSL https://ollama.com/install.sh | sh

# Windows
# Download from https://ollama.com/download/windows
```

Pull the required model:

```
ollama pull phi3.5:3.8b
```

Verify installation:

```
ollama list
```

Git

Install Git for repository access:

```
# macOS
brew install git

# Ubuntu/Debian
sudo apt-get install git

# RHEL/CentOS
sudo yum install git
```

3.1.3 Optional Software

FAISS (for semantic search)

Install FAISS for vector-based search:

```
pip install faiss-cpu sentence-transformers
```

If you don't install FAISS, the system automatically falls back to TF-IDF keyword matching.

3.2 Installation

3.2.1 Quick Installation

Clone the repository:

```
git clone https://github.com/ujjwalredd/axiomeer.git
cd axiomeer
```

Create virtual environment:

```
python3 -m venv .venv
source .venv/bin/activate  # macOS/Linux
# .venv\Scripts\activate   # Windows
```

Install dependencies:

```
pip install --upgrade pip
pip install -e ".[dev]"
```

Start the server:

```
uvicorn apps.api.main:app --reload
```

Verify installation:

```
curl http://127.0.0.1:8000/health
```

3.2.2 Docker Installation (Recommended for Production)

Create Dockerfile:

```
FROM python:3.10-slim

WORKDIR /app
COPY . /app
```

```
RUN pip install --no-cache-dir -e .

# Pre-download semantic search model (optional)
RUN python -c "from sentence_transformers import SentenceTransformer; SentenceTransformer('all-

EXPOSE 8000

CMD ["uvicorn", "apps.api.main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Build and run:

```
docker build -t axiomeer:latest .
docker run -p 8000:8000 axiomeer:latest
```

3.2.3 Production Deployment

For production environments, use:

```
# Install production dependencies only
pip install -e .

# Run with multiple workers
uvicorn apps.api.main:app \
  --host 0.0.0.0 \
  --port 8000 \
  --workers 4 \
  --log-level info
```

Configure environment variables (see Configuration section).

3.3 First Steps

3.3.1 1. Verify Installation

Check the health endpoint:

```
curl http://127.0.0.1:8000/health
```

Expected response:

```
{"status": "ok"}
```

3.3.2 2. List Available Products

View the product catalog:

```
python -m marketplace.cli apps
```

You should see 60 products listed across categories.

3.3.3 3. Run a Test Query

Execute a simple search:

```
python -m marketplace.cli shop \
  "What is the weather in Indianapolis?" \
  --auto-caps \
  --execute-top
```

This tests the complete pipeline: capability inference, ranking, execution, and validation.

3.3.4 4. View Execution Receipt

Check the audit log:

```
python -m marketplace.cli runs --n 5
```

3.3.5 5. Access API Documentation

Open your browser to:

```
http://127.0.0.1:8000/docs
```

This provides interactive Swagger UI documentation for all endpoints.

4 Configuration

4.1 Environment Variables

All configuration is managed through environment variables. Copy the example file:

```
cp .env.example .env
```

Edit `.env` to customize settings.

4.2 Core Configuration

4.2.1 Database Settings

DATABASE_URL

SQLAlchemy connection string.

- Default: `sqlite:///./marketplace.db`
- Production: `postgresql://user:pass@host:5432/dbname`

Example:

```
DATABASE_URL=sqlite:///./marketplace.db
```

API_BASE_URL

API server base URL for CLI client.

- Default: `http://127.0.0.1:8000`
- Production: Your domain or IP

Example:

```
API_BASE_URL=https://api.axiomeer.com
```

4.2.2 LLM Configuration

OLLAMA_URL

Ollama inference endpoint.

- Default: `http://localhost:11434/api/generate`
- Remote: `http://ollama-server:11434/api/generate`

OLLAMA_TIMEOUT

Request timeout in seconds.

- Default: `60`
- Range: 30-300

ROUTER_MODEL

Model for capability extraction.

- Default: `phi3.5:3.8b`
- Alternatives: `qwen2.5:14b-instruct`, `llama3.1:8b`

SALES_AGENT_MODEL

Model for sales agent recommendations.

- Default: `phi3.5:3.8b`
- Note: Smaller models = faster responses

SALES_AGENT_TIMEOUT

Sales agent request timeout.

- Default: `60`
- Range: 30-300

4.3 Ranking Configuration

4.3.1 Router Weights

These weights control how products are ranked. Values are relative and normalized at runtime.

W_CAP - Capability match weight

- Default: `0.70`
- Importance: High (primary ranking factor)

W_LAT - Latency penalty weight

- Default: `0.20`
- Importance: Medium

W_COST - Cost consideration weight

- Default: `0.10`
- Importance: Low (most products are free)

W_TRUST - Trust score weight

- Default: `0.15`
- Importance: Medium

W_REL - Relevance score weight

- Default: `0.25`
- Importance: Medium-High

4.3.2 Routing Thresholds

MIN_CAP_COVERAGE

Minimum required capability coverage.

- Default: `1.0` (100% coverage required)
- Range: 0.0-1.0
- Note: Lower values allow partial matches

MIN_RELEVANCE_SCORE

Minimum relevance score when no capabilities specified.

- Default: `0.08`
- Range: 0.0-1.0

MIN_TOTAL_SCORE

Minimum overall score for top candidate.

- Default: `0.55`
- Range: 0.0-1.0
- Note: Lower values accept weaker matches

4.3.3 Sales Agent Settings

SALES_AGENT_TOP_K

Maximum candidates passed to sales agent.

- Default: `8`
- Range: 3-20

4.4 Semantic Search Configuration

SEMANTIC_SEARCH_ENABLED

Enable vector-based search.

- Default: `true`
- Options: `true`, `false`
- Note: Requires FAISS installation

SEMANTIC_SEARCH_TOP_K

Number of semantic candidates to retrieve.

- Default: `20`
- Range: 10-50

SEMANTIC_SEARCH_TIMEOUT_MS

Monitoring timeout (not search timeout).

- Default: `1000`
- Range: 500-5000

SEMANTIC_SEARCH_MODEL

Sentence-transformers model name.

- Default: `all-MiniLM-L6-v2`
- Alternatives: `paraphrase-MiniLM-L6-v2`, `all-mpnet-base-v2`

4.5 Cache Configuration

4.5.1 Shop Cache

SHOP_CACHE_TTL_SECONDS

Cache duration for shop responses.

- Default: `30`
- Range: 0-3600
- Note: 0 disables caching

4.5.2 Provider Caches

PROVIDER_CACHE_TTL_WEATHER

Weather data cache.

- Default: `30` (30 seconds)
- Rationale: Weather changes frequently

PROVIDER_CACHE_TTL_FX

Exchange rate cache.

- Default: `60` (1 minute)
- Rationale: Rates update every minute

PROVIDER_CACHE_TTL_WIKI

Wikipedia summary cache.

- Default: `3600` (1 hour)
- Rationale: Content rarely changes

PROVIDER_CACHE_TTL_WIKIDATA

Wikidata cache.

- Default: `3600` (1 hour)

PROVIDER_CACHE_TTL_WIKIDUMPS

Wikipedia dumps cache.

- Default: `3600` (1 hour)

PROVIDER_CACHE_TTL_RESTCOUNTRIES

Country data cache.

- Default: `3600` (1 hour)

PROVIDER_CACHE_TTL_OPENLIB

Open Library cache.

- Default: `3600` (1 hour)

PROVIDER_CACHE_TTL_DICTIONARY

Dictionary cache.

- Default: `86400` (24 hours)
- Rationale: Definitions don't change

4.6 Performance Tuning

4.6.1 For Low Latency

```
# Use smaller models
SALES_AGENT_MODEL=phi3.5:3.8b
ROUTER_MODEL=phi3.5:3.8b

# Aggressive caching
SHOP_CACHE_TTL_SECONDS=60
PROVIDER_CACHE_TTL_WEATHER=60

# Reduce candidates
SALES_AGENT_TOP_K=5
SEMANTIC_SEARCH_TOP_K=15
```

4.6.2 For High Accuracy

```
# Use larger models
SALES_AGENT_MODEL=qwen2.5:14b-instruct
ROUTER_MODEL=qwen2.5:14b-instruct

# Strict thresholds
MIN_CAP_COVERAGE=1.0
MIN_RELEVANCE_SCORE=0.10
MIN_TOTAL_SCORE=0.60

# More candidates
SALES_AGENT_TOP_K=10
SEMANTIC_SEARCH_TOP_K=30
```

4.6.3 For Resource-Constrained Environments

```
# Disable semantic search
SEMANTIC_SEARCH_ENABLED=false

# Minimal models
SALES_AGENT_MODEL=phi3.5:3.8b

# Maximum caching
SHOP_CACHE_TTL_SECONDS=300
```

5 Using the API

5.1 API Endpoints

All endpoints are available at `http://127.0.0.1:8000`. Complete interactive documentation is available at `/docs`.

5.2 Health Check

Endpoint: `GET /health`

Description: Verify server status.

Request:

```
curl http://127.0.0.1:8000/health
```

Response:


```
{
  "status": "ok"
}
```


5.3 Product Catalog

5.3.1 List All Products

Endpoint: `GET /apps`

Description: Retrieve all registered products.

Request:

```
curl http://127.0.0.1:8000/apps
```

Response:


```
[
  {
    "id": "realtime_weather_agent",
    "name": "Realtime Weather Agent v2",
    "description": "Returns current weather with sources and timestamps.",
    "capabilities": ["weather", "realtime", "citations"],
    "freshness": "realtime",
    "citations_supported": true,
    "latency_est_ms": 800,
    "cost_est_usd": 0.0,
    "executor_type": "http_api",
    "executor_url": "http://127.0.0.1:8000/providers/openmeteo_weather",
    "category": "utilities",
    "product_type": "api"
  }
]
```

5.3.2 Get Single Product

Endpoint: `GET /apps/{app_id}`

Description: Retrieve specific product details.

Request:

```
curl http://127.0.0.1:8000/apps/realtime_weather_agent
```

Response:

```
{
  "id": "realtime_weather_agent",
  "name": "Realtime Weather Agent v2",
  "description": "Returns current weather with sources and timestamps.",
  "capabilities": ["weather", "realtime", "citations"],
  "freshness": "realtime",
  "citations_supported": true,
  "latency_est_ms": 800,
  "cost_est_usd": 0.0
}
```

5.3.3 Register Product

Endpoint: `POST /apps`

Description: Register a new product. Returns 409 if ID already exists.

Request:

```
curl -X POST http://127.0.0.1:8000/apps \
  -H "Content-Type: application/json" \
  -d '{
    "id": "my_new_api",
    "name": "My New API",
    "description": "Custom API endpoint",
    "capabilities": ["search", "citations"],
    "freshness": "realtime",
    "citations_supported": true,
    "latency_est_ms": 500,
    "cost_est_usd": 0.0,
    "executor_type": "http_api",
    "executor_url": "http://127.0.0.1:8000/providers/my_api"
  }'
```

Response:

```
{
  "id": "my_new_api",
  "status": "registered"
}
```

5.3.4 Update Product

Endpoint: `PUT /apps/{app_id}`

Description: Update existing product (idempotent upsert).

Request:

```
curl -X PUT http://127.0.0.1:8000/apps/my_new_api \
  -H "Content-Type: application/json" \
  -d '{
    "id": "my_new_api",
    "name": "My Updated API",
    "description": "Updated description",
    "capabilities": ["search", "citations", "realtime"],
    "freshness": "realtime",
    "citations_supported": true,
    "latency_est_ms": 400,
    "cost_est_usd": 0.0,
    "executor_type": "http_api",
    "executor_url": "http://127.0.0.1:8000/providers/my_api"
  }'
```

5.4 Search and Recommendations

5.4.1 Shop for Products

Endpoint: `POST /shop`

Description: Get ranked product recommendations for a task.

Request:

```
curl -X POST http://127.0.0.1:8000/shop \
  -H "Content-Type: application/json" \
  -d '{
    "task": "Get current weather in Indianapolis",
    "required_capabilities": ["weather", "realtime"],
    "constraints": {
      "citations_required": true,
      "freshness": "realtime",
      "max_latency_ms": 2000,
      "max_cost_usd": 0.01
    }
  }'
```

Request Fields:

- `task` (string, required): Natural language description
- `required_capabilities` (array, optional): Capability tags
- `constraints` (object, optional):
  - `citations_required` (boolean): Require citation support
  - `freshness` (string): `static`, `daily`, or `realtime`

- `max_latency_ms` (integer): Maximum acceptable latency
- `max_cost_usd` (float): Maximum acceptable cost

Response:


```
{
  "status": "OK",
  "recommendations": [
    {
      "app_id": "realtime_weather_agent",
      "name": "Realtime Weather Agent v2",
      "score": 0.752,
      "capability_match": 1.0,
      "trust_score": 0.50,
      "relevance_score": 0.40
    }
  ],
  "explanation": [
    "Found 1 app matching requirements",
    "Top recommendation: Realtime Weather Agent v2"
  ],
  "sales_agent": {
    "summary": "Based on your requirements, I recommend Realtime Weather Agent v2.",
    "final_choice": "realtime_weather_agent",
    "recommendations": [
      {
        "app_id": "realtime_weather_agent",
        "name": "Realtime Weather Agent v2",
        "rationale": "Highest score and supports citations, real-time data for weather.",
        "tradeoff": "Estimated latency: 800ms; Estimated cost: $0.0000"
      }
    ]
  },
  "metrics": {
    "total_time_ms": 245,
    "semantic_search_time_ms": 120,
    "tfidf_time_ms": 45,
    "scoring_time_ms": 80,
    "semantic_search_used": true,
    "semantic_search_available": true
  }
}
```


Status Values:

- `OK`: Products found matching requirements
- `NO_MATCH`: No products meet minimum thresholds

5.5 Execution

5.5.1 Execute Product

Endpoint: `POST /execute`

Description: Execute a product, validate output, and log receipt.

Request:

```
curl -X POST http://127.0.0.1:8000/execute \
  -H "Content-Type: application/json" \
  -d '{
    "app_id": "realtime_weather_agent",
    "task": "Current weather in Indianapolis",
    "inputs": {
      "lat": 39.77,
      "lon": -86.16
    },
    "require_citations": true
  }'
```

Request Fields:

- `app_id` (string, required): Product identifier
- `task` (string, required): Task description
- `inputs` (object, optional): Provider-specific parameters
- `require_citations` (boolean, optional): Enforce citation validation

Response (Success):


```
{
  "ok": true,
  "run_id": 42,
  "output": {
    "answer": "At 2026-02-06T19:45, temperature is 1.0 C, weather_code=3, wind_speed=17.6 km/h
    "citations": ["https://open-meteo.com/"],
    "retrieved_at": "2026-02-07T00:49:34.241856+00:00",
    "quality": "verified"
  },
  "provenance": {
    "sources": ["https://open-meteo.com/"],
    "retrieved_at": "2026-02-07T00:49:34.241856+00:00",
    "notes": []
  },
  "latency_ms": 910,
  "validation_errors": []
}
```


Response (Validation Failure):


```
{
  "ok": false,
```

```
  "run_id": 43,
  "output": {
    "answer": "Some answer without citations"
  },
  "provenance": null,
  "latency_ms": 120,
  "validation_errors": [
    "Citations required but missing in output",
    "Timestamp 'retrieved_at' missing"
  ]
}
```


5.6 Execution Receipts

5.6.1 List Recent Receipts

Endpoint: `GET /runs`

Description: Retrieve recent execution receipts (limit 50).

Request:

```
curl http://127.0.0.1:8000/runs
```

Response:


```
[
  {
    "id": 42,
    "app_id": "realtime_weather_agent",
    "task": "Current weather in Indianapolis",
    "ok": true,
    "latency_ms": 910,
    "created_at": "2026-02-07T00:49:34.227835",
    "require_citations": true
  }
]
```


5.6.2 Get Full Receipt

Endpoint: `GET /runs/{run_id}`

Description: Retrieve complete execution receipt including output.

Request:

```
curl http://127.0.0.1:8000/runs/42
```

Response:


```
{
  "id": 42,
  "app_id": "realtime_weather_agent",
  "task": "Current weather in Indianapolis",
```

```
  "ok": true,
  "output": {
    "answer": "At 2026-02-06T19:45, temperature is 1.0 C, weather_code=3, wind_speed=17.6 km/h
    "citations": ["https://open-meteo.com/"],
    "retrieved_at": "2026-02-07T00:49:34.241856+00:00",
    "quality": "verified"
  },
  "validation_errors": [],
  "latency_ms": 910,
  "require_citations": true,
  "created_at": "2026-02-07T00:49:34.227835"
}
```


5.7 Trust Scores

5.7.1 List All Trust Scores

Endpoint: `GET /trust`

Description: Retrieve trust scores for all products.

Request:

```
curl http://127.0.0.1:8000/trust
```

Response:


```
[
  {
    "app_id": "realtime_weather_agent",
    "trust_score": 0.50,
    "total_executions": 10,
    "successful_executions": 9,
    "success_rate": 0.90,
    "avg_latency_ms": 850,
    "p50_latency_ms": 820,
    "p95_latency_ms": 950,
    "citation_pass_rate": 1.0
  }
]
```


5.7.2 Get Product Trust Score

Endpoint: `GET /apps/{app_id}/trust`

Description: Get trust score and performance stats for one product.

Request:

```
curl http://127.0.0.1:8000/apps/realtime_weather_agent/trust
```

Response:

```
{
  "app_id": "realtime_weather_agent",
  "trust_score": 0.50,
  "total_executions": 10,
  "successful_executions": 9,
  "success_rate": 0.90,
  "avg_latency_ms": 850,
  "p50_latency_ms": 820,
  "p95_latency_ms": 950,
  "p99_latency_ms": 980,
  "citation_pass_rate": 1.0,
  "last_execution": "2026-02-07T00:49:34.227835"
}
```


5.8 Semantic Search Status

Endpoint: `GET /semantic-search/stats`

Description: Get semantic search engine status and metrics.

Request:

```
curl http://127.0.0.1:8000/semantic-search/stats
```

Response:


```
{
  "enabled": true,
  "initialized": true,
  "total_products": 60,
  "model": "all-MiniLM-L6-v2",
  "embedding_dim": 384
}
```

6 Using the CLI

The command-line interface provides convenient access to all marketplace operations.

6.1 General Usage

All CLI commands use the format:

```
python -m marketplace.cli <command> [arguments] [flags]
```

The API server must be running at the configured API_BASE_URL.

6.2 Available Commands

6.2.1 apps - List Products

List all registered products in the catalog.

Usage:

```
python -m marketplace.cli apps
```

Output:

```
                    Registered Apps (60 total)

 app_id                    name                          freshness

 realtime_weather_agent    Realtime Weather Agent v2     realtime
 exchange_rates            Exchange Rate Lookup          realtime
 nasa_apod                 NASA APOD                     daily
```

Options:

None. This command displays all products.

6.2.2 shop - Search for Products

Search for products and get ranked recommendations.

Usage:

```
python -m marketplace.cli shop <task> [flags]
```

Arguments:

- `task` (required): Natural language task description

Flags:

–auto-caps

Automatically infer capabilities from task using LLM.

```
python -m marketplace.cli shop "weather forecast" --auto-caps
```

–caps <capability1,capability2>

Manually specify required capabilities (comma-separated).

```
python -m marketplace.cli shop "get data" --caps weather,realtime,citations
```

–freshness <static|daily|realtime>

Require specific data freshness.

```
python -m marketplace.cli shop "currency rates" --freshness realtime
```

–citations

Require citation support (default: true).

```
python -m marketplace.cli shop "research papers" --citations
```

–no-citations

Don't require citations.

```
python -m marketplace.cli shop "random data" --no-citations
```

–max-latency-ms

Maximum acceptable latency.

```
python -m marketplace.cli shop "weather" --max-latency-ms 1000
```

–max-cost-usd

Maximum acceptable cost.

```
python -m marketplace.cli shop "api call" --max-cost-usd 0.01
```

–execute-top

Execute the top recommendation immediately.

```
python -m marketplace.cli shop "weather" --auto-caps --execute-top
```

Examples:

Simple search:

```
python -m marketplace.cli shop "What is the weather in Indianapolis?" --auto-caps
```

Search with constraints:

```
python -m marketplace.cli shop \
  "I need realtime weather with citations for Indianapolis" \
  --auto-caps \
  --freshness realtime \
  --citations \
  --max-latency-ms 2000
```

Search and execute:

```
python -m marketplace.cli shop \
  "What is the currency rate of USD to EUR? Cite sources." \
  --auto-caps \
  --citations \
  --execute-top
```

Output:

```
Auto capabilities: ['weather', 'realtime', 'citations']
Status: OK
Current weather query

                  Top Recommendations

  #   app_id                   name                         score

  1   realtime_weather_agent   Realtime Weather Agent v2   0.752


Recommendation 1: Realtime Weather Agent v2 (realtime_weather_agent)
 - Capability match: 1.00 (covers ['citations', 'realtime', 'weather'])
 - Freshness matches requirement: realtime
 - Supports citations/provenance: yes
 - Trust score: 0.50
 - Relevance score: 0.40
Rationale: App supports required capabilities and provides real-time weather.
Tradeoff: Estimated latency: 800ms; Estimated cost: $0.0000

Execution result:
ok=True
Provenance:
{'sources': ['https://open-meteo.com/'], 'retrieved_at': '2026-02-07T00:49:34.241856+00:00'}
Output:
{'answer': 'At 2026-02-06T19:45, temperature is 1.0 C, weather_code=3, wind_speed=17.6 km/h (O
```

6.2.3 runs - View Execution Receipts

List recent execution receipts.

Usage:

```
python -m marketplace.cli runs [flags]
```

Flags:

–n

Number of recent runs to show (default: 10).

```
python -m marketplace.cli runs --n 20
```

Output:

```
              Recent Runs (top 10)

 id   app_id                   ok     latency_ms   created_at

  7   open_library             True          759   2026-02-07T00:50…
  6   realtime_weather_agent   True          910   2026-02-07T00:49…
  5   exchange_rates           True          311   2026-02-07T00:49…
```

6.2.4 trust - View Trust Scores

Display trust scores for products.

Usage:

```
python -m marketplace.cli trust [app_id]
```

Arguments:

- `app_id` (optional): Specific product to query

Examples:

All trust scores:

```
python -m marketplace.cli trust
```

Specific product:

```
python -m marketplace.cli trust exchange_rates
```

Output:

```
Trust Score: exchange_rates
  Total executions: 25
  Success rate: 96.0%
  Avg latency: 305ms
  P95 latency: 450ms
  Citation pass rate: 100.0%
  Trust score: 0.93
```

6.2.5 publish - Register Product

Publish a product manifest to the marketplace.

Usage:

```
python -m marketplace.cli publish <manifest_path>
```

Arguments:

- `manifest_path` (required): Path to JSON manifest file

Example:

```
python -m marketplace.cli publish manifests/categories/utilities/my_product.json
```

Output:

```
Published: my_product (My Product Name)
```

Notes:

- Publishing is idempotent (safe to run multiple times)
- Manifests are validated before registration
- Invalid manifests are rejected with clear error messages
- All manifests in `manifests/categories/` are auto-loaded on server startup

7 Publishing Products

7.1 Overview

Products are published to the marketplace by creating JSON manifest files. Manifests define product metadata, capabilities, and execution endpoints.

7.2 Manifest Structure

7.2.1 Minimal Example

```
{
  "id": "my_api",
  "name": "My API Product",
  "description": "Brief description of what this does",
  "capabilities": ["search", "citations"],
  "freshness": "realtime",
  "citations_supported": true,
  "latency_est_ms": 500,
  "cost_est_usd": 0.0,
  "executor_type": "http_api",
  "executor_url": "http://127.0.0.1:8000/providers/my_api"
}
```

7.2.2 Complete Example

```
{
  "id": "comprehensive_api",
  "name": "Comprehensive API Example",
  "description": "Detailed description of functionality",
  "category": "knowledge",
  "subcategory": "research",
  "tags": ["academic", "citations", "search"],
  "capabilities": ["search", "citations", "summarize"],
  "freshness": "realtime",
  "citations_supported": true,
  "product_type": "api",
  "latency_est_ms": 1200,
  "cost_est_usd": 0.0,
  "executor_type": "http_api",
  "executor_url": "http://127.0.0.1:8000/providers/comprehensive_api",
  "metadata": {
    "api_version": "v2",
    "rate_limit": "100/hour",
    "documentation_url": "https://example.com/docs"
  }
}
```

7.3 Required Fields

7.3.1 id

Unique identifier for the product.

- Type: string
- Format: lowercase, alphanumeric, underscores allowed
- Example: `"my_weather_api"`

7.3.2 name

Display name shown to users.

- Type: string
- Length: 5-100 characters
- Example: `"Real-time Weather Data API"`

7.3.3 description

Clear explanation of what the product does.

- Type: string
- Length: 10-500 characters
- Best practice: Focus on functionality, not marketing
- Example: `"Returns current weather conditions with temperature, precipitation, and wind data."`

7.3.4 capabilities

Array of capability tags for discovery.

- Type: array of strings
- Common values: `weather`, `finance`, `search`, `citations`, `realtime`, `summarize`, `translate`, `rag`, `docs`
- Example: `["weather", "realtime", "citations"]`

7.3.5 freshness

Data update frequency.

- Type: string
- Values: `static`, `daily`, `realtime`
- Example: `"realtime"`

7.3.6 citations_supported

Whether the product returns citations.

- Type: boolean
- Example: `true`

7.3.7 latency_est_ms

Estimated execution latency in milliseconds.

- Type: integer
- Range: 10-30000
- Example: `800`

7.3.8 cost_est_usd

Estimated cost per execution in USD.

- Type: float
- Range: 0.0+
- Example: `0.0` (free), `0.001` (1/10th of a cent)

7.3.9 executor_type

Execution protocol.

- Type: string
- Currently supported: `http_api`
- Example: `"http_api"`

7.3.10 executor_url

HTTP endpoint the marketplace calls.

- Type: string
- Format: Valid HTTP/HTTPS URL
- Example: `"http://127.0.0.1:8000/providers/my_api"`

7.4 Optional Fields

7.4.1 category

High-level product category.

- Type: string
- Common values: `knowledge`, `financial`, `government_data`, `utilities`, `media`, `ai_models`, `geographic`, `language`
- Example: `"knowledge"`

7.4.2 subcategory

More specific categorization.

- Type: string
- Example: `"academic_research"`

7.4.3 tags

Additional searchable tags.

- Type: array of strings
- Example: `["academic", "peer-reviewed", "citations"]`

7.4.4 product_type

Type of product.

- Type: string
- Values: `api`, `model`, `dataset`, `tool`, `aggregator`
- Example: `"api"`

7.4.5 metadata

Additional flexible metadata.

- Type: object
- Free-form JSON
- Example:

```
{
  "api_version": "v2",
  "rate_limit": "1000/hour",
  "documentation_url": "https://example.com/docs"
}
```

7.5 Publishing Process

7.5.1 1. Create Manifest File

Save manifest to `manifests/categories/<category>/<product_id>.json`

Example:

```
mkdir -p manifests/categories/utilities
nano manifests/categories/utilities/my_weather_api.json
```

7.5.2 2. Validate Manifest

Manifests are validated on publish. Check for:

- All required fields present
- Valid field types
- ID format (lowercase, alphanumeric, underscores)
- Valid URLs
- Capability values

7.5.3 3. Create Provider Endpoint

Implement the HTTP endpoint specified in `executor_url`.

Example endpoint in `apps/api/providers.py`:

```
@router.get("/providers/my_api")
async def my_api_endpoint(
```

```
    query: str = Query(..., description="Search query")
) -> Dict[str, Any]:
    """
    My API endpoint implementation.
    """
    # Call external API
    response = requests.get(
        f"https://external-api.com/search?q={query}",
        timeout=5
    )
    data = response.json()

    # Return standardized format
    return {
        "answer": f"Results for {query}: {data['result']}",
        "citations": [data['source_url']],
        "retrieved_at": datetime.now(timezone.utc).isoformat(),
        "quality": "verified"
    }
```

7.5.4 4. Publish to Marketplace

Option A: Auto-load (recommended)

Place manifest in `manifests/categories/` and restart server:

```
uvicorn apps.api.main:app --reload
```

Option B: Manual publish

```
python -m marketplace.cli publish manifests/categories/utilities/my_weather_api.json
```

7.5.5 5. Verify Registration

```
python -m marketplace.cli apps | grep my_weather_api
```

Or via API:

```
curl http://127.0.0.1:8000/apps/my_weather_api
```

7.6 Provider Response Format

7.6.1 Required Response Fields

All provider endpoints must return:

```
{
  "answer": "The actual response content",
  "citations": ["https://source1.com", "https://source2.com"],
  "retrieved_at": "2026-02-07T00:49:34.241856+00:00"
}
```

7.6.2 Optional Response Fields

```
{
  "answer": "Response content",
  "citations": ["https://source.com"],
  "retrieved_at": "2026-02-07T00:49:34.241856+00:00",
  "quality": "verified",
  "metadata": {
    "provider_version": "1.2",
    "processing_time_ms": 120
  }
}
```

7.6.3 Validation Requirements

When `require_citations` is true:

1. **citations** must be a non-empty array of strings
2. **retrieved_at** must be a non-empty ISO 8601 timestamp
3. **answer** must be a non-empty string

If validation fails, execution is marked `ok: false` with detailed errors.

7.7 Best Practices

7.7.1 Manifest Design

Use clear, descriptive names:

Good: `"NASA Astronomy Picture of the Day"` Bad: `"APOD API"`

Write helpful descriptions:

Good: `"Returns daily astronomy images with scientific explanations from NASA."` Bad: `"NASA API endpoint"`

Choose accurate capabilities:

- Include all relevant capabilities
- Don't include capabilities the product doesn't support
- Use standard capability names for better discovery

Estimate latency accurately:

- Test actual response times
- Include 95th percentile latency
- Account for network overhead

7.7.2 Provider Implementation

Return genuine citations:

- Use actual source URLs
- Include all data sources used
- Don't fabricate or mock citations

Use accurate timestamps:

- Timestamp when data was retrieved
- Use UTC timezone
- ISO 8601 format required

Handle errors gracefully:

```
try:
    response = external_api.get(query)
    return {
        "answer": response.data,
        "citations": [response.source],
        "retrieved_at": datetime.now(timezone.utc).isoformat()
    }
except Exception as e:
    raise HTTPException(
        status_code=502,
        detail=f"External API error: {str(e)}"
    )
```

Implement caching when appropriate:

```
from functools import lru_cache

@lru_cache(maxsize=100)
def cached_api_call(query: str):
    # Expensive API call
    return external_api.get(query)
```

7.7.3 Testing

Before publishing, test:

1. **Endpoint accessibility:**

```
curl http://127.0.0.1:8000/providers/my_api?query=test
```

2. **Response format:** Verify all required fields present
3. **Citation validation:** Ensure citations are real URLs
4. **Error handling:** Test with invalid inputs
5. **Latency:** Measure actual response times
6. **Integration:** Test via `/shop` and `/execute` endpoints

8 Use Cases and Examples

8.1 Use Case 1: Weather Data Retrieval

8.1.1 Scenario

An AI agent needs current weather data for a specific location with citations.

8.1.2 Implementation

Using CLI:

```
python -m marketplace.cli shop \
  "What is the weather in Indianapolis right now? Cite sources." \
  --auto-caps \
  --freshness realtime \
  --citations \
  --execute-top
```

Using API:

```
# Step 1: Search for products
curl -X POST http://127.0.0.1:8000/shop \
  -H "Content-Type: application/json" \
  -d '{
    "task": "Current weather in Indianapolis",
    "required_capabilities": ["weather", "realtime", "citations"],
    "constraints": {
      "citations_required": true,
      "freshness": "realtime",
      "max_latency_ms": 2000
    }
  }'

# Step 2: Execute top recommendation
curl -X POST http://127.0.0.1:8000/execute \
  -H "Content-Type: application/json" \
  -d '{
    "app_id": "realtime_weather_agent",
    "task": "Current weather in Indianapolis",
    "inputs": {"lat": 39.77, "lon": -86.16},
    "require_citations": true
  }'
```

8.1.3 Result

```
{
  "ok": true,
  "output": {
    "answer": "At 2026-02-06T19:45, temperature is 1.0 C, weather_code=3, wind_speed=17.6 km/h
    "citations": ["https://open-meteo.com/"],
```

```
    "retrieved_at": "2026-02-07T00:49:34.241856+00:00",
    "quality": "verified"
  },
  "latency_ms": 910
}
```


8.2 Use Case 2: Financial Data Lookup

8.2.1 Scenario

Get current cryptocurrency prices with source attribution.

8.2.2 Implementation

```
python -m marketplace.cli shop \
  "What is the current price of Bitcoin in USD? Cite sources." \
  --auto-caps \
  --citations \
  --execute-top
```

8.2.3 Result

The system finds CoinGecko or Coinbase endpoints, executes the best match, and returns verified price data with citations.

8.3 Use Case 3: Research Paper Search

8.3.1 Scenario

Find academic papers on a specific topic.

8.3.2 Implementation

```
python -m marketplace.cli shop \
  "Find recent papers about neural networks" \
  --auto-caps \
  --caps search,citations \
  --execute-top
```

8.3.3 Products Matched

- arXiv Scientific Papers
- PubMed Biomedical Literature
- Semantic Scholar

8.3.4 Result

Returns paper listings with titles, abstracts, authors, and DOI links.

8.4 Use Case 4: Multi-Step Workflow

8.4.1 Scenario

Complex workflow requiring multiple API calls.

8.4.2 Implementation

Step 1: Get country data

```
python -m marketplace.cli shop \
  "What is the currency code for Japan?" \
  --auto-caps \
  --execute-top
```

Step 2: Get exchange rate

```
python -m marketplace.cli shop \
  "What is the exchange rate from USD to JPY?" \
  --auto-caps \
  --execute-top
```

8.4.3 Automation

Use Python client:

```
import requests

API_BASE = "http://127.0.0.1:8000"

# Step 1: Find country
shop_response = requests.post(
    f"{API_BASE}/shop",
    json={
        "task": "Get currency code for Japan",
        "required_capabilities": ["search"]
    }
)
top_app = shop_response.json()["recommendations"][0]["app_id"]

execute_response = requests.post(
    f"{API_BASE}/execute",
    json={
        "app_id": top_app,
        "task": "Currency for Japan",
        "inputs": {"country": "Japan"}
    }
)

# Extract currency code from response
currency = extract_currency(execute_response.json()["output"])
```

```
# Step 2: Get exchange rate
shop_fx = requests.post(
    f"{API_BASE}/shop",
    json={
        "task": f"Exchange rate USD to {currency}",
        "required_capabilities": ["finance", "citations"]
    }
)

# Execute exchange rate query
# ... continue workflow
```

8.5 Use Case 5: Validation and Quality Checks

8.5.1 Scenario

Verify that all responses include proper citations and timestamps.

8.5.2 Implementation

```
import requests

def execute_with_validation(app_id, task, inputs):
    """Execute and validate response quality."""

    response = requests.post(
        "http://127.0.0.1:8000/execute",
        json={
            "app_id": app_id,
            "task": task,
            "inputs": inputs,
            "require_citations": True
        }
    )

    result = response.json()

    if not result["ok"]:
        print(f"Validation failed: {result['validation_errors']}")
        return None

    if result["output"]["quality"] != "verified":
        print("Warning: Quality not verified")

    return result["output"]

# Use in workflow
weather_data = execute_with_validation(
```

```
    "realtime_weather_agent",
    "Current weather",
    {"lat": 39.77, "lon": -86.16}
)

if weather_data:
    print(f"Answer: {weather_data['answer']}")
    print(f"Sources: {weather_data['citations']}")
```

8.6 Use Case 6: Trust Score Monitoring

8.6.1 Scenario

Monitor provider reliability over time.

8.6.2 Implementation

```
import requests
import time

def monitor_provider_trust(app_id, interval_seconds=300):
    """Monitor trust score changes over time."""

    while True:
        response = requests.get(
            f"http://127.0.0.1:8000/apps/{app_id}/trust"
        )

        trust_data = response.json()

        print(f"Trust Score: {trust_data['trust_score']:.2f}")
        print(f"Success Rate: {trust_data['success_rate']*100:.1f}%")
        print(f"Avg Latency: {trust_data['avg_latency_ms']}ms")
        print(f"Total Executions: {trust_data['total_executions']}")
        print("---")

        time.sleep(interval_seconds)

# Monitor realtime_weather_agent
monitor_provider_trust("realtime_weather_agent")
```

8.7 Use Case 7: A/B Testing Providers

8.7.1 Scenario

Compare multiple providers for the same capability.

8.7.2 Implementation

```
import requests

def compare_providers(task, capabilities):
    """Compare multiple providers for same task."""

    # Get all matching providers
    shop_response = requests.post(
        "http://127.0.0.1:8000/shop",
        json={
            "task": task,
            "required_capabilities": capabilities
        }
    )

    recommendations = shop_response.json()["recommendations"]

    results = []
    for rec in recommendations[:3]:  # Test top 3
        exec_response = requests.post(
            "http://127.0.0.1:8000/execute",
            json={
                "app_id": rec["app_id"],
                "task": task,
                "require_citations": True
            }
        )

        result = exec_response.json()
        results.append({
            "app_id": rec["app_id"],
            "score": rec["score"],
            "ok": result["ok"],
            "latency_ms": result["latency_ms"],
            "has_citations": len(result.get("output", {}).get("citations", [])) > 0
        })

    return results

# Compare weather providers
comparison = compare_providers(
    "Current weather in New York",
    ["weather", "realtime"]
)

for r in comparison:
    print(f"{r['app_id']}: score={r['score']:.3f}, "
```

```
        f"latency={r['latency_ms']}ms, ok={r['ok']}")
```

9 Troubleshooting

9.1 Common Issues

9.1.1 Server Won't Start

Symptom: Error when running `uvicorn apps.api.main:app`

Possible Causes:

1. Port 8000 already in use
2. Missing dependencies
3. Python version incompatible

Solutions:

Check port availability:

```
lsof -i :8000
# Kill process using port 8000 if needed
kill -9 <PID>
```

Use different port:

```
uvicorn apps.api.main:app --port 8001
```

Reinstall dependencies:

```
pip install --upgrade pip
pip install -e ".[dev]"
```

Verify Python version:

```
python3 --version  # Must be 3.10+
```

9.1.2 Ollama Connection Failed

Symptom: `Connection refused` when accessing `/shop`

Possible Causes:

1. Ollama not running
2. Ollama on different port
3. Model not pulled

Solutions:

Start Ollama:

```
ollama serve
```

Verify Ollama is running:

```
curl http://localhost:11434/api/tags
```

Pull required model:

```
ollama pull phi3.5:3.8b
```

Check Ollama URL in configuration:

```
echo $OLLAMA_URL
# Should be: http://localhost:11434/api/generate
```

9.1.3 Semantic Search Unavailable

Symptom: Warning: `Semantic search unavailable: Missing dependencies`

Cause: FAISS or sentence-transformers not installed

Solutions:

Install dependencies:

```
pip install faiss-cpu sentence-transformers
```

Or disable semantic search:

```
export SEMANTIC_SEARCH_ENABLED=false
```

Restart server after installation.

9.1.4 NO_MATCH Results

Symptom: `/shop` returns `NO_MATCH` status

Possible Causes:

1. No products match required capabilities
2. Thresholds too strict
3. No products registered

Solutions:

Check registered products:

```
python -m marketplace.cli apps
```

Relax thresholds:

```
export MIN_CAP_COVERAGE=0.8
export MIN_RELEVANCE_SCORE=0.05
export MIN_TOTAL_SCORE=0.45
```

Use manual capabilities:

```
python -m marketplace.cli shop "task" --caps weather
# Instead of --auto-caps
```

9.1.5 Validation Errors

Symptom: Execution returns `ok: false` with validation errors

Possible Causes:

1. Provider not returning citations
2. Missing timestamp
3. Citations required but not supported

Solutions:

Check provider response format:

```
curl http://127.0.0.1:8000/providers/my_endpoint
```

Verify manifest:

```
{
  "citations_supported": true  # Must be true if citations required
}
```

Fix provider to include required fields:

```
return {
    "answer": "...",
    "citations": ["https://source.com"],  # Required
    "retrieved_at": "2026-02-07T00:00:00+00:00"  # Required
}
```

9.1.6 High Latency

Symptom: Slow response times

Possible Causes:

1. Large language model
2. Semantic search overhead
3. External API latency
4. No caching

Solutions:

Use smaller model:

```
export SALES_AGENT_MODEL=phi3.5:3.8b
```

Enable caching:

```
export SHOP_CACHE_TTL_SECONDS=60
export PROVIDER_CACHE_TTL_WEATHER=30
```

Reduce candidates:

```
export SALES_AGENT_TOP_K=5
export SEMANTIC_SEARCH_TOP_K=15
```

Disable semantic search:

```
export SEMANTIC_SEARCH_ENABLED=false
```

9.1.7 Database Errors

Symptom: SQLite errors or missing tables

Possible Causes:

1. Corrupted database

2. Schema mismatch
3. Permissions issue

Solutions:

Delete and recreate database:

```
rm marketplace.db
uvicorn apps.api.main:app --reload
# Database auto-created on startup
```

Check file permissions:

```
ls -l marketplace.db
# Should be readable and writable
```

Use PostgreSQL for production:

```
export DATABASE_URL=postgresql://user:pass@host:5432/dbname
```

9.1.8 Test Failures

Symptom: `pytest` shows failing tests

Possible Causes:

1. Server not running (for integration tests)
2. Ollama not available
3. Network issues

Solutions:

Run unit tests only (no network):

```
pytest -m "not integration"
```

Start server before integration tests:

```
# Terminal 1
uvicorn apps.api.main:app --reload

# Terminal 2
pytest -v
```

Check Ollama availability:

```
curl http://localhost:11434/api/tags
```

9.2 Error Messages

9.2.1 "builtin type SwigPyPacked has no module attribute"

Type: Warning (can be ignored)

Cause: FAISS internal SWIG bindings

Solution: Already suppressed in `pytest.ini`

9.2.2 "Semantic search exceeded timeout"

Type: Warning

Cause: Semantic search taking longer than SEMANTIC_SEARCH_TIMEOUT_MS

Solution:

Increase timeout:

```
export SEMANTIC_SEARCH_TIMEOUT_MS=3000
```

Or optimize by reducing products indexed:

```
export SEMANTIC_SEARCH_TOP_K=15
```

9.2.3 "No apps met minimum capability coverage"

Type: Expected behavior

Cause: No products match required capabilities

Solution:

Check available capabilities:

```
python -m marketplace.cli apps | grep capabilities
```

Relax coverage requirement:

```
export MIN_CAP_COVERAGE=0.8
```

Use broader capabilities:

```
# Instead of very specific capabilities
--caps search  # Instead of --caps advanced_academic_search
```

9.2.4 "Citations required but missing in output"

Type: Validation error

Cause: Provider didn't return citations field

Solution:

Update provider code:

```
return {
    "answer": "...",
    "citations": ["https://source.com"],  # Add this
    "retrieved_at": datetime.now(timezone.utc).isoformat()
}
```

Or don't require citations:

```
python -m marketplace.cli shop "task" --no-citations
```

9.3 Performance Optimization

9.3.1 Reduce Latency

Goal: Minimize end-to-end query time

Actions:

1. Use smallest effective model
2. Enable aggressive caching
3. Reduce candidate set size
4. Disable semantic search if not needed

Configuration:

```
export ROUTER_MODEL=phi3.5:3.8b
export SALES_AGENT_MODEL=phi3.5:3.8b
export SALES_AGENT_TOP_K=5
export SHOP_CACHE_TTL_SECONDS=120
export SEMANTIC_SEARCH_ENABLED=false
```

9.3.2 Improve Accuracy

Goal: Better product recommendations

Actions:

1. Use larger models
2. Enable semantic search
3. Increase candidate set
4. Stricter thresholds

Configuration:

```
export ROUTER_MODEL=qwen2.5:14b-instruct
export SALES_AGENT_MODEL=qwen2.5:14b-instruct
export SEMANTIC_SEARCH_ENABLED=true
export SEMANTIC_SEARCH_TOP_K=30
export SALES_AGENT_TOP_K=10
export MIN_TOTAL_SCORE=0.60
```

9.3.3 Optimize Memory

Goal: Reduce RAM usage

Actions:

1. Disable semantic search
2. Reduce cache sizes
3. Use smaller models

Configuration:

```
export SEMANTIC_SEARCH_ENABLED=false
export SHOP_CACHE_TTL_SECONDS=30
export SALES_AGENT_MODEL=phi3.5:3.8b
```

9.4 Getting Help

9.4.1 Documentation Resources

- **README.md**: Quick start and overview
- **This guide**: Comprehensive documentation
- **API docs**: http://127.0.0.1:8000/docs
- **Test results**: Test Images/test-run.md

9.4.2 Debugging Steps

1. Check server logs
2. Verify configuration
3. Test individual components
4. Review execution receipts
5. Check trust scores

9.4.3 Reporting Issues

When reporting issues, include:

1. Axiomeer version (from git commit or release tag)
2. Python version (`python3 --version`)
3. Operating system
4. Full error message or traceback
5. Steps to reproduce
6. Configuration (environment variables)
7. Expected vs actual behavior

9.4.4 Support Channels

- GitHub Issues: https://github.com/ujjwalredd/axiomeer/issues
- Documentation: README.md and this guide
- Test suite: Run `pytest -v` for diagnostics

10 Best Practices

10.1 Production Deployment

10.1.1 Use Environment Variables

Never hardcode configuration. Use `.env` files:

```
# .env
DATABASE_URL=postgresql://user:pass@db:5432/axiomeer
API_BASE_URL=https://api.yourdomain.com
OLLAMA_URL=http://ollama:11434/api/generate
SEMANTIC_SEARCH_ENABLED=true
W_CAP=0.70
W_REL=0.25
MIN_CAP_COVERAGE=1.0
```

10.1.2 Run Multiple Workers

Use Uvicorn with multiple workers for concurrency:

```
uvicorn apps.api.main:app \
  --host 0.0.0.0 \
  --port 8000 \
  --workers 4 \
  --log-level info
```

10.1.3 Use PostgreSQL

Replace SQLite with PostgreSQL for production:

```
export DATABASE_URL=postgresql://user:pass@host:5432/axiomeer
```

10.1.4 Enable Monitoring

Track performance metrics:

```
import requests
import time

def monitor_system():
    while True:
        # Check health
        health = requests.get("http://api/health")

        # Check semantic search
        stats = requests.get("http://api/semantic-search/stats")

        # Check trust scores
        trust = requests.get("http://api/trust")

        # Log metrics
```

```
        log_metrics(health, stats, trust)

        time.sleep(60)
```

10.1.5 Implement Rate Limiting

Add rate limiting to prevent abuse:

```
from fastapi import FastAPI
from slowapi import Limiter
from slowapi.util import get_remote_address

limiter = Limiter(key_func=get_remote_address)
app = FastAPI()
app.state.limiter = limiter

@app.post("/shop")
@limiter.limit("10/minute")
async def shop_endpoint(...):
    # Endpoint logic
    pass
```

10.1.6 Set Up Backup

Regular database backups:

```
#!/bin/bash
# backup.sh

DATE=$(date +%Y%m%d_%H%M%S)
pg_dump -U user -h host axiomeer > backup_$DATE.sql
```

10.2 Security Best Practices

10.2.1 API Authentication

Add authentication for production:

```
from fastapi import Depends, HTTPException, Security
from fastapi.security import HTTPBearer

security = HTTPBearer()

async def verify_token(token: str = Security(security)):
    if not validate_token(token.credentials):
        raise HTTPException(status_code=403, detail="Invalid token")
    return token

@app.post("/shop")
async def shop(request: ShopRequest, token=Depends(verify_token)):
```

```
    # Endpoint logic
    pass
```

10.2.2 Input Validation

Always validate inputs:

```
from pydantic import BaseModel, validator

class ShopRequest(BaseModel):
    task: str

    @validator('task')
    def task_not_empty(cls, v):
        if not v or len(v.strip()) == 0:
            raise ValueError('Task cannot be empty')
        if len(v) > 1000:
            raise ValueError('Task too long (max 1000 chars)')
        return v.strip()
```

10.2.3 Sanitize Outputs

Prevent injection attacks:

```
import html

def sanitize_output(data: dict) -> dict:
    """Sanitize all string fields."""
    return {
        k: html.escape(v) if isinstance(v, str) else v
        for k, v in data.items()
    }
```

10.2.4 HTTPS Only

Use HTTPS in production:

```
# Nginx reverse proxy config
server {
    listen 443 ssl;
    server_name api.yourdomain.com;

    ssl_certificate /path/to/cert.pem;
    ssl_certificate_key /path/to/key.pem;

    location / {
        proxy_pass http://127.0.0.1:8000;
    }
}
```

10.3 Development Best Practices

10.3.1 Test Before Deploying

Run full test suite:

```
# Unit tests (69 tests)
pytest -v

# Benchmark tests (13 comprehensive tests)
# Terminal 1: Start server
uvicorn apps.api.main:app --reload

# Terminal 2: Run benchmarks
pytest tests/test_benchmarks.py -v
```

Benchmark Test Categories:

1. **Real API Validation** - Verify genuine data retrieval with citations
2. **Fake Query Rejection** - Demonstrate NO_MATCH for impossible requests (no hallucination)
3. **Latency Benchmarks** - Measure semantic search, endpoint, and pipeline performance
4. **Citation Validation** - Enforce citation requirements and quality checks
5. **Trust Scoring** - Verify score calculation and real-time updates
6. **Provenance Tracking** - Validate execution receipt logging

10.3.2 Use Version Control

Commit manifests and configuration:

```
git add manifests/categories/
git add .env.example
git commit -m "Add new weather provider"
```

10.3.3 Document Changes

Update documentation when adding features:

```
# Update README.md
# Update this guide
# Add examples to docs/
```

10.3.4 Code Review

Review manifest changes:

- Verify all required fields
- Check capability accuracy
- Test provider endpoints
- Validate response format

10.3.5 Monitor Trust Scores

Track provider reliability:

```
# Daily trust score check
python -m marketplace.cli trust > daily_trust_$(date +%Y%m%d).txt
```

10.4 Operational Best Practices

10.4.1 Regular Updates

Keep dependencies current:

```
pip install --upgrade pip
pip install --upgrade -e ".[dev]"
```

10.4.2 Log Analysis

Monitor logs for issues:

```
# Check for errors
grep ERROR logs/axiomeer.log

# Check latency
grep "total_time_ms" logs/axiomeer.log | awk '{print $NF}' | sort -n
```

10.4.3 Capacity Planning

Monitor resource usage:

```
# Memory usage
ps aux | grep uvicorn

# Disk usage
du -sh marketplace.db

# Request rate
grep "POST /shop" logs/access.log | wc -l
```

10.4.4 Incident Response

When issues occur:

1. Check server logs
2. Verify external API status
3. Review recent configuration changes
4. Check trust scores for failing providers
5. Review execution receipts
6. Rollback if needed

11 Appendix

11.1 Glossary

App / Product: A tool, API, dataset, or model registered in the marketplace

Capability: A functional tag describing what a product can do (e.g., "weather", "finance", "citations")

Citation: A reference to the source of data (typically a URL)

Executor: The component that calls provider endpoints

Freshness: How frequently data is updated (static, daily, realtime)

Manifest: JSON file describing a product's metadata and capabilities

Provider: An organization or service offering a product through the marketplace

Provenance: The origin and history of data, including sources and timestamps

Receipt: A logged record of an execution with full details

Router: The component that ranks and recommends products

Semantic Search: Vector-based product discovery using embeddings

TF-IDF: Text Frequency-Inverse Document Frequency, a keyword-based relevance algorithm

Trust Score: A reliability metric based on success rate and citation compliance

Validation: The process of verifying output format and citations

11.2 Environment Variable Reference

Complete list of all configuration variables:

```
# Database
DATABASE_URL=sqlite:///./marketplace.db
API_BASE_URL=http://127.0.0.1:8000

# LLM
OLLAMA_URL=http://localhost:11434/api/generate
OLLAMA_TIMEOUT=60
ROUTER_MODEL=phi3.5:3.8b
ANSWER_MODEL=phi3.5:3.8b
SALES_AGENT_MODEL=phi3.5:3.8b
SALES_AGENT_TIMEOUT=60

# Router Weights
W_CAP=0.70
W_LAT=0.20
W_COST=0.10
W_TRUST=0.15
W_REL=0.25
```

```
# Routing Thresholds
MIN_CAP_COVERAGE=1.0
MIN_RELEVANCE_SCORE=0.08
MIN_TOTAL_SCORE=0.55

# Sales Agent
SALES_AGENT_TOP_K=8

# Caching
SHOP_CACHE_TTL_SECONDS=30
PROVIDER_CACHE_TTL_WEATHER=30
PROVIDER_CACHE_TTL_FX=60
PROVIDER_CACHE_TTL_WIKI=3600
PROVIDER_CACHE_TTL_WIKIDATA=3600
PROVIDER_CACHE_TTL_WIKIDUMPS=3600
PROVIDER_CACHE_TTL_RESTCOUNTRIES=3600
PROVIDER_CACHE_TTL_OPENLIB=3600
PROVIDER_CACHE_TTL_DICTIONARY=86400

# Semantic Search
SEMANTIC_SEARCH_ENABLED=true
SEMANTIC_SEARCH_TOP_K=20
SEMANTIC_SEARCH_TIMEOUT_MS=1000
SEMANTIC_SEARCH_MODEL=all-MiniLM-L6-v2
```

11.3 API Response Schemas

11.3.1 ShopResponse

```
{
  "status": "OK",
  "recommendations": [
    {
      "app_id": "string",
      "name": "string",
      "score": 0.75,
      "capability_match": 1.0,
      "trust_score": 0.85,
      "relevance_score": 0.45
    }
  ],
  "explanation": ["string"],
  "sales_agent": {
    "summary": "string",
    "final_choice": "string",
    "recommendations": [
      {
        "app_id": "string",
```

```
        "name": "string",
        "rationale": "string",
        "tradeoff": "string"
      }
    ]
  },
  "metrics": {
    "total_time_ms": 245,
    "semantic_search_time_ms": 120,
    "tfidf_time_ms": 45,
    "scoring_time_ms": 80,
    "semantic_search_used": true,
    "semantic_search_available": true
  }
}
```


11.3.2 ExecuteResponse


```
{
  "ok": true,
  "run_id": 42,
  "output": {
    "answer": "string",
    "citations": ["string"],
    "retrieved_at": "2026-02-07T00:00:00+00:00",
    "quality": "verified"
  },
  "provenance": {
    "sources": ["string"],
    "retrieved_at": "2026-02-07T00:00:00+00:00",
    "notes": []
  },
  "latency_ms": 910,
  "validation_errors": []
}
```


11.3.3 TrustScoreResponse


```
{
  "app_id": "string",
  "trust_score": 0.85,
  "total_executions": 100,
  "successful_executions": 95,
  "success_rate": 0.95,
  "avg_latency_ms": 850,
  "p50_latency_ms": 820,
  "p95_latency_ms": 950,
  "p99_latency_ms": 980,
```

```
  "citation_pass_rate": 1.0,
  "last_execution": "2026-02-07T00:00:00+00:00"
}
```

11.4 Capability Tags Reference

Common capability tags used in the marketplace:

Data Categories: - `weather` - Weather and climate data - `finance` - Financial and market data - `government_data` - Official government statistics - `knowledge` - Academic and research data - `media` - Images, videos, audio - `geographic` - Location and mapping data

Data Operations: - `search` - Search capabilities - `rag` - Retrieval-augmented generation - `summarize` - Text summarization - `translate` - Language translation - `sentiment` - Sentiment analysis - `ner` - Named entity recognition

Data Quality: - `citations` - Provides source citations - `realtime` - Real-time data - `daily` - Daily updated data - `static` - Static datasets

Technical: - `api` - REST API endpoint - `model` - AI model inference - `dataset` - Structured dataset - `tool` - Utility or tool - `aggregator` - Multi-source aggregation

11.5 Product Catalog

Complete list of 60 products by category:

11.5.1 Government Data (11)

1. NASA Astronomy Picture of the Day
2. NASA Mars Rover Photos
3. NASA Near Earth Objects
4. US Census Demographics
5. World Bank Global Indicators
6. FRED Economic Data
7. IMF Financial Statistics
8. EU Open Data
9. UK Police Crime Data
10. COVID-19 Global Statistics
11. Data.gov Datasets

11.5.2 Knowledge & Research (8)

1. arXiv Scientific Papers
2. PubMed Biomedical Literature
3. Semantic Scholar
4. Crossref Scholarly Metadata
5. Wikidata Knowledge Graph
6. DBpedia Structured Data
7. Internet Archive
8. Project Gutenberg

11.5.3 Financial & Crypto (3)

1. CoinGecko Cryptocurrency Data
2. Coinbase Exchange Rates
3. Blockchain.com Bitcoin Data

11.5.4 Geographic & Location (3)

1. OpenStreetMap Geocoding
2. GeoNames Geographic Database
3. IP Geolocation API

11.5.5 Media & Entertainment (6)

1. OMDB Movie Database
2. TMDB Movie & TV Database
3. MusicBrainz Music Metadata
4. Genius Song Lyrics
5. Unsplash Free Photos
6. Pexels Photos & Videos

11.5.6 Language & Translation (2)

1. LibreTranslate
2. Datamuse Word API

11.5.7 Utilities & Tools (7)

1. Random User Generator
2. Random Data Generator
3. QR Code Generator
4. Lorem Picsum
5. JokeAPI
6. Open Trivia Database
7. Bored API

11.5.8 AI Models (12)

1. GPT-2 Text Generation (HF)
2. Sentiment Analysis (HF)
3. English to Spanish Translation (HF)
4. English to French Translation (HF)
5. Text Summarization (HF)
6. Question Answering (HF)
7. Named Entity Recognition (HF)
8. Image Classification (HF)
9. Llama 3.1 8B (Ollama)
10. Mistral 7B (Ollama)
11. CodeLlama 13B (Ollama)
12. Deepseek Coder 6.7B (Ollama)

11.5.9 General (8)

1. Real-time Weather Agent
2. Wikipedia Search
3. Wikipedia Dumps
4. Wikidata Search
5. REST Countries
6. Open Library
7. Dictionary API
8. Exchange Rates API

11.6 Version History

11.6.1 v6.0 (February 2026)

Major Features: - Semantic search with vector embeddings (FAISS) - Hybrid relevance scoring (70% semantic + 30% TF-IDF) - Performance metrics in all responses - Expanded to 60 products - 52 provider endpoints - Graceful fallback to TF-IDF - Production monitoring endpoints

Improvements: - 100% provider endpoint success (previously 92.3%) - Benchmark success rate: 90.48% - Hot-reload capability for product index - Configurable semantic search model - Thread-safe operations

11.6.2 v5.0 (January 2026)

Features: - 60 products across 8 categories - Enhanced categorization (category, subcategory, tags) - Provider endpoint caching - Trust score tracking - Manifest schema validation

11.6.3 Earlier Versions

- v4.0: Initial 8-provider integration
- v3.0: Sales agent recommendations
- v2.0: Capability extraction with LLM
- v1.0: Basic routing and execution

11.7 License

Axiomeer Product Guide v6.0 February 2026 © Axiomeer

For the latest documentation, visit: https://github.com/ujjwalredd/axiomeer

12 Benchmark Testing

12.1 Overview

Comprehensive benchmark tests validate system performance, demonstrate no-hallucination behavior, and measure latency across all components.

Test File: `tests/test_benchmarks.py` (13 tests)

12.2 Test Categories

12.2.1 1. Real API Validation

Verifies that real queries return genuine data with proper citations.

Tests:

- `test_weather_api_real_data`: Weather API returns real data with citations
- `test_finance_api_real_data`: Financial API returns verified cryptocurrency data
- `test_research_api_real_data`: Research API returns academic papers

Validation:

- Response status: 200 OK
- Citations present: Non-empty array
- Retrieved timestamp: ISO 8601 format
- Quality field: "verified"

Example Output:

```
Weather query: 245ms
Semantic search: 120ms
Found 1 products
Citations: 1
Quality: verified
```

12.2.2 2. Fake Query Rejection

Demonstrates that impossible or non-existent queries are properly rejected without hallucination.

Tests:

- `test_fake_capability_returns_no_match`: Code execution capability (doesn't exist) → NO_MATCH
- `test_impossible_requirement_returns_no_match`: 1ms latency requirement (impossible) → NO_MATCH
- `test_nonexistent_product_returns_no_match`: Quantum computing (doesn't exist) → NO_MATCH

Validation:

- Status: NO_MATCH
- Recommendations: Empty array
- Explanation: Clear reason for rejection

Example Output:

```
Fake query rejected: 185ms
 Status: NO_MATCH
 Explanation: ["No apps met minimum capability coverage (1.00)"]
```

This proves: System does not hallucinate or fabricate responses for non-existent capabilities.

12.2.3 3. Latency Benchmarks

Measures performance across semantic search, API endpoints, and end-to-end pipeline.

Tests:

- `test_semantic_search_performance`: Multiple queries to measure semantic search speed
- `test_api_endpoint_latency`: Individual endpoint response times
- `test_end_to_end_execution_latency`: Complete shop → execute pipeline

Target Performance:

Operation	Target	Typical
Semantic Search	<50ms	15-25ms
Health Check	<100ms	<10ms
Shop Query	Variable	100-300ms
Provider Execute	Variable	500-2000ms

Example Output:

```
Semantic Search Benchmark (5 queries):
 Average total latency: 245ms
 Average semantic search: 22ms
 Min: 180ms, Max: 310ms

API Endpoint Latencies:
 /health: 3ms
 /apps: 5ms
 /trust: 8ms
 /semantic-search/stats: 2ms

End-to-End Pipeline:
 Shop: 235ms
 Execute: 810ms
 Total: 1045ms
 Provider latency: 780ms
```

12.2.4 4. Citation Validation

Enforces citation requirements and validates output quality.

Tests:

- `test_citations_required_enforcement`: Verify citations are enforced when required
- `test_provenance_tracking`: Validate execution receipt logging

Validation:

- Citations field: Present and non-empty
- Retrieved_at field: Valid ISO timestamp
- Run ID: Generated and stored
- Receipt: Retrievable via `/runs/{run_id}`

Example Output:

```
Citations validated: 2 sources
Provenance tracked: run_id=42
 App: exchange_rates
 Success: True
```

12.2.5 5. Trust Scoring

Verifies trust score calculation and real-time updates.

Tests:

- `test_trust_scores_available`: Trust scores calculated for executed products
- `test_trust_score_updates_after_execution`: Scores update in real-time

Validation:

- Trust score: 0.0-1.0 range
- Success rate: Calculated from executions
- Total executions: Increments after each run
- Latency metrics: P50, P95, P99 percentiles

Example Output:

```
Trust scores available: 15 products
 Sample: exchange_rates = 0.93
 Success rate: 96.0%

Trust score updated for exchange_rates
 Executions: 24 → 25
```

12.2.6 6. Provenance Tracking

Validates that all executions are logged with full provenance data.

Tests:

- Execution creates run_id
- Receipt contains: app_id, task, output, timestamps
- Receipt retrievable via API

Validation:

- Run ID present in execute response

- Full receipt accessible
- Provenance data immutable

12.3 Running Benchmarks

12.3.1 Prerequisites

```
# Start server
uvicorn apps.api.main:app --reload
```

12.3.2 Run All Benchmarks

```
pytest tests/test_benchmarks.py -v
```

12.3.3 Run Specific Category

```
# Real API validation only
pytest tests/test_benchmarks.py::TestRealAPIValidation -v

# Fake query rejection only
pytest tests/test_benchmarks.py::TestFakeQueryRejection -v

# Latency benchmarks only
pytest tests/test_benchmarks.py::TestLatencyBenchmarks -v
```

12.3.4 Expected Results

All tests passing:

```
tests/test_benchmarks.py::TestRealAPIValidation::test_weather_api_real_data PASSED
tests/test_benchmarks.py::TestRealAPIValidation::test_finance_api_real_data PASSED
tests/test_benchmarks.py::TestRealAPIValidation::test_research_api_real_data PASSED
tests/test_benchmarks.py::TestFakeQueryRejection::test_fake_capability_returns_no_match PASSED
tests/test_benchmarks.py::TestFakeQueryRejection::test_impossible_requirement_returns_no_match
tests/test_benchmarks.py::TestFakeQueryRejection::test_nonexistent_product_returns_no_match PA
tests/test_benchmarks.py::TestLatencyBenchmarks::test_semantic_search_performance PASSED
tests/test_benchmarks.py::TestLatencyBenchmarks::test_api_endpoint_latency PASSED
tests/test_benchmarks.py::TestLatencyBenchmarks::test_end_to_end_execution_latency PASSED
tests/test_benchmarks.py::TestCitationValidation::test_citations_required_enforcement PASSED
tests/test_benchmarks.py::TestCitationValidation::test_provenance_tracking PASSED
tests/test_benchmarks.py::TestTrustScoring::test_trust_scores_available PASSED
tests/test_benchmarks.py::TestTrustScoring::test_trust_score_updates_after_execution PASSED

============================== 13 passed in 15.32s ==============================
```

12.4 Performance Summary

From benchmark results:

- **Semantic search:** 15-25ms average (well under 50ms target)
- **Health endpoint:** <10ms (meets target)

- **Shop queries:** 180-310ms (within expected range)
- **Provider execution:** 500-2000ms (varies by external API)
- **Fake query rejection:** <200ms (fast rejection)
- **Citation enforcement:** 100% (no failures)

Reliability:

- Unit tests: 69/69 passing (100%)
- Benchmark tests: 13/13 passing (100%)
- No hallucination: Proven via fake query tests
- Citation compliance: 100% enforcement

12.5 Troubleshooting Benchmarks

12.5.1 Tests fail: Connection refused

Cause: API server not running

Solution:

```
# Terminal 1
uvicorn apps.api.main:app --reload

# Terminal 2 (wait for server startup)
pytest tests/test_benchmarks.py -v
```

12.5.2 Semantic search tests skipped

Cause: Semantic search not initialized

Solution:

```
pip install faiss-cpu sentence-transformers
# Restart server
```

12.5.3 Slow performance

Expected: First run may be slower (model loading, cache warming)

Typical: Subsequent runs faster due to caching

12.5.4 Provider tests fail

Cause: External API rate limiting or downtime

Expected: These tests call real external APIs and may occasionally fail

Solution: Retry after a few minutes