

ARGUS

Contributor's Guide

1. What ARGUS Is

ARGUS catches silent failures in AI agent pipelines **before deployment** — no real users are affected. Attach the adapter to your LangGraph pipeline, run the file normally, and ARGUS monitors every node as it executes.

What it catches

- Silent tool-call failures (error payloads the pipeline swallows)
- Structural mismatches (missing fields a successor expects)
- Semantic degradation (placeholder outputs, repeated filler, corrupted LLM JSON)
- Behavioral anomalies (output size spikes, timing outliers)

What it reports

- Root cause chain — traces backward to where the failure originated
- Per-node status: pass / fail / crashed / semantic_fail / interrupted
- Replay from any step

Machine-readable output

ARGUS produces a structured JSON alongside the dashboard. This JSON is designed for another AI to consume directly — failure locations, root cause chains, and (upcoming) a fixture prompt telling the AI exactly what broke and how to fix it. No human needs to open a dashboard and interpret a trace. ARGUS hands the diagnosis to the machine.

2. The Problem

Agent pipelines fail silently. A tool returns empty, an LLM says "I can't help with that," a retrieval step returns zero documents — none of these raise exceptions. The pipeline reports success and ships a wrong result. Existing tools (LangSmith, LangFuse, Arize) monitor *after* deployment. ARGUS operates *before* — during development and CI.

Target audience

- Solo devs building agent pipelines who want confidence before shipping
- Enterprise teams needing a CI gate for agent correctness

3. Vision

ARGUS should be the default debugging and observability layer for AI agent development. The target workflow: a dev finishes a pipeline, prompts their AI assistant "run this through ARGUS," the AI runs it, reads the ARGUS JSON, auto-fixes failures, replays from the broken node, and reports back clean.

Open question: when the AI is testing a pipeline that needs human inputs (user queries, uploads, approvals), how does it supply those realistically? We don't have a clean answer yet.

4. Current Status

ARGUS works. The detection pipeline catches real failures. But it is early software.

Works well

- Tool failure scanning, structural inspection, semantic signatures
- Root cause analysis on linear pipelines
- `argus check` and `pytest --argus` CI integration

Needs work

- **Complex graphs:** loops, conditionals, and parallel fan-out produce inconsistent results
- **False positives:** semantic checker flags valid short/unusual LLM responses
- **Detection bugs:** edge cases with nested tool calls, valid empty results, intentionally dropped fields
- **Latency:** 20+ node pipelines add noticeable overhead (LLM judge is the bottleneck)
- **Fixture prompt:** not yet implemented — JSON provides diagnosis but not prescription

5. What You Can Touch

Open (Apache-2.0)	Closed (PRs will be closed)
<code>src/argus/</code> — detection, CLI, adapter, UI	<code>cloud/</code>
<code>tests/</code> , <code>fixtures/</code> , <code>docs</code> , <code>website/</code>	<code>supabase/</code>
<code>CONTRIBUTING.md</code> , <code>skills</code> , <code>signatures</code>	<code>LICENSE</code> , <code>pyproject.toml</code> , <code>.github/</code> (without maintainer approval)

LangGraph only. No CrewAI/ADK/AutoGen adapters — open an "Adapter interest" issue instead. Core `wrap()/NodeEvent` API is still moving.

Fastest first PRs: fixtures, signatures, unit tests, docs.

6. Local Setup

```
git clone <your-fork>
cd ARGUS
pip install -e ".[dev]"
ruff check .
ruff format src/
pytest tests/
```

`mypy src/argus` is nice-to-have (CI doesn't run it). `ruff` + `pytest` will block merge. **No AI co-author trailers on commits.**

7. CLA (One-Time)

First PR triggers a CLA signature so your code can stay Apache-2.0 and ship in the hosted product.

- CLA Assistant comments on your PR.
- Reply with exactly: *I have read the CLA Document and I hereby sign the CLA*
- Stored on `cla-signatures` branch. Never again for that account.
- Blocked merge + green CI? Likely unsigned CLA — comment recheck after signing.
- **Employer IP:** if writing at work, get permission before signing. Read `CLA.md`.

8. PR Process

Default branch is `master`. No direct pushes. Only maintainers merge.

- Fork → branch off `master` → one topic per branch
- New detection: include a fixture the old code misses
- API/behavior changes: update `CLAUDE.md` / docs in the same PR
- Issue first for anything bigger than a small fix
- Security bugs: `SECURITY.md` or email `varaddur@gmail.com` — not a public issue

9. Merge Gates

Check	Runs	Common fail
CI (3.9, 3.11, 3.12)	<code>ruff check . + pytest</code>	Lint, import, test failure
CLA	Signature on file	Unsigned / new co-author
Code owner review	CODEOWNERS	No maintainer approval

Will be closed even if green: touches `ccloud//supabase/`, new framework adapter, large rewrites without fixtures, secrets in diff, investigation code without tests.

Green CI does not mean it will merge. Review still decides.

10. What We Expect

- **Own at least one issue** — pick it, comment, see it through.
- **Submit at least one detection logic PR** — a signature, fixture, or inspector improvement.

11. Quick Reference

Do	Don't
Fork, branch, PR to <code>master</code>	Push to <code>master</code>
Sign the CLA with the exact comment	Skip the CLA
<code>ruff + pytest</code> locally before PR	Assume green CI = merged
One concern per PR; fixture for detection	Start a new framework adapter

Ask on an issue if unsure about scope	Edit cloud/ or supabase/
Read CONTRIBUTING.md first	Add AI co-author trailers
	File security bugs publicly