F_D Functor Framework — Design & Implementation Guide

Version: 1.0.0 **Implements**: REQ-ITER-003 (Functor Encoding Tracking), REQ-EVAL-002 (Evaluator Composition) **Package**: `imp_claude/code/genisis/`

---

1. Conceptual Overview

The genisis methodology defines **8 functional units** (evaluate, construct, classify, route, propose, sense, emit, decide) and **3 category renderings** for each:

Category	Symbol	Meaning	Example
F_D	Deterministic	Subprocess, regex, dict lookup	Run `pytest`, parse REQ tags
F_P	Probabilistic	LLM / agent call	"Does this code meet the criterion?"
F_H	Human	Interactive prompt	"Do you approve this design?"

Profile encoding selects which category renders each unit. A standard project uses `evaluate→F_D` (run tests), while a spike uses `evaluate→F_P` (agent judges experiment quality). Two units are **category-fixed**: `emit` is always `F_D`, `decide` is always `F_H`.

The functor framework provides the **executable code** behind these abstractions.

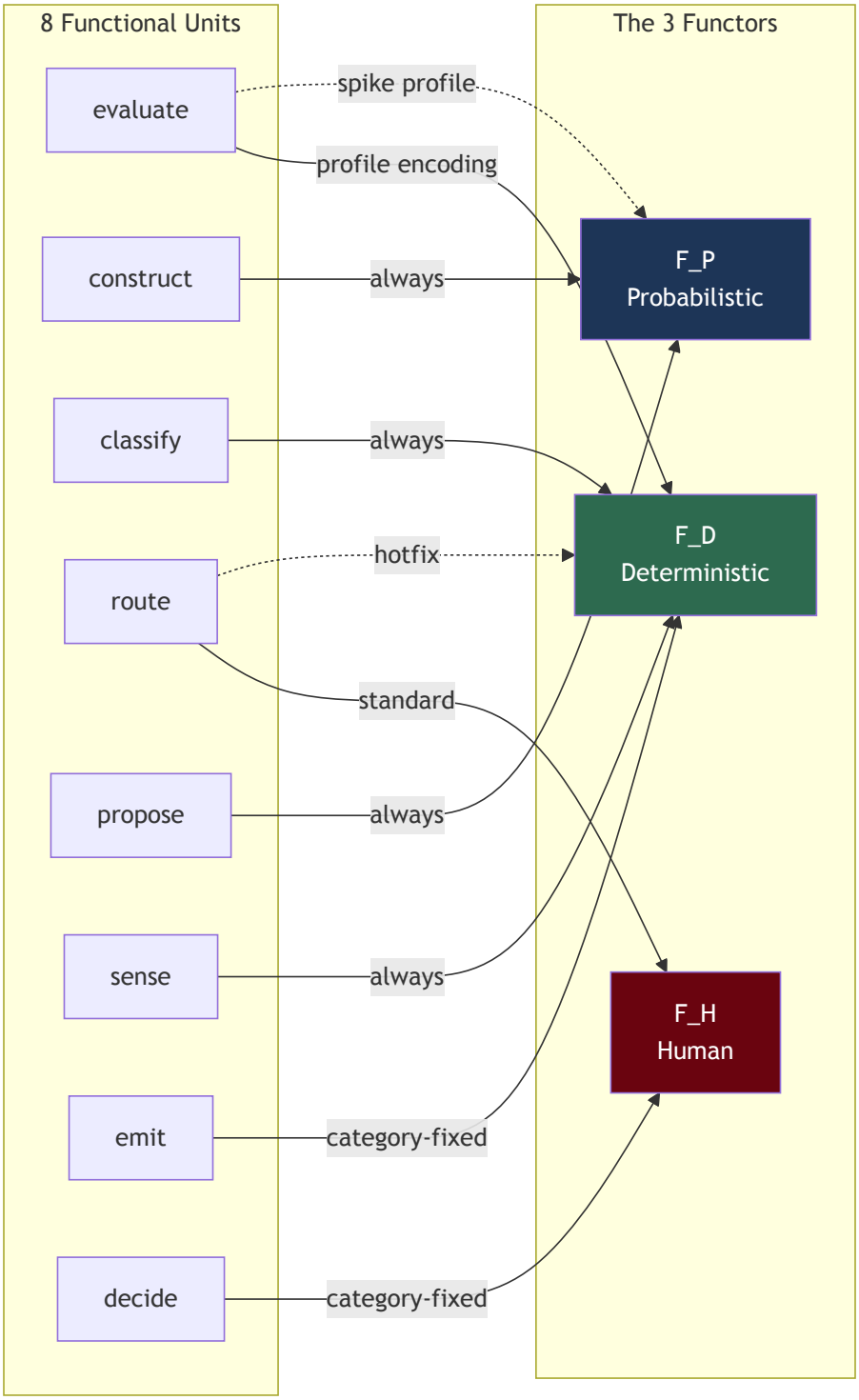
8 Functional Units
The 3 Functors
evaluate
construct
classify
route
propose
sense
emit
decide
spike profile
profile encoding
always
always
hotfix
standard
always
always
category-fixed
category-fixed
F_P
Probabilistic
F_D
Deterministic
F_H
Human

Diagram 0

2. Package Structure

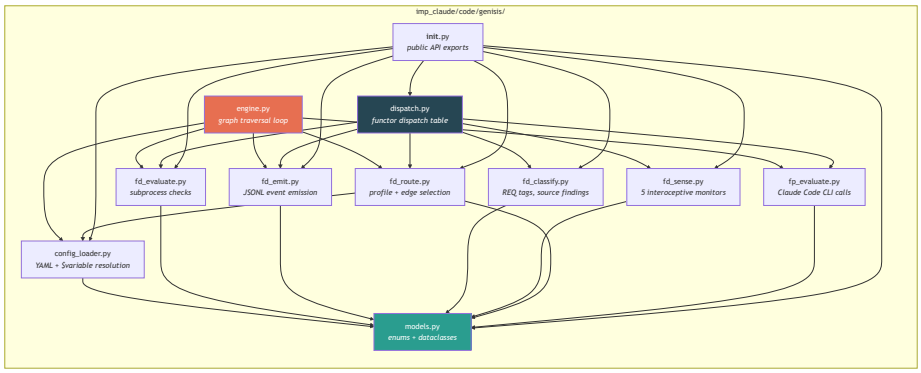

Diagram 1

---

3. Data Model (Class Diagram)

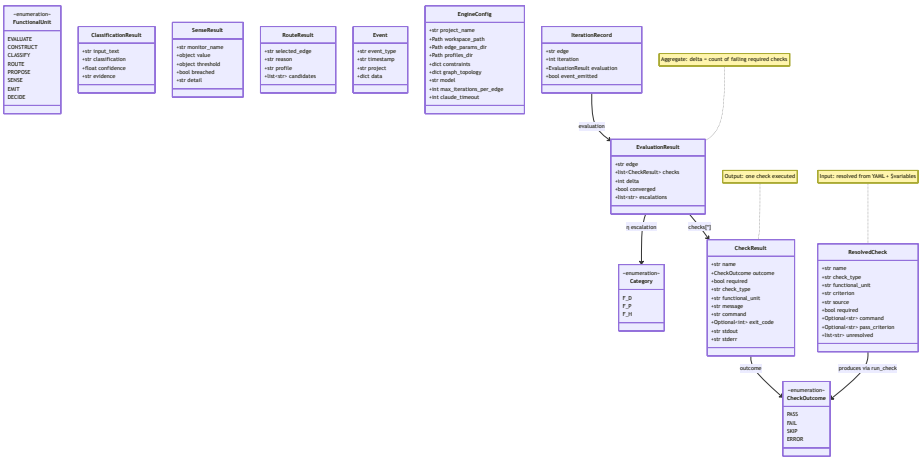

Diagram 2

CATEGORY_FIXED Invariant

```
CATEGORY_FIXED = {
    FunctionalUnit.EMIT:   Category.F_D,   # emit is ALWAYS
        deterministic
    FunctionalUnit.DECIDE: Category.F_H,   # decide is ALWAYS human
}
```

This is enforced across all 6 profiles and validated in tests.

---

4. Configuration Resolution Pipeline

The configuration system composes constraints from four layers:

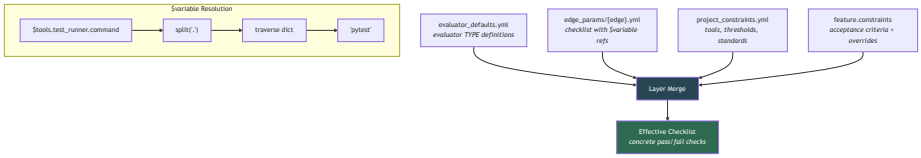

Diagram 3

$Variable Resolution Regex

```
Pattern: \$(\w+(?:\.\w+)*)

Matches:
  $tools.test_runner.command      → constraints["tools"]
["test_runner"]["command"]
  $thresholds.test_coverage_minimum → constraints["thresholds"]
["test_coverage_minimum"]
  $standards.style_guide          → constraints["standards"]
["style_guide"]
```

Resolution Rules

1. Edge checklist defines default checks
2. `$variables` resolve from `project_constraints.yml`
3. Feature `threshold_overrides` apply on top
4. Feature `acceptance_criteria` append to checklist
5. `required=true` at any layer stays `true` (most restrictive wins)
6. Unresolved `$variables` → check **SKIPPED** with warning (tracked in `unresolved[]`)

Sequence: `resolve_checklist()`

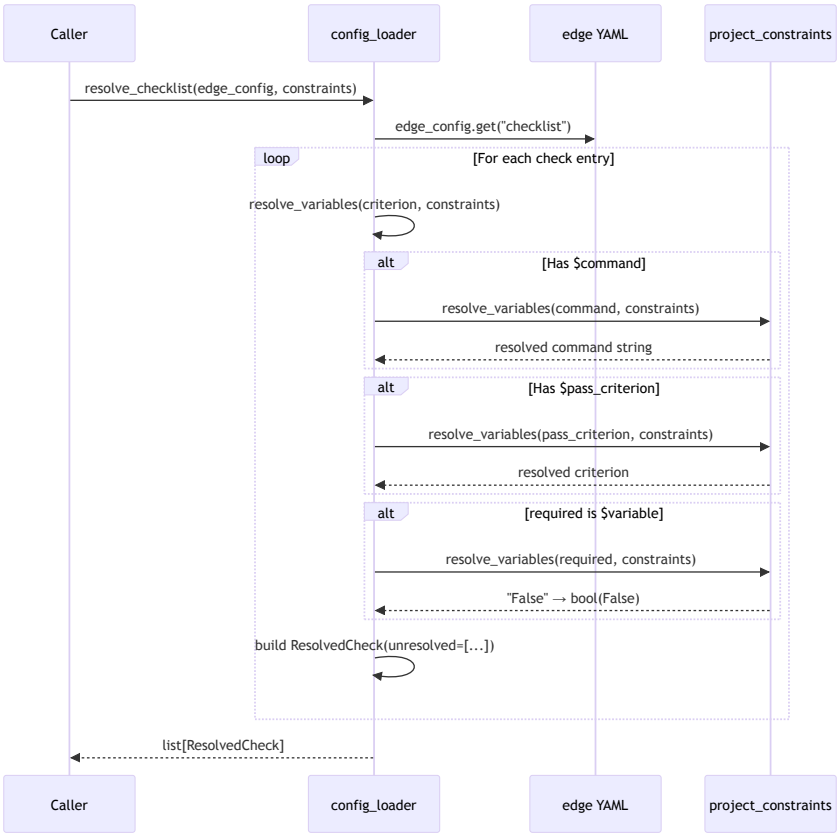

Diagram 4

---

5. F_D Evaluate — The Deterministic Evaluator

State Machine: Check Execution

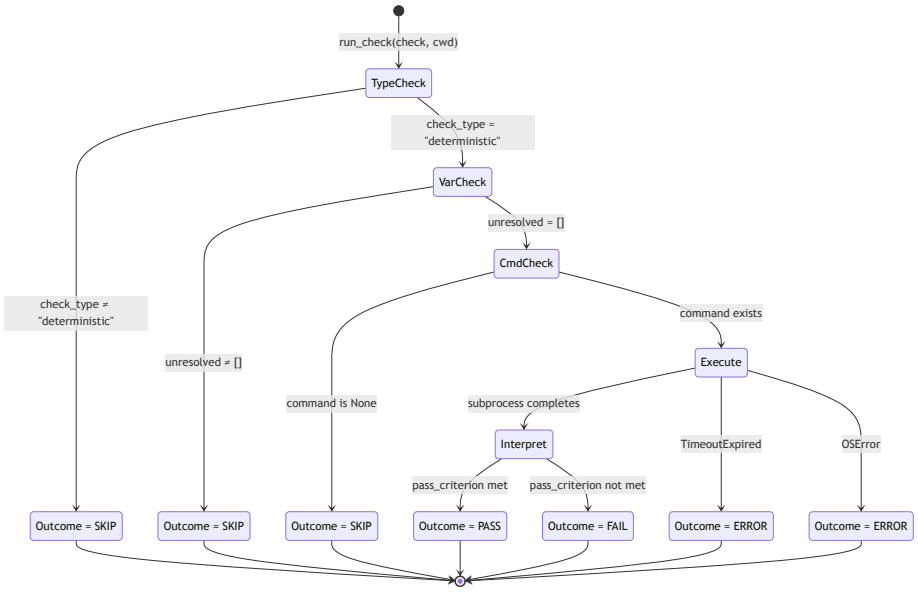

Diagram 5

Pass Criterion Interpretation

The `_interpret_result()` function interprets subprocess output against the `pass_criterion` string:

Criterion Pattern	Interpretation
`"exit code 0"` (or empty)	`returncode == 0` → PASS
`"coverage percentage >= N"`	Parse `(\d+)%` from stdout, compare to threshold
`"zero violations"` / `"zero errors"`	`returncode == 0` → PASS
Default fallback	`returncode == 0` → PASS

Checklist Aggregation

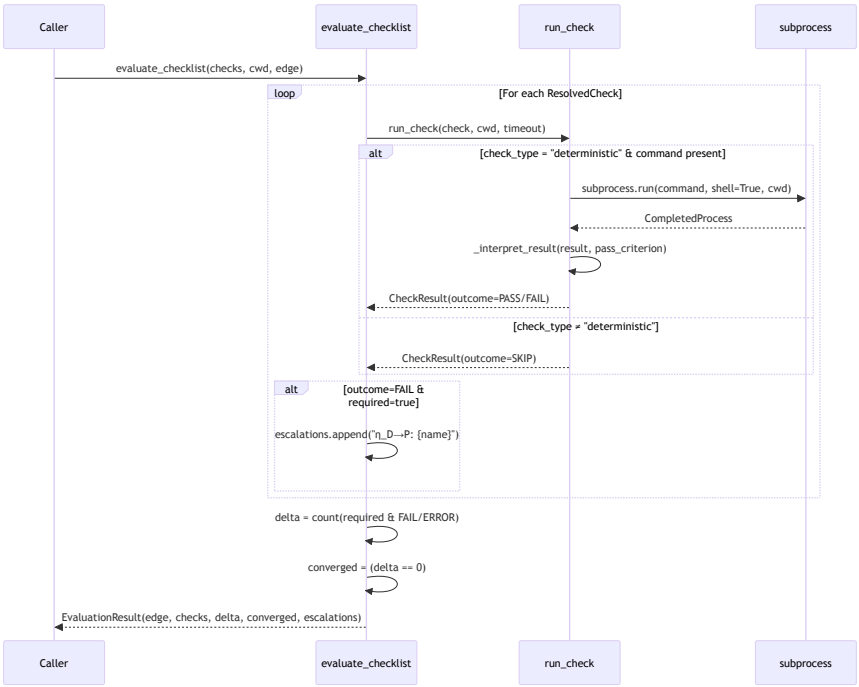

Diagram 6

Delta Formula

```
delta = Σ{ 1 | check ∈ checks, check.required ∧ check.outcome ∈ {FAIL, ERROR} }
converged = (delta == 0)
```

- `SKIP` outcomes (agent, human, unresolved) do **not** count toward delta
- Non-required (`required=false`) failures do **not** count toward delta
- Delta is a **non-negative integer** — the distance from convergence

6. The η (Natural Transformation) — Escalation Boundary

When a deterministic check fails, the framework surfaces an **escalation signal** that would hand off to the next-higher category. This is the natural transformation η: F_D → F_P → F_H.

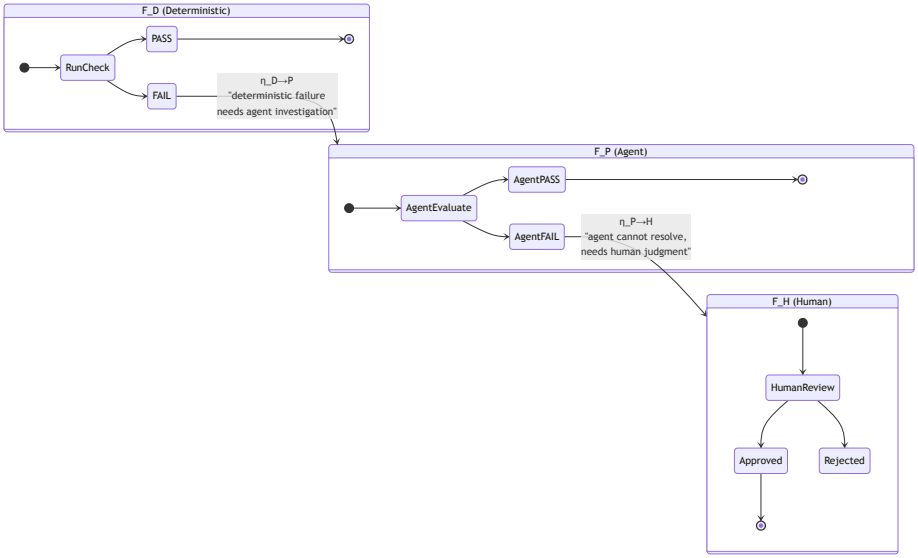

Diagram 7

Where η fires in the code

`fd_evaluate.py` — in `evaluate_checklist()`:

```
if cr.check_type == "deterministic" and cr.outcome in (FAIL, ERROR)
        and cr.required:
    escalations.append(f"η_D→P: {cr.name} failed — may need agent
        investigation")
```

`engine.py` — in `iterate_edge()`:

```
if cr.check_type == "deterministic":
    escalations.append(f"η_D→P: {cr.name} — deterministic failure")
elif cr.check_type == "agent":
    escalations.append(f"η_P→H: {cr.name} — agent evaluation failed")
```

The escalation signals are **informational** — the engine records them, but the current implementation does not yet automatically dispatch to F_P or F_H. That is future work.

7. F_D Emit — Event Emission

Emit is **category-fixed F_D** — it always fires, regardless of profile. The LLM cannot skip it.

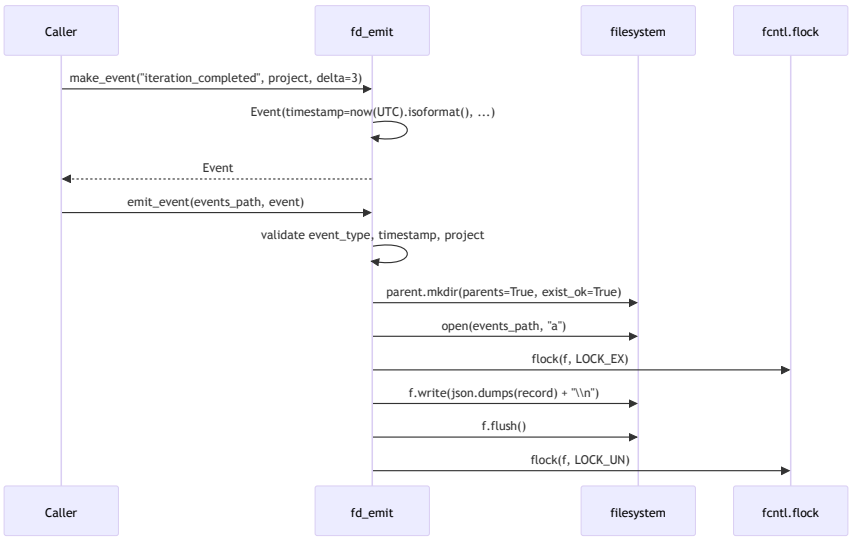

Diagram 8

Event Format (JSONL)

```
{"event_type":"iteration_completed","timestamp":"2026-02-
        24T10:30:00+00:00","project":"my_proj","feature":"REQ-F-AUTH-
        001","edge":"code↔unit_tests","delta":3,"status":"iterating"}
```

Event Types

Event Type	When Emitted
`project_initialized`	`/gen-init`
`iteration_completed`	Every iteration boundary
`edge_started`	Edge traversal begins
`edge_converged`	All required checks pass
`spawn_created`	Child vector spawned
`spawn_folded_back`	Child results returned
`checkpoint_created`	Session snapshot
`review_completed`	Human review done
`gaps_validated`	Traceability check

Event Type	When Emitted
`release_created`	Release package

---

8. F_D Classify — Deterministic Classification

Three classifiers, all regex/keyword-based (no LLM):

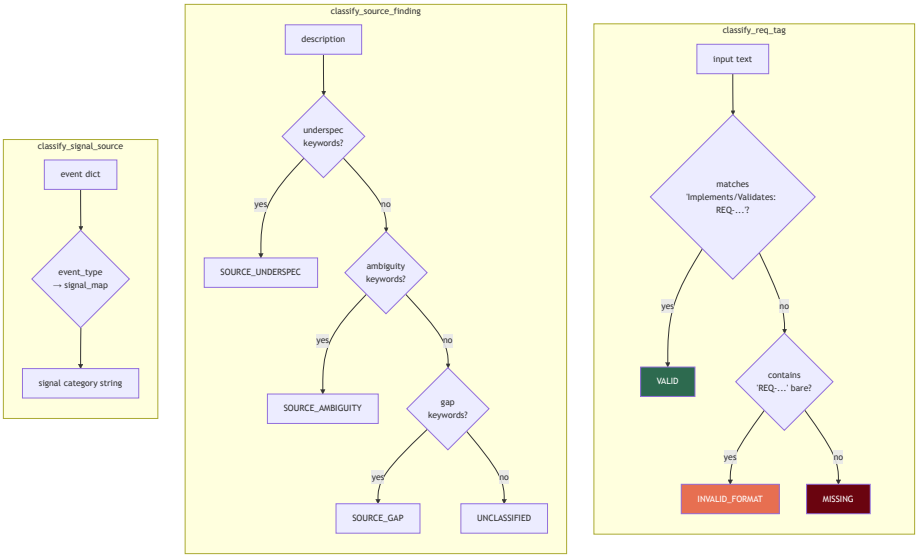

Diagram 9

Keyword Sets

Classification	Keywords
`SOURCE_UNDERSPEC`	underspecified, insufficient detail, needs clarification, placeholder
`SOURCE_AMBIGUITY`	unclear, ambiguous, vague, undefined, unspecified, unknown, tbd
`SOURCE_GAP`	missing, absent, gap, omitted, incomplete, not defined, lacks

Priority order: underspec → ambiguity → gap (first match wins).

---

9. F_D Sense — Interoceptive Monitors

Five monitors map to the spec's sensory system (INTRO-001 through INTRO-007):

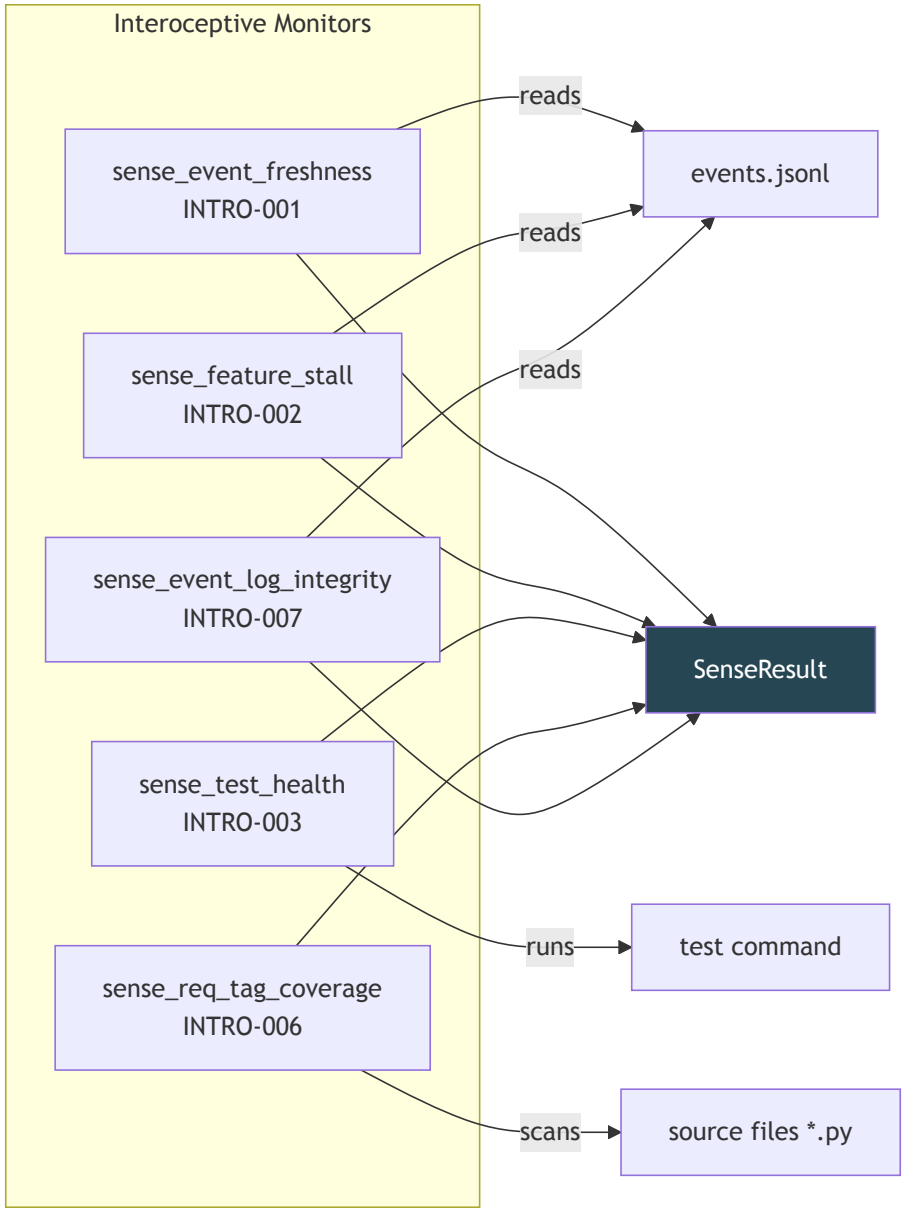
Interoceptive Monitors
sense_event_freshness
INTRO-001
sense_feature_stall
INTRO-002
sense_event_log_integrity
INTRO-007
sense_test_health
INTRO-003
sense_req_tag_coverage
INTRO-006
reads
reads
reads
events.jsonl
SenseResult
runs
test command
scans
source files *.py

Diagram 10

Stall Detection State Machine

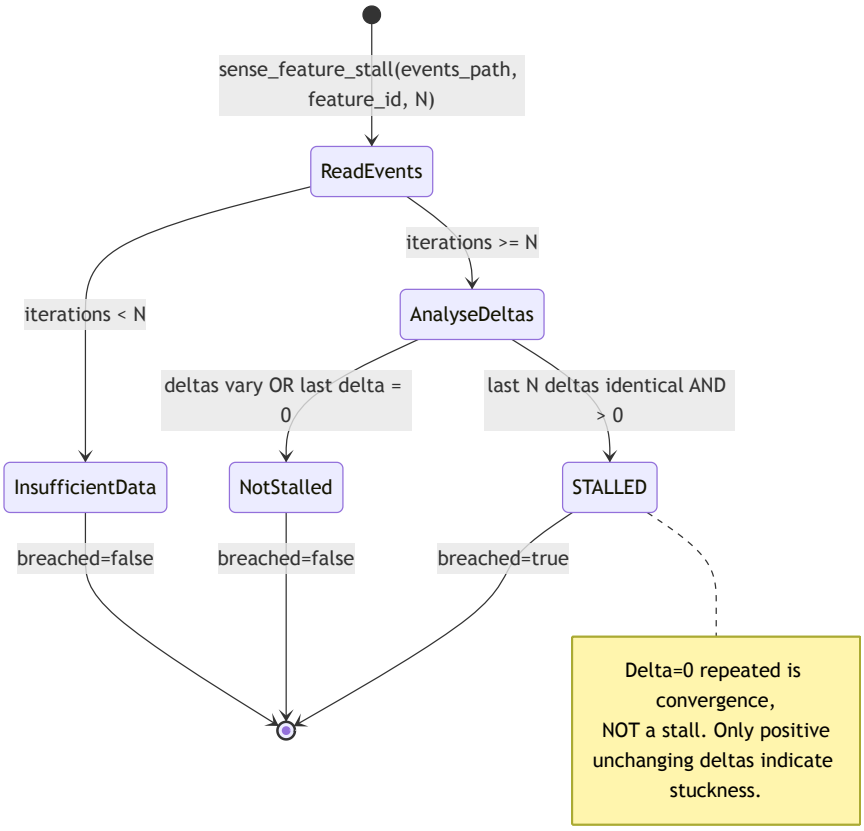

Diagram 11

10. F_D Route — Profile & Edge Selection

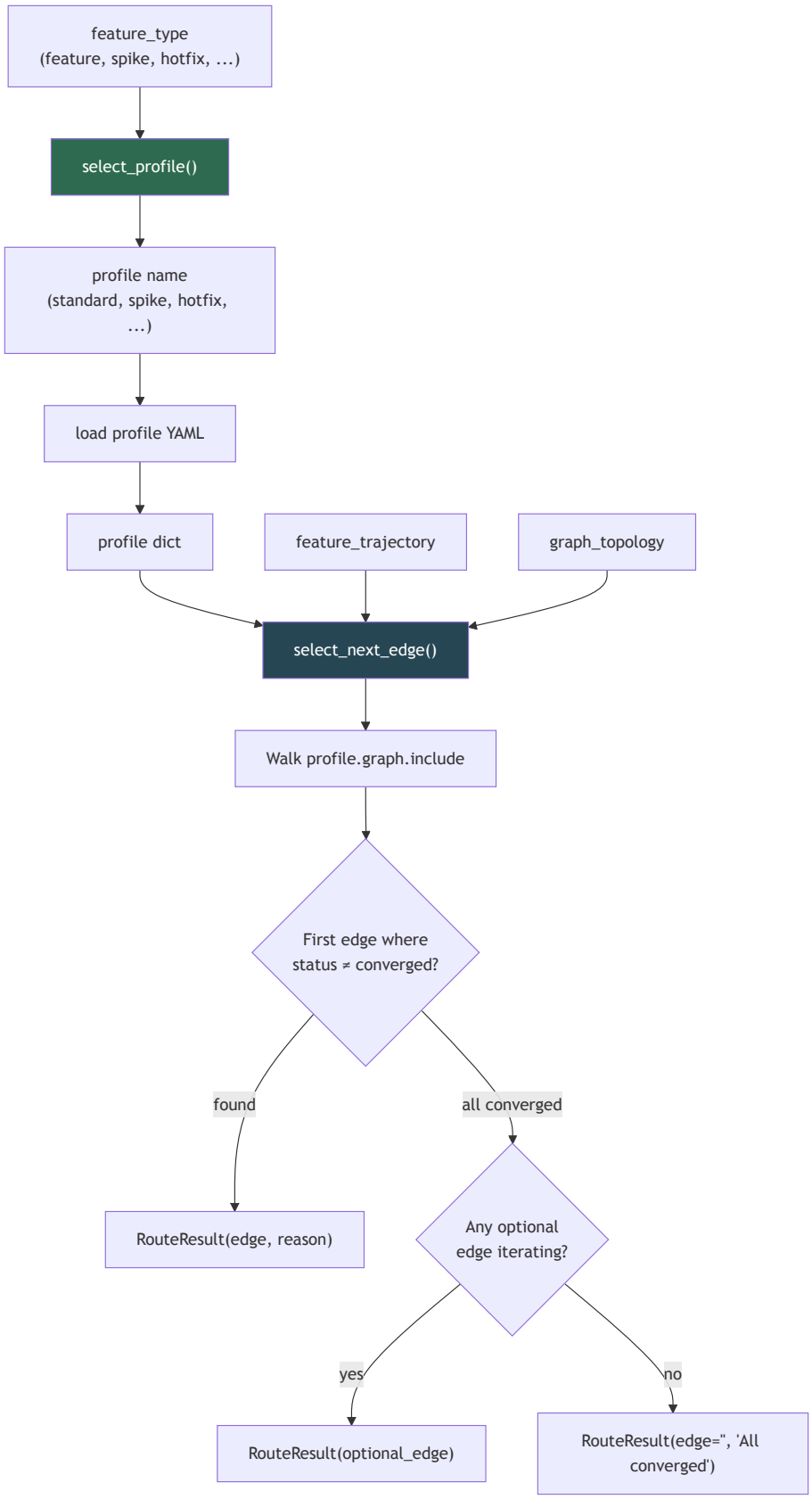

Diagram 12

Vector Type → Profile Mapping

Vector Type	Profile	Rationale
`feature`	`standard`	Normal development flow
`discovery`	`poc`	Exploration, lighter process
`spike`	`spike`	Time-boxed risk assessment
`poc`	`poc`	Proof of concept
`hotfix`	`hotfix`	Emergency, minimal process

Edge Naming Convention

Graph edges use Unicode arrows (→, ↔). Trajectory keys normalise these:

```
"code↔unit_tests" → trajectory key "code_unit_tests"
"intent→requirements" → trajectory key "intent_requirements"
```

---

11. Dispatch Table

The dispatch table maps `(FunctionalUnit, Category)` to a callable:

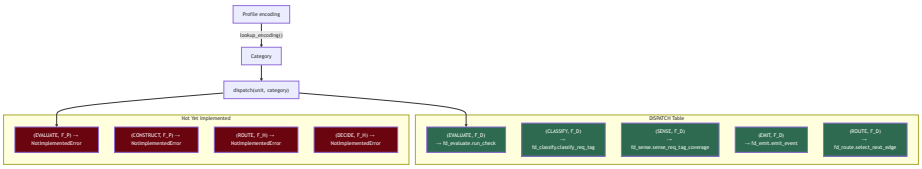

Diagram 13

`lookup_and_dispatch(unit, profile)` — End-to-End

```python
def lookup_and_dispatch(unit: FunctionalUnit, profile: dict) ->
        Callable:
    category = fd_route.lookup_encoding(profile, unit.value)  # Step
        1: profile → category
    return dispatch(unit, category)                            # Step
        2: table lookup
```

---

12. Profile Encoding Matrix

Each profile encodes the 8 functional units to categories differently:

Unit	Standard	Hotfix	Spike	PoC	Full	Minimal
evaluate	F_D	F_D	**F_P**	F_D	F_D	F_D
construct	F_P	F_P	F_P	F_P	F_P	F_P
classify	F_D	F_D	F_D	F_D	F_D	F_D
route	F_H	**F_D**	**F_P**	F_H	F_H	F_P
propose	F_P	F_P	F_P	F_P	F_P	F_P
sense	F_D	F_D	F_D	F_D	F_D	F_D
emit	F_D	F_D	F_D	F_D	F_D	F_D
decide	F_H	F_H	F_H	F_H	F_H	F_H

Key variations: - **Spike** flips `evaluate` to `F_P` — exploration code isn't test-driven - **Hotfix** flips `route` to `F_D` — fixed emergency path, no human routing - **Minimal** flips `route` to `F_P` — agent picks route - `emit` (F_D) and `decide` (F_H) are **invariant** across all profiles

---

13. Engine — Graph Traversal Loop

The engine (`engine.py`) is the top-level deterministic controller. It owns the loop; the LLM is called from within.

Full Traversal Sequence

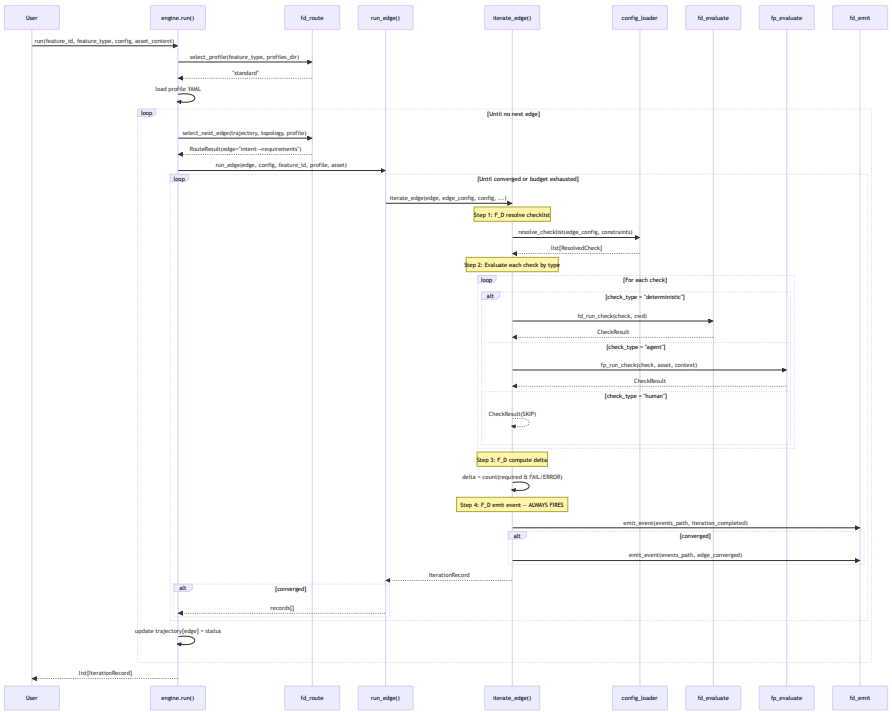

Diagram 14

Engine State Machine

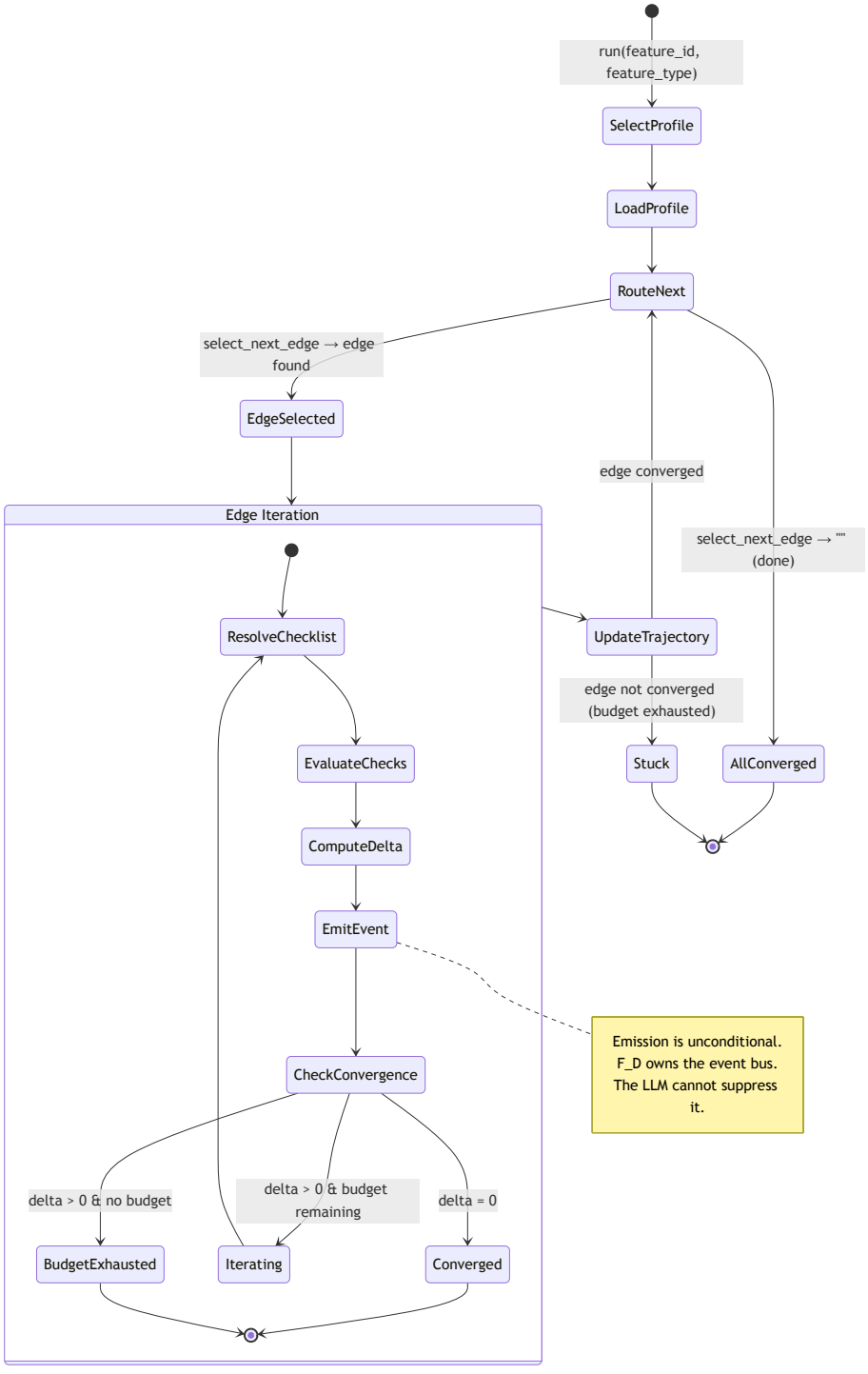

Diagram 15

---

14. F_P Evaluate — LLM Integration

`fp_evaluate.py` wraps the Claude Code CLI for agent-based evaluation:

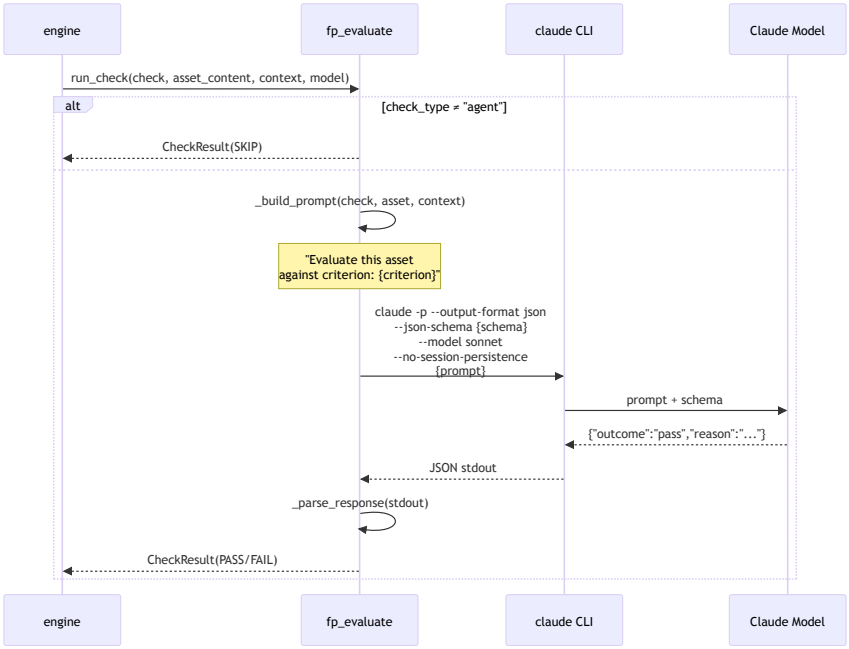

Diagram 16

Response Schema (JSON Schema)

```
{
    "type": "object",
    "properties": {
        "outcome": {"type": "string", "enum": ["pass", "fail"]},
        "reason": {"type": "string"}
    },
    "required": ["outcome", "reason"]
}
```

---

15. Data Flow — Complete Pipeline

This diagram shows how data flows through the entire system for one iteration:

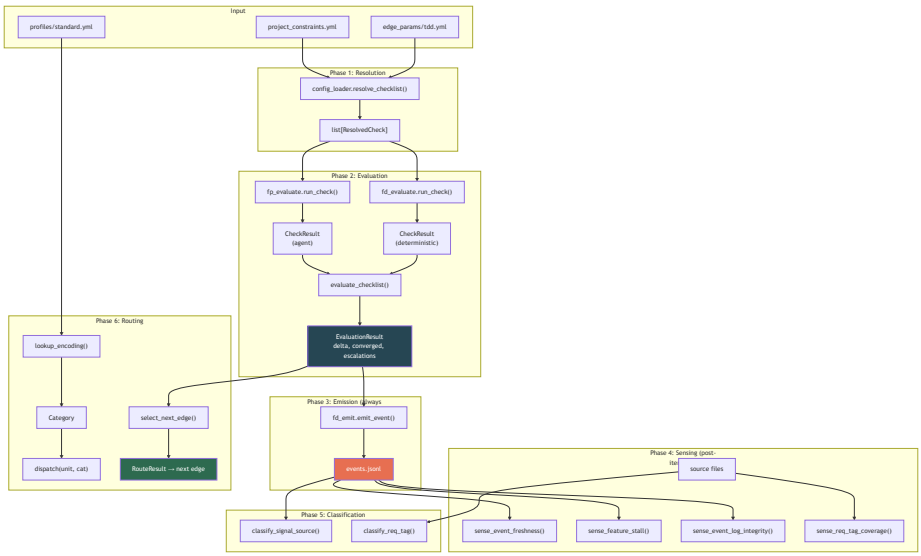

Diagram 17

---

16. Test Architecture

Test Files

File	Tests	What it covers
`test_config_loader.py`	16	`resolve_variable`, `resolve_variables`, `resolve_checklist`, `load_yaml`, real config integration
`test_functor_fd.py`	39	Unit tests for all F_D modules: evaluate, emit, classify, sense, route, dispatch
`test_functor_e2e.py`	50	6 end-to-end scenarios wiring the full pipeline

E2E Test Scenarios

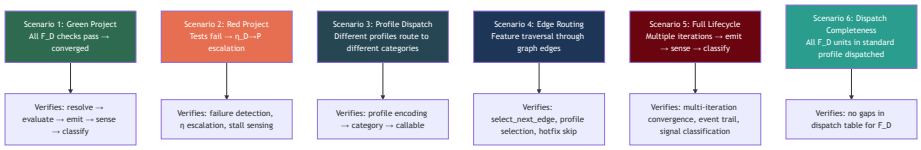

Diagram 18

---

17. Dependencies

stdlib only (except PyYAML):

Dependency	Used by	Purpose
`dataclasses`	`models.py`	Data model definitions
`enum`	`models.py`	Category, FunctionalUnit, CheckOutcome
`subprocess`	`fd_evaluate.py`, `fd_sense.py`, `fp_evaluate.py`	Shell command execution
`fcntl`	`fd_emit.py`	Advisory file locking for JSONL append
`json`	`fd_emit.py`, `fd_sense.py`, `fp_evaluate.py`	Event serialization/parsing
`re`	`config_loader.py`, `fd_classify.py`, `fd_sense.py`	Pattern matching
`yaml` (PyYAML)	`config_loader.py`	YAML parsing
`shutil`	`fp_evaluate.py`	`which()` to find claude CLI

---

18. What's Not Implemented Yet

Gap	Category	Notes
F_P modules (classify, route, sense)	F_P	`fp_evaluate.py` exists for evaluate only
F_H modules (all)	F_H	Interactive prompts — future work
Automatic η dispatch	η	Escalation signals are recorded but not auto-dispatched
CLI entry point	Infra	No `python -m genisis` yet
Feature constraint merging	Config	`feature.threshold_overrides` + `acceptance_criteria` not yet composed

Gap	Category	Notes
Construct / Propose	F_D/F_P	No modules — these are always F_P (agent-generated)