



Agentic AI Foundation

AGNTCon + MCPCon North America

Where the **agentic stack** is being built.

Harness Engineering: From MAST's Failure Taxonomy to MCP-Native Multi-Agent Production

Jay Mehta | Principal Solutions Architect and AI Practice Head, Cygnet Infotech

Agentic Engineering track | AGNTCon + MCPCon North America 2026 | San Jose, Oct 22-23

Basecamp: multi-agent systems fail often

41–86.7%

failure rates across seven open-source multi-agent systems in the MAST study.

2025-era systems on different benchmarks; per-system rates are not directly comparable.

Two real stories. They come back in every part of this talk.

Case 1 | The support agent

One bad tool call silently corrupted hundreds of support tickets. No error. We found it in an audit.

Case 2 | The presales assistant

It looped forever on an input it had never seen and burned real compute before we caught it.

Where the **agentic stack** is being built.

1 | What is MAST-F?

1,642

annotated execution
traces

7

multi-agent frameworks

14

failure modes in 3
categories

$\kappa = 0.88$

human annotator
agreement

System design · 44.2%

Inter-agent misalignment · 32.3%

Task verification · 23.5%

THE DEFINITION

MAST-F = MAST's Failure taxonomy: a data-grounded catalogue of how multi-agent LLM systems fail.

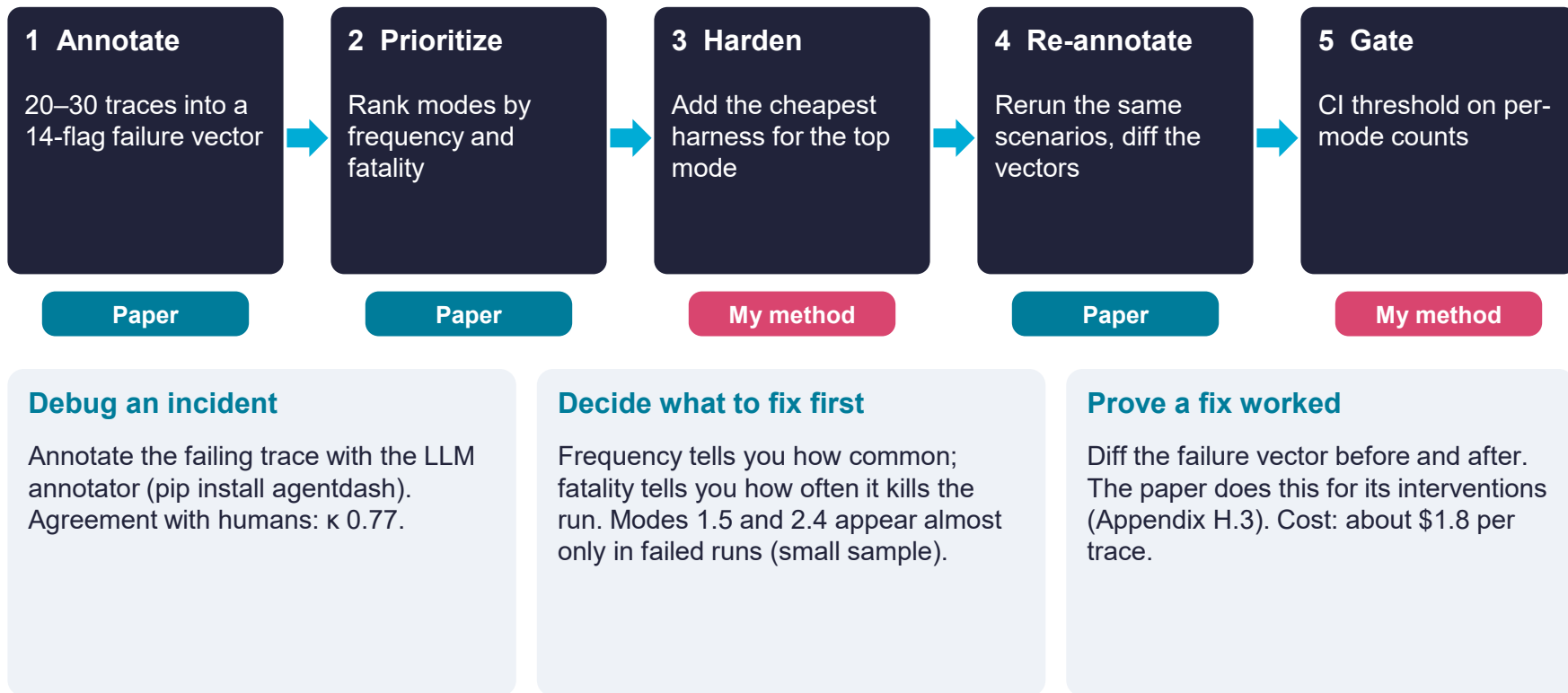
Objective: replace anecdotes with evidence on what fails, how often, and at which stage.

PROVENANCE

Cemri, Pan, Yang et al., UC Berkeley. NeurIPS 2025, Datasets and Benchmarks track. arXiv 2503.13657.

Systems: ChatDev, MetaGPT, HyperAgent, AppWorld, AG2, Magentic-One, OpenManus.

1 | MAST-F in practice: the loop



2 | What is Harness Engineering?

THE IDEA

Agent system = LLM core + harness

The LLM core is probabilistic. The harness is the deterministic runtime around it: what the agent sees, which tools it may call, how far it may loop, and who decides it is done.

Recent work frames the same idea as model + harness + environment (Zhong and Zhu, arXiv 2605.13357).

Why it matters: every handoff and tool call is a place to fail.

TETHER

Clear roles and handoffs
Pydantic v2

ANCHOR

Safe tool calls
MCP | the support agent

BELAY DEVICE

Limits on loops
LangGraph | the presales assistant

VERIFY GATE

Nothing is done until checked
Deterministic checks

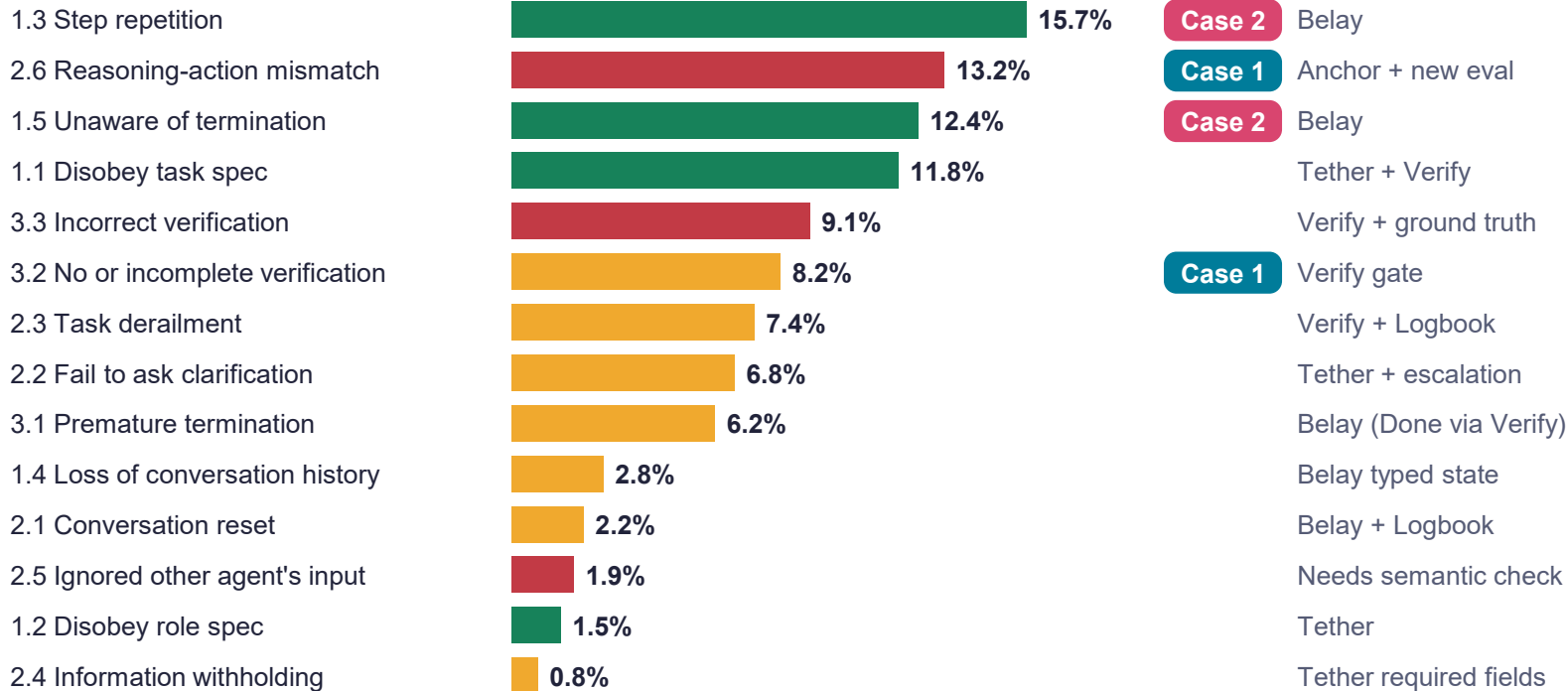
LOGBOOK

See what happened
Opik + OpenTelemetry

3 | Triage: which failure first, and who owns it

Strong 41% Partial 34% Weak 24%

Coverage is my assessment. Tags mark our two cases.



3 | Case 1: the support agent

WITHOUT THE GEAR

- The agent updates a ticket with status = null and priority = NaN.
- The Zendesk tool accepts it and says OK.
- The ticket is corrupted. No error anywhere.
- Hundreds of tickets later, an audit finds it.

WITH THE GEAR

- The tool boundary checks the call first: status cannot be null.
- It sends back a clear error; the agent uses a safe default.
- The ticket stays valid, and the trace shows what was blocked.
- If it keeps failing, a person is called in.

MCP tool contract (illustrative pattern)

```
update_ticket(ticket_id: str, status: Literal[open, pending, solved], priority: Literal[P0..P3]) # null or NaN rejected at the boundary
```

Closest MAST-F labels: 3.2 (nobody checked the write) and 2.6. This was agent-to-tool, not agent-to-agent.

Evidence: a clean-room replay of the incident, not a controlled study. Production figures cannot be shared.

3 | Case 2: the presales assistant

WITHOUT THE GEAR

- The reviewer says REVISE and the loop starts again, with no way out.
- The drafter writes the same proposal again and again.
- It keeps running until a person steps in.
- Real compute is burned before anyone notices.

WITH THE GEAR

- The guard compares each draft with the last one. Same draft: stop.
- It also stops after a maximum number of rounds.
- The run goes to a person for review.
- The trace shows which guard fired.

Belay guard (illustrative pattern)

```
if signature(draft_n) == signature(draft_n_minus_1) or iteration > MAX_ITER: route("human_review")
```

MAST-F kinds 1.3 (repeating steps) and 1.5 (no stopping rule): together 28.1% of failures.

Watching tool calls alone would miss this loop. Compare the draft itself.

4 | High-level architecture

BELAY DEVICE | typed LangGraph StateGraph

Case 2 stops here

Trigger



Intake



Agent A



Agent B



Verify gate



Done

max iterations

repeated-state signature

token + time budget

context compactor

Trigger: support ticket or RFP. Only Verify can reach Done. Escalation edge to human review.

TETHER Pydantic v2 role and handoff contracts at every node boundary, versioned; an invalid handoff returns a synthetic 422

ANCHOR | MCP client and gateway

Case 1 stops here

Allowed tools per role, request and response schema, structured error, permissions, sandboxed execution.

MCP: helpdesk

MCP: CRM pricing

Case 1 is blocked at the Anchor. **Case 2** is stopped by a Belay guard. Both show up in the Logbook.

Failure injection: the next slide replays both cases, harness off, then on.

LOGBOOK

Opik + OpenTelemetry. A span for every node, tool call and guard event, tagged mast.target_mode. Export traces below.

MAST-F LOOP | offline, CI

1 Annotate (agentdash)

2 Prioritize

3 Harden the runtime

4 Re-annotate

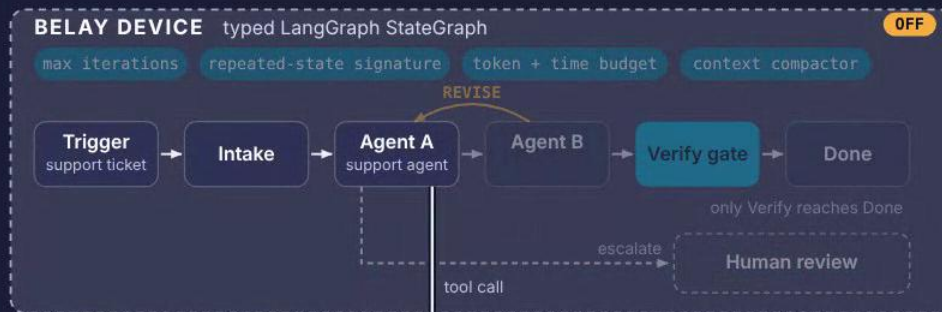
5 CI regression gate

Where the fall stops

Case 1 the support agent

Case 2 the presales assistant

☐ Harness off



TETHER

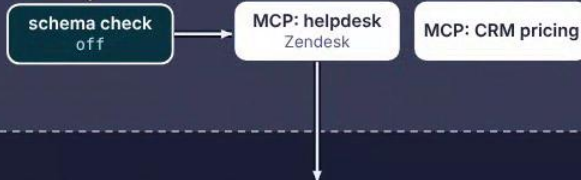
Pydantic v2 role and handoff contracts at every node boundary, versioned.

OFF

ANCHOR MCP client and gateway

allowed tools per role
request and response schema
structured error to the agent
permissions, sandboxed run

OFF



ZENDESK TICKETS illustrative

■ untouched ■ valid ■ corrupted

STANDARD LOGS

no harness

OFF

Now

6 steps

In progress

The result appears when the run ends.

MAST-F label

3.2 No or incomplete verification

2.6 Reasoning-action mismatch

Closest labels. This was agent-to-tool, not agent-to-agent.

MAST-F LOOP

offline, in CI - the author's method

OFF

1 Annotate (agentdash)

2 Prioritize

3 Harden the runtime

4 Re-annotate

5 CI regression gate

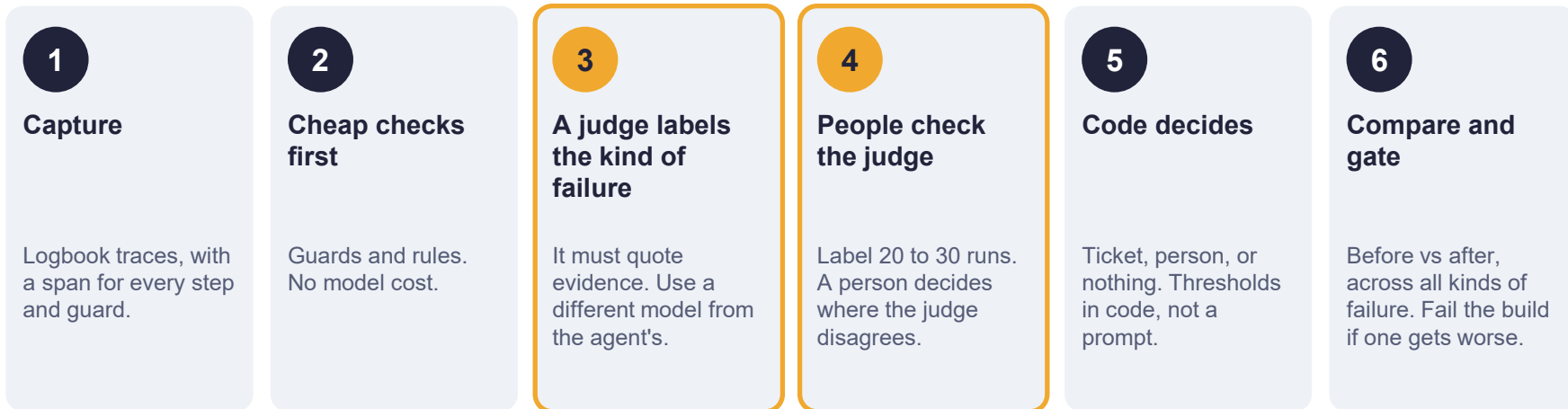
Dimmed: not on this case's path.

OFF: that gear is switched off.

Ticket, round and compute figures are illustrative.

4 | How the loop runs: a pipeline

The gear catches the failures you know. This finds the ones nothing catches.



Amber boxes: the two steps where a model can be wrong. That is why people check it.

The judge is a sensor, not an authority.

The judge pattern and its 0.77 agreement with humans come from the MAST paper (Cemri et al., arXiv 2503.13657). The pipeline design is my method.

5 | Challenges and mitigation approach

Challenge	In our two cases	Mitigation
Seeing failures at all	The support agent stayed silent: no guard fired, an audit found it.	Label a sample of 20–30 traces with the annotator, then hand-check some
A call can be valid and still wrong	A well-formed ticket update can still carry the wrong status.	Check the call against the plan; use real data as ground truth; ask a person for risky actions
Guards can misfire	A repeat check on the presales assistant could stop a good run.	Tune the limits; send to a person instead of failing hard
MCP guards the call, not the meaning	It stopped the bad write. It does not fix agents misreading each other.	Say that clearly; check what agents hand to each other
Cost grows with every agent and every check	Many agents use about 15 times the tokens of a chat. Checks cost tokens too.	Cheap checks first; new guards in shadow mode; track tokens per finished task
Our proof is a replay	Both cases are replays, and production figures cannot be shared.	Shadow-mode traces; redact customer data before any hosted judge

6 | What improves: our cases, public data

The support agent

Before: Found in an audit.

After: The bad write is blocked at the tool boundary.

The presales assistant

Before: Ran until we noticed.

After: Stops on a repeat and goes to a person.

My production figures cannot be shared. Public data below shows the same kind of gain.

MAST paper: ChatDev

task success, %

Baseline



Better role prompts



New topology + verify



LangChain Deep Agents

Terminal Bench 2.0, %, same model

Before



After (harness only)



+13.7 pts: self-check, loop detection, context, traces.

MAST paper: AG2

MathChat accuracy, %

	GPT-4	GPT-4o
Baseline	84.75	84.25
Prompt	89.75	89.00
Topology	85.50	88.83

GPT-4 topology gain: $p = 0.4$, not significant.

The MAST-F loop is how you measure your own before and after.

Route plan: what to adopt first

1

Anchor

Stops the support agent's bad write at the tool boundary

2

Tether

Clear roles and handoffs. Cheap, and it catches drift early

3

Belay

Stops the presales loop: a repeat check and a round limit

4

Logbook

Shows what 1 to 3 really catch. Never skip it

5

Re-annotate

MAST-F: compare before and after, keep or revert

Before step 1: annotate 20–30 of your own traces. If your data points to a different top mode, follow your data.

Preconditions: you can capture full traces and type your tool boundaries. Closed systems that expose no traces cannot be annotated.

Key takeaways

1

Structure, not just the model

Most failures in MAST come from how agents are wired: handoffs, tools, stopping rules.

2

Label your own traces first

20 to 30 of them. Let your data pick the first fix.

3

Match the gear to the failure

A silent bad write needs an Anchor. A runaway loop needs a Belay.

4

Measure before and after

Use the MAST-F loop, and count tokens per finished task.

Summit and questions

MAST tells you where climbers fall.

Harness Engineering is the gear that catches you.

Questions? Jay Mehta, Cygnet Infotech

Talk page and QR

Scan for the talk page, or to leave feedback.



Talk page



Feedback

Sources

Cemri et al., arXiv 2503.13657 (MAST)
Zhong and Zhu, arXiv 2605.13357
LangChain, Improving Deep Agents with harness engineering, 17 Feb 2026

Backup: five components, four layers

Component (research design)	Layer	MAST-F modes	Open-source tooling
C1 Context governance	Added control	1.4 (2.8%)	Custom compactor
C2 State machine, circuit breakers	Belay Device	1.3, 1.5 (28.1%)	LangGraph
C3 Inter-agent contract gateway	Tether	2.4, 2.6	Pydantic v2
C4 Tool execution and isolation	Anchor (MCP)	Tool-contract failures	MCP SDK, container sandbox
C5 Verification and termination	Verify gate	3.1, 3.2, 3.3 (23.5%)	Deterministic assertions
Observability and MAST-F triage	Logbook + MAST-F	All (detection)	Opik + OpenTelemetry

Shares from Figure 1. C2 = 15.7 + 12.4; C5 = 6.2 + 8.2 + 9.1. Add tool-argument schema checks to C4: that is the Case 1 gap.

Backup: the pipeline in detail

Stage	What it does	Guards against
1 Capture and normalize	A span for every node, tool call and guard event. One readable transcript; redact customer data; split long traces into overlapping chunks.	Failures you cannot see; data leaving
2 Cheap checks	Budget over, repeated draft, guard fired, schema rejection. Choose what the judge reads, plus a sample of clean runs.	Wasted judge cost; silent misses
3 Judge	Labels the 14 kinds. Per kind: yes or no, confidence, quoted evidence, or abstain. Different model from the agent's.	Unsupported flags; self-scoring
4 Check the judge	Known-good and known-bad traces every run. Compare with a human-labelled set, kind by kind.	Judge drift; false alarms
5 Code decides	Thresholds in code: ticket and person, person review, backlog, or nothing.	A model deciding policy
6 Compare and gate	Failure vector per run, change from baseline, every kind shown. Fail the build on a regression.	Fixing one kind, breaking another