Waft System Overview

Version: 0.3.1-alpha

Date: 2026-01-10

Status: Active Development

Table of Contents

What is Waft?

Waft is a Python meta-framework for directed evolution of self-modifying AI agents.

Think of it as an "operating system" for AI agent research projects. Waft provides:

- Project scaffolding (uv-based Python projects)
- Memory management (_pyrite directory structure)
- Epistemic tracking (knowing what you know/don't know)

The Core Promise

> "Don't just build agents. Breed

them."

Waft enables you to:

1. Create self-modifying agents that

can evolve their own code

2. Test agents in simulated

environments (Scint Gym)

3. Track complete evolutionary

lineages

4. Generate scientific data suitable

for research publication

Core Concepts

1. The Three Pillars

Pillar 1: The Substrate

Agents write their own Python source

code ("code as DNA").

- Genome ID: SHA-256 hash of

agent's code + configuration

- Mutations: Agents can spawn

variants with code changes

- Hot-swapping: Adopt better

genomes mid-execution

- Reproduction: Create child agents

with specific genetic modifications

Pillar 2: The Physics (Scint System)

Reality Fracture Detection acts as

natural selection.

Four types of errors agents must

handle:

- SYNTAXTEAR: Formatting errors

(JSON, XML, code)

- LOGICFRACTURE: Math errors,

contradictions, schema violations

- SAFETY_VOID: Harmful content,

PII leaks

- HALLUCINATION: Fabricated facts,

wrong citations

Fitness Equation:

```
Fitness = (Stability  0.4) + (Efficiency  0.3) + (Safety  0.3)
```

```
If Fitness < 0.5 -> DEATH (evolutionary dead end)
```

Pillar 3: The Flight Recorder

Every evolutionary action is logged to

JSONL for scientific analysis.

Event Types Logged:

- SPAWN - Agent creates variant

- MUTATE - Agent modifies genome

- GYM_EVAL - Fitness evaluation

- SURVIVAL - Passed fitness

threshold

- DEATH - Failed fitness test

This enables:

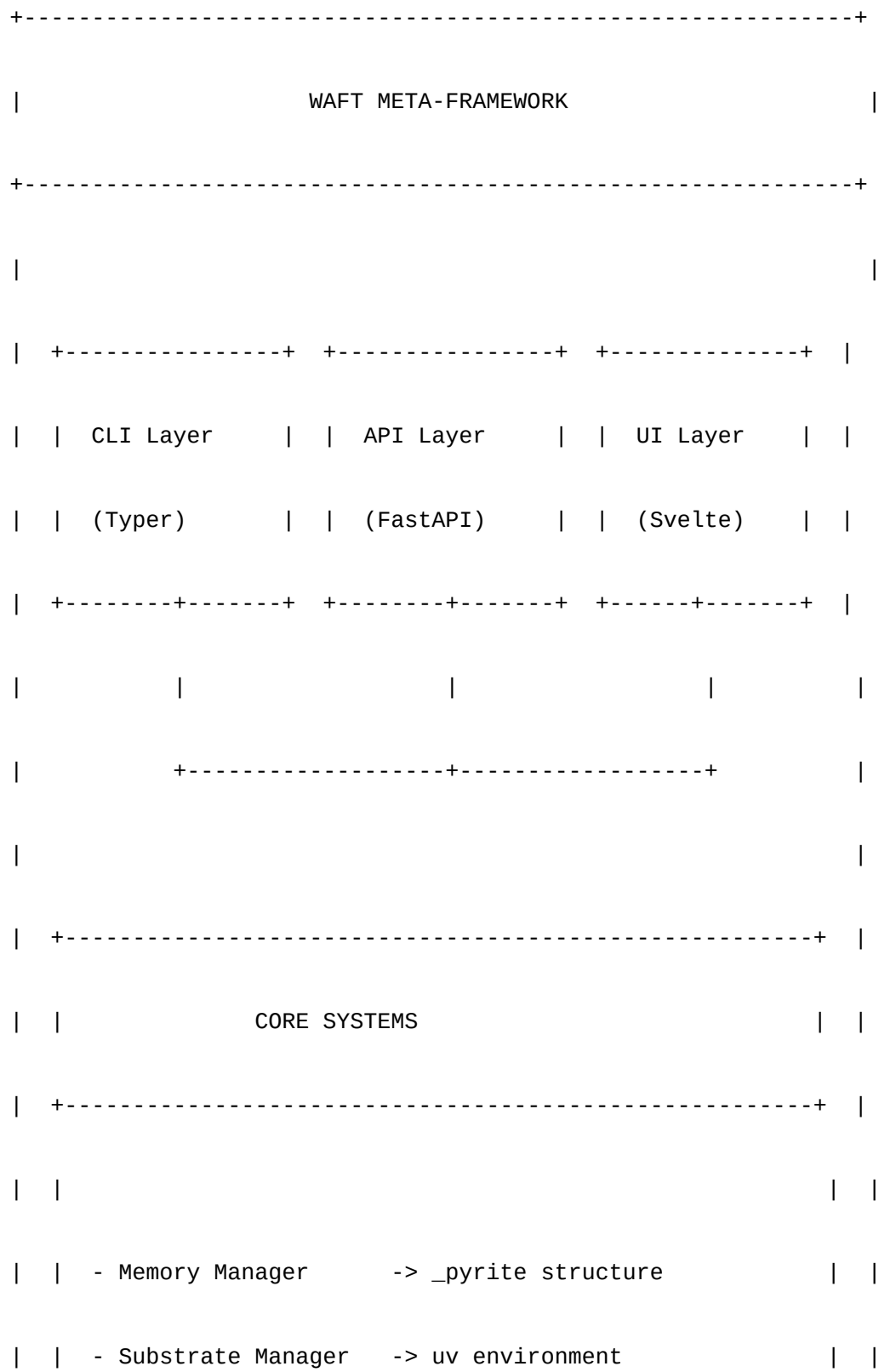
- Phylogenetic tree reconstruction

- Mutation impact measurement

- Convergence analysis

- Scientific publication

System Architecture


WAFT META-FRAMEWORK
CLI Layer
(Typer)
API Layer
(FastAPI)
UI Layer
(Svelte)
CORE SYSTEMS
- Memory Manager -> _pyrite structure
- Substrate Manager -> uv environment

How It Works

Project Lifecycle

1. Project Creation (waft new my_project)

```
$ waft new my_laboratory
```

Creates:

- uv-based Python project

(pyproject.toml)

- _pyrite/ memory structure

- .github/workflows/ CI/CD

- Justfile for task automation

- src/agents.py template

2. Development Workflow

```
# Verify project structure

$ waft verify



# Add dependencies

$ waft add numpy



# Sync environment

$ waft sync



# Run tests

$ just test  # or waft test (if configured)
```

3. Epistemic Tracking

```
# Create session

$ waft session create



# Log discoveries

$ waft finding log "Discovered X has property Y" --impact 0.8



# Log knowledge gaps

$ waft unknown log "Need to investigate Z"



# Check epistemic state

$ waft assess
```

4. Gamification

Every command rolls dice:

```
# Command: waft new

Ability: CHA (Charisma)

Roll: d20 + CHA modifier

DC: 10



Result:

  20 -> Critical Success -> Bonus XP + Credits

  15-19 -> Superior -> Extra XP

  10-14 -> Normal Success -> Standard XP

  5-9 -> Mixed Result -> Reduced XP

  2-4 -> Failure -> No XP

  1 -> Critical Failure -> Lose Integrity
```

5. Agent Evolution (Experimental)

```
# Spawn variants

$ waft spawn --agent RefactorAgent --mutation improved_prompt.jso



# Evaluate fitness

$ waft eval --agent RefactorAgent



# Evolve to best variant

$ waft evolve --agent RefactorAgent --generation 5
```

Key Components

Core Systems

1. Memory Manager (src/waft/core/memory.py)

Manages the _pyrite/ directory

structure:

```
_pyrite/
+-- active/          # Current work
+-- backlog/         # Future work
+-- standards/       # Project standards
+-- gym_logs/        # Fitness test results
+-- science/         # Scientific observations
    +-- laboratory.jsonl  # Event log
```

2. Substrate Manager (src/waft/core/substrate.py)

Manages uv environment:

- Project initialization

- Dependency management

- Lock file verification

- Virtual environment setup

3. Empirica Manager (src/waft/core/empirica.py)

Epistemic state tracking:

- Session management

- Finding/Unknown logging

- Safety gates

(PROCEED/HALT/BRANCH/REVISE)

- Moon phase calculation (epistemic

confidence)

4. Gamification Manager (src/waft/core/gamification.py)

D&D-style progression:

- Character stats (STR, DEX, CON,

INT, WIS, CHA)

- XP and leveling

- Integrity/Insight tracking

- Achievement system

5. TavernKeeper (src/waft/core/tavern_keeper/keeper.py)

RPG game master:

- Dice rolling (d20 system)

- Narrative generation (Tracery

grammars)

- Reward calculation

- Chronicle journaling

6. Decision Engine (src/waft/core/decision_matrix.py)

Weighted Sum Model (WSM) for

decisions:

- Criteria weighting

- Option scoring

- Mathematical decision analysis

- Visualization

7. TheObserver (src/waft/core/science/observer.py)

Scientific logging singleton:

- Immutable JSONL logging

- Complete event context

- Phylogenetic data

- Research-grade output

Agent System

BaseAgent (src/waft/core/agent/base.py)

Self-modifying agent with OODA

cycle:

OODA Loop:

1. Observe - Gather environment

data

2. Decide - Make decisions based on

observations

3. Act - Execute decisions

4. Reflect - Learn from outcomes

Capabilities:

- Genome management (DNA-like

code tracking)

- Inventory system (items/tools)

- Reproduction (spawn variants)

- Archetypes (explorer, builder,

analyzer, etc.)

Fitness Testing

Scint Gym (src/gym/rpg/scint.py)

Reality Fracture Detection System:

Quest Structure:

1. Generate scenario with intentional

errors

2. Agent attempts to stabilize (fix

errors)

3. Measure success across 4

dimensions

4. Calculate fitness score

5. Classify: SURVIVAL or DEATH

Error Types:

```
class ScintType(Enum):

    SYNTAX_TEAR = "syntax_tear"        # Format errors

    LOGIC_FRACTURE = "logic_fracture"  # Logic errors

    SAFETY_VOID = "safety_void"        # Safety violations

    HALLUCINATION = "hallucination"    # Factual errors
```

Usage Guide

Basic Commands

```
# Project Management

waft new <name>              # Create new project

waft init                    # Initialize in existing project

waft verify                  # Verify project structure

waft info                    # Show project info

waft sync                    # Sync dependencies

waft add <package>           # Add dependency


# Epistemic Tracking

waft session create          # Start session

waft session bootstrap       # Load context + dashboard

waft finding log <text>      # Log discovery

waft unknown log <text>      # Log knowledge gap

waft check                   # Run safety gate
```

Configuration

pyproject.toml

```
[project]

name = "my-project"

version = "0.1.0"

requires-python = ">=3.10"



dependencies = [

    "waft>=0.3.0",

    # ... your dependencies

]
```

_pyrite/standards/

Store project standards:

- codestyle.md

- architecture.md

- testingstandards.md

Technical Details

Technology Stack

Backend:

- Python 3.10+
- Typer (CLI)
- FastAPI (API)
- Pydantic (validation)
- Rich (terminal UI)
- uv (package management)

Frontend:

- SvelteKit (web dashboard)

- Tailwind CSS

- TypeScript

Data:

- TinyDB (lightweight JSON DB)

- JSONL (event logging)

- SQLite (analytics)

File System Layout

```
project/
+-- pyproject.toml              # uv project config
+-- uv.lock                     # Dependency lock
+-- Justfile                    # Task automation
+-- .github/workflows/          # CI/CD
+-- src/
|   +-- your_code.py
+-- tests/
|   +-- test_your_code.py
+-- _pyrite/                    # Waft memory
    +-- active/
    +-- backlog/
    +-- standards/
    +-- gym_logs/
```

Data Models

Genome ID

```
genome_id = sha256(
    agent_code +
    agent_config +
    agent_prompts
).hexdigest()
```

Event Log Entry


```
{
  "timestamp": "2026-01-10T15:00:00.000Z",
  "event_type": "SPAWN",
  "genome_id": "a4c426d...",
  "parent_id": "dd11732...",
  "generation": 5,
  "payload": {
    "mutations": ["improved_prompt"],
    "git_diff": "...",
    "scientific_name": "Evolutius Maximus"
  },
  "fitness": {
    "stability": 0.85,
    "efficiency": 0.72,
```

API Endpoints

The FastAPI server provides:

```
GET  /health                    # Health check
GET  /status                    # System status

POST /decision/analyze          # WSM analysis
GET  /decision/criteria         # List criteria

POST /empirica/session          # Create session
GET  /empirica/status          # Session status
POST /empirica/finding          # Log finding
POST /empirica/unknown          # Log unknown

GET  /gym/quests                # List quests
```

Directory Structure

```
waft/
+-- src/waft/
|   +-- main.py                # CLI entry (2020 lines)
|   +-- logging.py             # Centralized logging
|   +-- config/                # Configuration
|   |   +-- theme.py           # Visual constants
|   |   +-- abilities.py       # Command abilities
|   +-- cli/                   # CLI components
|   |   +-- epistemic_display.py
|   |   +-- hud.py
|   +-- api/                   # FastAPI server
|   |   +-- main.py
|   |   +-- models.py
```

Performance Characteristics

- CLI startup: ~200-500ms

- Project creation: ~2-5 seconds

- Dependency sync: Variable

(network-dependent)

- Dashboard render: ~100-300ms

- Fitness evaluation: ~1-10 seconds

per quest

- Event logging: ~1-5ms per event

(JSONL append)

Security Considerations

1. No credential storage - Waft

doesn't store passwords/tokens

2. Local-first - All data stored locally

3. No telemetry - No data sent to

external servers

4. Safe defaults - Sandboxed

execution when possible

5. Audit logging - Complete event

history for accountability

Limitations & Known Issues

1. Platform Support:

- [x] Linux

- [x] macOS

- [!] Windows (partial support, some

terminal features may not work)

2. Scale Limits:

- Not tested beyond ~1000 agents

- JSONL log can grow large (>100MB

after extensive use)

- Dashboard performance degrades

with >500 events

3. Dependencies:

- Requires uv (external tool)

- Git required for version tracking

- Python 3.10+ required

4. Incomplete Features:

- Evolution cycle not fully automated

- Some Scint types under-tested

- Karma/reincarnation system

incomplete

Future Roadmap

Phase 2 (Architectural Improvements)

- Split main.py into command

modules

- Refactor visualizer.py god object

- Decompose agent/base.py

- Create Manager interface

Phase 3 (Feature Completion)

- Complete evolution automation

- Enhanced Scint Gym testing

- Multi-agent coordination

- Distributed evaluation

Phase 4 (Production Hardening)

- Comprehensive test coverage

- Performance optimization

- Documentation completion

- Stability improvements

Getting Help

- Documentation: /docs directory

- Issues:

https://github.com/ctavolazzi/waft/issues

- Commands: waft --help or waft

<command> --help

- Examples: /examples directory

Contributing

See CONTRIBUTING.md for:

- Development setup

- Code standards

- Pull request process

- Testing requirements

License

MIT License - See LICENSE file

Last Updated: 2026-01-10

Maintainer: Waft Team

Repository:

https://github.com/ctavolazzi/waft