Framework Comparison Analysis: E2E Agent vs Deterministic Engine

Date: 2026-02-24 **Version**: 1.0 **Implements**: Cost-benefit analysis between two execution models

---

1. The Two Frameworks

1.1 E2E Agent Framework (Headless Claude)

One `claude -p` session drives the **entire** methodology loop: - Reads configs, resolves $variables, routes edges - **Constructs** artifacts (requirements, design, code, tests) - **Evaluates** all checks (agent checks happen in the same LLM session) - Emits events, computes delta, decides convergence - Routes to next edge, repeats

Key insight: Agent checks labelled "agent" in the checklist are evaluated **within the same Claude session** that's orchestrating everything. There are no separate subprocess calls. The LLM does both orchestration AND evaluation in one continuous context.

1.2 Deterministic Engine Framework (`engine.py`)

Python code drives the loop deterministically: - `config_loader.py` loads YAML, resolves $variables - `fd_route.py` selects edges, profiles, encodings - `fd_evaluate.py` runs deterministic checks (subprocess) - `fp_evaluate.py` calls `claude -p` per agent check (**separate subprocess per check**) - `fd_emit.py` appends events to JSONL - `engine.py` computes delta, decides convergence

Key insight: The engine makes **one `claude -p` subprocess call per agent check**. Each call starts a fresh LLM session with no shared context. The engine does NOT implement F_P construct — it can evaluate but cannot generate artifacts.

---

2. What Each Framework Actually Does

Capability	E2E Agent	Engine
Load YAML configs	LLM reads files	`config_loader.py` (deterministic)

Capability	E2E Agent	Engine
Resolve $variables	LLM text processing	`resolve_variable()` regex (deterministic)
Route to next edge	LLM reads profile	`fd_route.select_next_edge()` (deterministic)
Construct artifacts	**LLM generates**	**NOT IMPLEMENTED**
Evaluate (deterministic)	LLM runs shell commands	`fd_evaluate.run_check()` (subprocess)
Evaluate (agent)	**Same LLM session**	**Separate `claude -p` per check**
Evaluate (human)	LLM auto-passes in –auto	`CheckOutcome.SKIP`
Emit events	LLM appends JSONL	`fd_emit.emit_event()` (deterministic)
Compute delta	LLM counts failures	`sum(1 for cr if FAIL/ERROR)` (deterministic)
η escalation	LLM reports	Pattern matching on check outcomes

---

3. LLM Call Count Comparison

3.1 Standard 4-Edge Profile: `intent→requirements→design→code↔unit_tests`

Checks per edge (from edge_params YAML):

Edge	Deterministic	Agent	Human	Total
intent→requirements	0	12	2	14
requirements→design	0	14	1	15
design→code	5	3	0	8
code↔unit_tests	8	7	0	15
TOTAL	**13**	**36**	**3**	**52**

Required agent checks: **33** (3 optional: diagrams_present, advisory_dimensions_considered, code_req_keys_have_telemetry)

3.2 E2E Agent: LLM Calls

Component	LLM Calls	Notes
Full orchestration session	**1**	Single `claude -p` session for everything
Agent evaluator checks	**0 additional**	Evaluated within the same session
Artifact construction	**0 additional**	Generated within the same session
Total LLM calls	**1**	

The single session handles: 4 constructions + 36 agent evaluations + routing + emission = all in one context window.

3.3 Engine Framework: LLM Calls (if fully implemented)

Component	LLM Calls	Notes
Orchestration	**0**	Deterministic Python code
Artifact construction (F_P)	**4**	One `claude -p` per edge (construct)
Agent evaluator checks	**33**	One `claude -p` per required agent check
Total LLM calls	**37**	Each starts a fresh session

---

4. Cost Model

4.1 Token Economics

E2E Agent (1 session): - Context accumulates across the entire run - Early edges are cheap (small context) - Late edges are expensive (full project in context) - Typical run: ~780s, ~$2-5 budget (from E2E metadata)

Engine (37 separate sessions): - Each `claude -p` call starts fresh — no accumulated context - Each evaluation call: ~2K tokens prompt + response (check criterion + asset content) - Each construction call: ~5-10K tokens (similar to one E2E edge) - BUT: no

shared context means each call must re-read the asset from scratch - No cross-check coherence (check N doesn't know check N-1's result)

4.2 Cost Estimate: Engine Framework (per iteration)

Assuming sonnet model, standard 4-edge profile:

Call Type	Count	Est. Tokens/Call	Total Tokens
Construct (intent→req)	1	~15K	15K
Construct (req→design)	1	~20K	20K
Construct (design→code)	1	~15K	15K
Construct (code↔tests)	1	~15K	15K
Agent eval (intent→req)	11	~3K	33K
Agent eval (req→design)	13	~4K	52K
Agent eval (design→code)	3	~4K	12K
Agent eval (code↔tests)	6	~4K	24K
TOTAL	**37**		**~186K tokens**

E2E Agent (single session): - Run 0011: 780s, 4 edges, single session - Typical sonnet session: ~100-200K tokens total (input + output) - Context reuse means later edges benefit from earlier context

4.3 The Hidden Cost: Context Coherence

The engine's per-check `claude -p` calls lose **context coherence**:

Property	E2E Agent	Engine
Check sees previous check results	Yes	No
Check sees constructed artifact	Yes (in context)	Must pass as `asset_content`
Check sees project constraints	Yes (read once)	Must include in prompt each time
Check can reference design decisions	Yes (accumulated)	No cross-reference
Evaluator understands "why"	Yes (narrative context)	No (just criterion + asset)

This means the engine's agent checks are **shallower** — they evaluate the criterion in isolation without the rich context that the E2E agent has accumulated.

---

5. Test Suite Comparison

5.1 Three Test Suites

Suite	Tests	Time	LLM Calls	What It Tests
`test_functor_e2e.py`	24	0.76s	0	Hand-wired module pipeline
`test_functor_uat.py`	33	18.9s	0*	Engine API pipeline
`e2e/test_e2e_convergence.py`	34	~780s	1	Full headless Claude

*UAT tests get 0 LLM calls because agent checks ERROR (can't nest Claude sessions). In production the engine would make 33 LLM calls per iteration.

5.2 Timing Breakdown (UAT tests)

From `--durations=0`:

Test Category	Slowest	Time	Bottleneck
EdgeRouting (engine.run())	hotfix_skips_design	1.57s	Multiple edges, each resolves YAML + runs checks
RedProject (run_edge)	stall_detected	1.45s	3 iterations × deterministic subprocess
CrossMethod (iterate_edge)	green_delta	0.89s	Single edge, subprocess for tests
GreenProject (iterate_edge)	green_pipeline	0.58s	Single edge, subprocess for tests
ProfileDispatch	all_profiles_emit	0.01s	Pure Python, no subprocess

Test Category	Slowest	Time	Bottleneck
DispatchCompleteness	all_fd_dispatched	<0.01s	Pure Python, no subprocess

Observation: UAT time is dominated by subprocess calls for deterministic checks (pytest, py_compile). Agent checks ERROR immediately (no `claude` CLI in test env), so the 18.9s is the **floor** — production would add ~33 × 10-30s per LLM call.

5.3 E2E Convergence Runs (Historical)

Run	Elapsed	Edges	Iterations	Result	Notes
0011 (latest)	780s	4/4	4×1=4	34/34 pass	Standard convergence
0010	210s	1/1	2×1=2	34/34 pass	Homeostasis (code↔tests only)
0009	890s	4/4	4×1=4	34/34 pass	Standard convergence
0007	~780s	4/4	4×1=4	34/34 pass	Standard convergence
0004	timeout	-/4	-	FAILED	Stall during convergence

Average successful standard run: **~816s (~13.6 min)**

---

6. What the Engine Doesn't Do (Yet)

6.1 Missing Capabilities

Capability	Status	Impact
F_P Construct	NOT IMPLEMENTED	Can't generate requirements, design, code, tests
F_P Evaluate (batched)	One call per check	33 cold-start sessions vs 1 warm session
F_H Evaluate	SKIP	Human checks silently skipped

Capability	Status	Impact
Context accumulation	None	Each LLM call is stateless
Source analysis	NOT IMPLEMENTED	Backward gap detection requires LLM
Process gap detection	NOT IMPLEMENTED	Inward gap detection requires LLM

6.2 The Fundamental Gap

The engine framework is an **evaluator** but not a **builder**. It can: - Run deterministic checks (subprocess) - Dispatch to LLM for individual semantic checks - Route, emit, compute delta

It cannot: - Generate artifacts (requirements, design, code, tests) - Accumulate context across checks - Do source analysis or process gap detection - Make coherent evaluations that reference earlier results

The E2E agent does **all of these** in a single session.

---

7. Cost-Benefit Summary

7.1 Engine Framework Advantages

Benefit	Value
Deterministic orchestration	Routing, emission, delta are reproducible
Testable without LLM	57 tests run in <20s
Observable	Every step is a Python function call, debuggable
Configurable	Edge params, profiles, constraints are YAML
Repeatable	Same inputs → same deterministic outputs
Parallelisable	Agent checks could run concurrently (future)

7.2 Engine Framework Costs

Cost	Impact
33 cold-start LLM calls per iteration	~$2-8 per run (comparable to E2E)
No context coherence	Shallow evaluations, no cross-check reasoning
No construction	Can't build artifacts — only evaluate them
Session overhead	Each `claude -p` has ~5-10s startup cost (37 × 5s = 185s just starting)
Asset re-reading	Each call must re-read the asset (no shared memory)

7.3 E2E Agent Advantages

Benefit	Value
Full pipeline	Constructs AND evaluates in one session
Context accumulation	Later checks benefit from earlier reasoning
Coherent evaluation	Agent sees the whole picture
Single session	One startup cost, warm context throughout
Source analysis	Can identify ambiguities, gaps, underspec

7.4 E2E Agent Costs

Cost	Impact
Non-deterministic orchestration	Routing decisions are LLM-dependent
Untestable without LLM	34 tests require ~13min Claude run
Opaque	Decision path is in LLM reasoning, not code
Context window growth	Late edges pay for accumulated context
Single point of failure	If the session crashes, everything restarts

---

8. The Three Processing Layers

The spec (§4.6) defines three processing phases that map directly onto the cost structure:

8.1 Reflex (F_D) — Free, Deterministic

Template-driven synthesis where the output is mechanically derived from inputs. No LLM. No judgment. Pure data transform.

```
Input (structured)  →  Rule/Template  →  Output (structured)
```

Already implemented in the engine: - `fd_emit.py` — event construction from structured data - `config_loader.py` — $variable resolution, checklist construction - `fd_route.py` — profile selection, next edge, encoding lookup - `fd_evaluate.py` — subprocess execution, pass criterion checking - `fd_classify.py` — regex classification, keyword matching - `fd_sense.py` — file scanning, staleness detection, integrity checks - `engine.py` — delta computation, convergence decision, iteration loop

F_D construct examples (reflex-level synthesis): - Event emission (structured data → JSONL line) - Feature vector trajectory updates (status changes) - Traceability reports (grep + format) - Checklist report formatting (results → table) - Context hash computation

These are the **lowest level of reflex actions** — basic, template-driven.

8.2 Affect (F_P) — Costly, LLM-Driven

Judgment-based generation and evaluation. Requires the LLM.

Construction (generates artifacts): - Intent → requirements document (creative) - Requirements → design document + ADRs (architectural judgment) - Design → code (implementation judgment) - Code ↔ tests (test strategy, coverage judgment)

Evaluation (semantic assessment): - "Does the code match the design?" — requires understanding both - "Are acceptance criteria specific?" — requires domain judgment - "Is test quality adequate?" — requires testing expertise - Source analysis: ambiguity, gap, underspec detection

This is where the LLM earns its keep. These tasks cannot be template-driven.

8.3 Conscious (F_H) — Blocking, Human-Driven

Decisions and approvals that require human authority.

- Architecture approval
- Priority validation
- Completeness sign-off
- Scope decisions

Currently: auto-passed in E2E (`--auto` mode), skipped in engine.

8.4 Cost by Layer

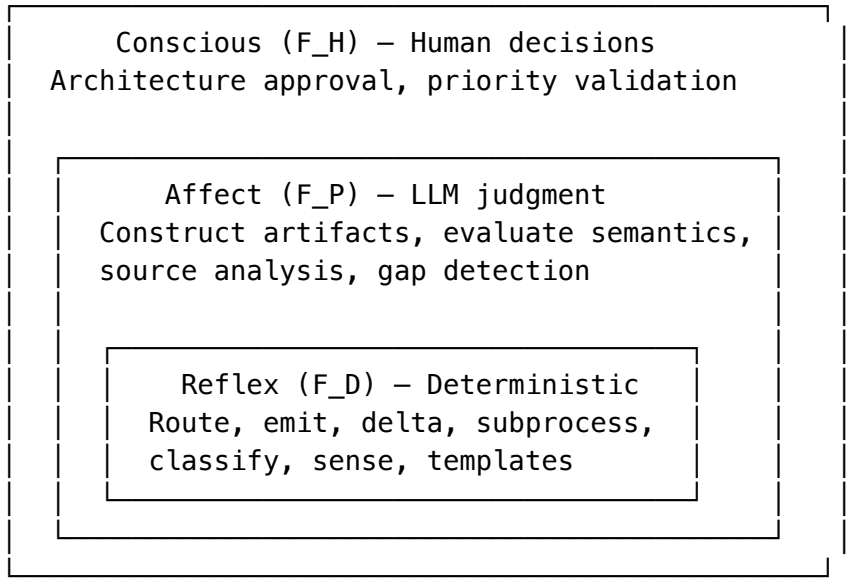
```
Reflex (F_D)    — templates, transforms, rules    → FREE
(deterministic)
Affect (F_P)    — judgment, generation, evaluation → COSTLY (LLM
calls)
Conscious (F_H) — decisions, approvals             → BLOCKING (human
wait)
```

The engine already has the full reflex layer. The cost question is entirely about **how to structure the affect layer's LLM calls**.

---

9. Architectural Verdict

The two frameworks are **not competing alternatives** — they're **complementary layers**:




```
E2E Agent: One LLM session does ALL three layers
Engine:    Reflex is code, Affect is per-check LLM calls, Conscious is
skip
Target:    Reflex is code, Affect is per-EDGE LLM calls, Conscious is
UI
```

The Optimal Architecture

1. **Engine owns the loop** — reflex layer is deterministic Python
2. **One LLM call per edge** — construct + evaluate in a single session
3. **Deterministic checks** run as subprocess before LLM call
4. **LLM receives**: asset + context + all agent criteria → returns: artifact + evaluations
5. **Engine validates** LLM output deterministically (parsing, format, traceability)

What Needs to Change

Component	Current	Target
`fp_evaluate.py`	1 call per check (33 cold starts)	1 call per edge (batched criteria)
`fp_construct.py`	Not implemented	Combined with evaluate in per-edge call
Engine loop	Evaluate only, no construct	Construct + evaluate per edge
Context	None (each call is stateless)	Accumulated context passed between edges

Projected Cost: Hybrid Architecture

Call Type	Count	Notes
Edge 1: construct + evaluate	1	Intent → requirements + 11 agent checks
Edge 2: construct + evaluate	1	Requirements → design + 13 agent checks
Edge 3: construct + evaluate	1	Design → code + 3 agent checks
Edge 4: construct + evaluate	1	Code ↔ tests + 6 agent checks
Total LLM calls	**4**	One per edge (vs 37 current, vs 1 E2E)

This gives: deterministic orchestration + coherent per-edge LLM context + testable without LLM.

---

10. Telemetry Comparison: All Four Runs

10.1 Run Summary Table

Run	Type	Elapsed	Edges	Iterations	Tests	Result
E2E 0013 (fresh)	Convergence	400s	4/4	4×1=4	33/34	1 fail (timestamp)
E2E 0012 (fresh)	Homeostasis	230s	1+3*	1+1=2	12/12	PASS
E2E 0011 (prior)	Convergence	780s	4/4	4×1=4	34/34	PASS
UAT (engine)	Unit test	18.9s	varied	varied	33/33	PASS
Hand-wired	Unit test	0.76s	N/A	N/A	24/24	PASS

*Homeostasis: 3 edges auto-converged, 1 edge (code↔unit_tests) iterated.

10.2 E2E Convergence Events (Run 0013)

```
project_initialized  → 1 event
edge_started         → 4 events
iteration_completed  → 4 events (all converge iter 1, delta=0)
edge_converged       → 4 events
TOTAL: 13 events in 400s
```

Per-edge evaluator summary: - intent→requirements: 11/11 agent pass (0 deterministic, 0 human) - requirements→design: 4/4 pass - design→code: 3/3 pass - code↔unit_tests: 13/13 pass

Observation: No evaluator_detail events emitted. All first-pass convergence (delta=0 everywhere). No observable RED phase.

10.3 E2E Homeostasis Events (Run 0012)

```
project_initialized   → 1 event
edge_started          → 4 events
edge_converged        → 4 events (3 instant + 1 after iteration)
iteration_completed   → 2 events (code↔unit_tests: delta [7, 0])
evaluator_detail      → 7 events (per-failure records)
TOTAL: 18 events in 230s
```

The homeostasis run demonstrates the full iterate protocol:

Iteration	Evaluator	Result	Type	Evidence
1 (RED)	tests_pass	FAIL	deterministic	pytest exit code 1
1 (RED)	AC-CONV-001	FAIL	deterministic	c_to_f(0)=30 not 32
1 (RED)	AC-CONV-002	FAIL	deterministic	c_to_f(100)=230 not 212
1 (RED)	AC-CONV-003	FAIL	deterministic	f_to_c(32)=1.0 not 0.0
1 (RED)	AC-CONV-004	FAIL	deterministic	f_to_c(212)=91 not 100
1 (RED)	AC-CONV-007	FAIL	deterministic	f_to_k(32)=274.15
1 (RED)	AC-CONV-008	FAIL	deterministic	k_to_f(273.15)=30
2 (GREEN)	all 19	PASS	mixed	All formulas corrected

Delta sequence: $[7, 0]$ — homeostatic correction in 30 seconds.

10.4 UAT Engine Events (Test Environment)

Green project `iterate_edge("code↔unit_tests")`: - 1 iteration_completed event (delta = agent_error_count, not 0) - Deterministic checks: all PASS - Agent checks: all ERROR (can't nest Claude sessions) - No edge_converged event (delta > 0 due to agent errors)

Red project `run_edge("code↔unit_tests")`: - 3 iteration_completed events (budget exhaustion) - Deterministic checks: tests_pass FAIL, others PASS - Agent checks: all ERROR - η escalation: `η_D→P: tests_pass — deterministic failure`

10.5 Key Telemetry Differences

Signal	E2E (convergence)	E2E (homeostasis)	Engine (test env)
Det. delta (green)	0	0 (iter 2)	0
Det. delta (red)	N/A	7 (iter 1)	1+
Agent delta	0 (in-session)	0 (in-session)	N (all ERROR)
Total delta	0	$7 \rightarrow 0$	N (agent errors inflate)

Signal	E2E (convergence)	E2E (homeostasis)	Engine (test env)
Convergence	Yes (1 iter)	Yes (2 iter)	No (agent errors)
η triggers	None	None	From det. failures
evaluator_detail	Not emitted	7 events (failures)	N/A
Events richness	Aggregate only	Per-check breakdown	Aggregate only

10.6 Artifact Quality Variance Across Runs

Metric	Run 0013	Run 0012 (homeostasis)	Run 0011	Run 0009
Source LOC	68	67	154	169
Test LOC	189	75	273	405
Test count	~20	~7	40	22
REQ tags (source)	9	10	13	2
REQ tags (tests)	9	9	3	2

LLM-generated artifacts vary 2-6x in size across runs. The methodology converges correctly regardless — the evaluators catch what matters, not how much code exists.

---

10. Recommendations for Next Steps

10.1 Immediate: Batch Agent Evaluation

Modify `fp_evaluate.py` to batch all agent checks for an edge into **one `claude -p` call** with a multi-criteria JSON schema response. This reduces 33 calls → 4 calls.

10.2 Short-term: Add F_P Construct

Implement `fp_construct.py` that calls `claude -p` to generate artifacts per edge. The engine then evaluates the constructed artifact.

10.3 Medium-term: Combined Construct+Evaluate Call

One LLM call per edge that both constructs the artifact AND evaluates all agent checks. The engine validates the structured output deterministically.

10.4 Testing Strategy

Test Level	What	LLM Required	Time
Unit (test_functor_fd.py)	Individual modules	No	<1s
Hand-wired E2E (test_functor_e2e.py)	Module pipeline	No	<1s
Engine UAT (test_functor_uat.py)	Engine + deterministic	No	~19s
Engine UAT + LLM	Engine + real agent checks	Yes	~5-10min
Full E2E (e2e/test_e2e_convergence.py)	Headless Claude	Yes	~13min

10.5 Complex Scenario Tests (Future)

To find the cost-benefit crossover:

1. **Multi-iteration convergence**: Project with deliberate failures (red→green→red→green)
2. **Multi-feature**: 3+ features converging simultaneously
3. **η escalation chains**: Deterministic failure → agent re-evaluation → human escalation
4. **Profile switching**: Start as spike, promote to feature (re-traversal)
5. **Stale context**: Engine with accumulated context vs fresh sessions
6. **Parallel evaluation**: Engine dispatches agent checks concurrently