LLMDAY NYC · MICROSOFT EXPERIENCE CENTER · OCT 29, 2026

What did your agent actually do?

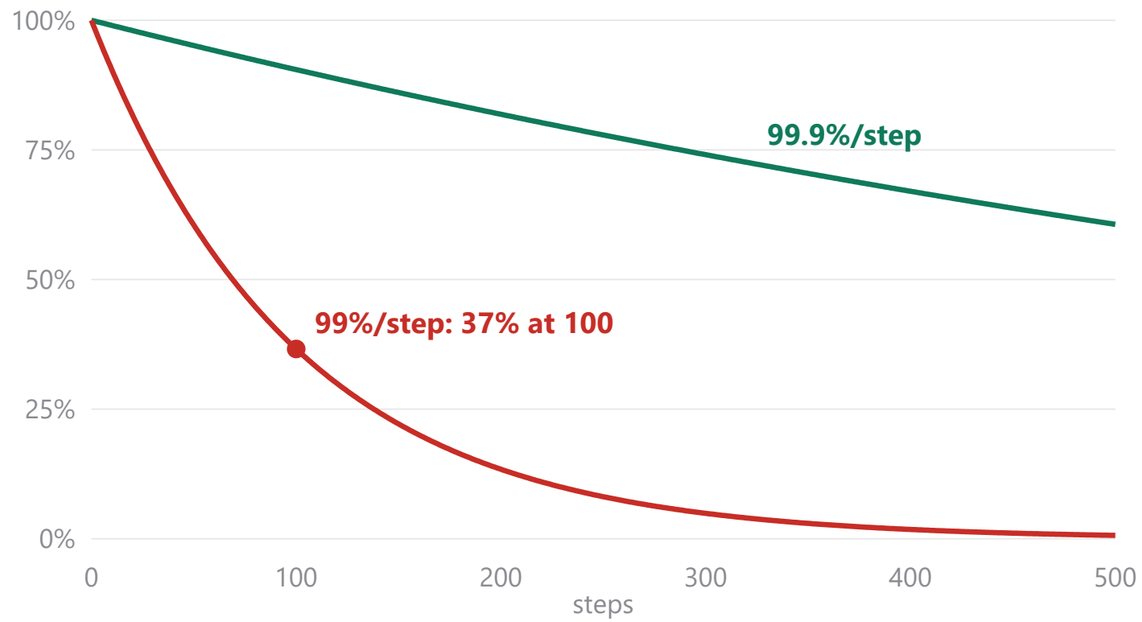
Maureese Williams · harnesses for long, risky work

```
$ ledger run "fix the failing test" --backend ollama --model qwen3.8:27b
+ goal reached
  plan        RunSuite → AuthorPatch:a → ApplyPatch:a → RunSuite
  run         4 steps - 0 LLM planning calls - profile attended
  verify      + 0 incidents
$ ledger verify-chain episode.jsonl
+ chain OK -- every hash links, nothing has been altered
```

Real output, Oct 21 run: local qwen3.8:27b, one GPU, 1 min 58 s.

THE PROBLEM

Long, risky work breaks agents



99% per step is 37% at 100 steps and 0.004% at 1,000.

Confidence is not evidence

`"tests pass"` `"file written"`

`"safe to deploy"` `"done"`

Most failures are the agent believing its own claim about the world.

Risk-averse work asks

What exactly did it do? Can you stop it **before** the bad step? Can you prove it afterwards?

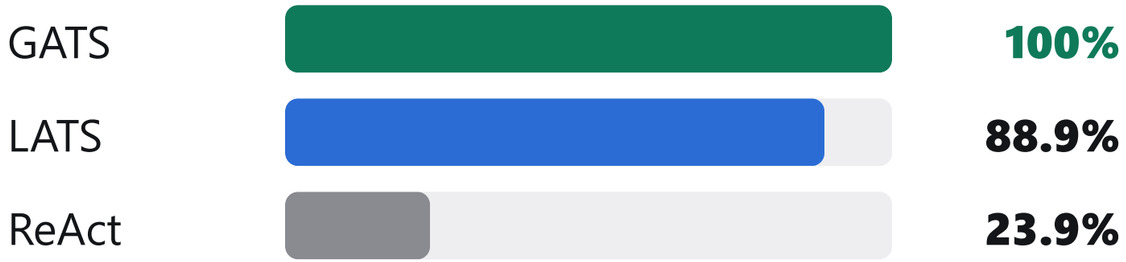
GATS · ARXIV:2607.08894 · WITH DYMITR NOWICKI

The model is the last resort, not the decider

UCB1 tree search over a **persistent state-transition graph** picks every action. Each transition goes to the cheapest layer that can answer it:

L1 symbolic contracts	`exact · free`
L2 statistics from its own runs	`learned`
L3 LLM prediction, cached	`novel only`

12-category stress test, 120 tasks (reproduces exactly)



0 LLM calls during planning · **0%** variance across seeds · **100%** on live Llama 3.2, where LATS fell below 1%

FROM PAPER TO PRODUCT · THE HONEST LIMIT

Search doesn't scale. So search less.

Over raw actions, tree search is impractical past about 20 actions or depth 10. So the planner searches a few coarse moves and pushes the open-ended work down into skills.

Plan level: searched

`RunSuite · AuthorPatch · ApplyPatch · Commit · CreatePR`

Few, coarse, contracts known. Every choice is scored and logged.

Skill level: the LLM inside

Writing the diff. Its output is a proposal: preflight, `git apply --check`, then the suite decides.

LEDGER · AUDIT-FIRST CODING AGENT ON GATS

Neural proposes, symbolic disposes

GATS plans
every action, 0 model calls

→

Skill
the LLM writes a patch

→

Sensors
decide what is true

→

Journal
hash-chained, replayable

State is what sensors say

`tests.pass:unit`, `repo.clean`, `recoverable`. The model never sets them.

Lies become incidents

Claimed effects minus observed effects is an `INCIDENT`; the planner routes around the falsified edge.

Gates, not hopes

Risk classes gate execution. A denial is an observation. Escalation is named: `no_path`, `infeasible`, `budget_exhausted`.

THE LEDGER

The answer to the title, from the record

```
$ ledger why episode.jsonl 1
step 1 @ 87b650f27839: chose AuthorPatch:a
  predicted → 378e3586acaf (layer L1, p=0.5) - 10 search iterations - 0 LLM planning calls
  arm            N  Q      UCB
  AuthorPatch:a  5  1.167  2.126  ⟸ chosen
  AuthorPatch:b  5  1.167  2.126

$ ledger replay episode.jsonl
+ replay OK -- 4 steps reproduced bit-exact

$ ledger verify-chain tampered.jsonl      # one field edited in the END entry
x chain BROKEN: entry 17: content hash mismatch
```

EVIDENCE, AND WHAT BROKE

Grade problem solving, not pass rate

Hard and impossible tasks, LEDGER alone on local qwen3.8:27b. Rubric: recognition, integrity, delivery, efficiency.

TASK	OUTCOME	SCORE
squeeze (pigeonhole)	proof, `infeasible`	8/10
crc (contradictory tests)	`infeasible`, values right	10/10*
primes, rk45	no patch, no gaming	4/10

* Not first contact. Scores are operator judgment; pass counts and test-edit checks are mechanical.

My harness got reward-hacked

The journal said **success**. The author had edited the tests; the hidden suite failed. It checked the bar, not whether the bar moved.

Fix: test integrity became a sensor. Not a better model.

WHERE HARNESSES GO

Models are parts. The harness is the product.

Swap the model

Claude for a local 27B model: state, verification and replay don't move. Guarantees live in the harness.

It's the RL environment

Sensors are the verifier, worktrees the rollouts, the journal the trajectories. Per-action postconditions solve credit assignment.

Self-improvement, gated

`ledger improve` files its own bugs as failing tests and fixes them, local model only. The gate caught 2 bad changes in 5.

Safe only while the verifier sits outside the thing being improved. Weights promoted only if plan variance stays at zero.

MONDAY MORNING · FOR ANY CODING AGENT

Context is the budget. A check is the finish line.

1 · Give every task a runnable check

Failing test first, scoped command, "end with the test output". Unskippable: `make check` in a Stop hook, exit 2.

2 · Treat context as the budget

One task per session, `/clear` between tasks, subagents for investigation, `HANDOFF.md` when bloated.

3 · Fresh-context review before "done"

`/code-review`, a second session, or Codex `/review`. Correctness gaps only.

4 · Make repeated corrections durable

Judgment rule → `AGENTS.md` · workflow → skill · mechanical rule → hook.

Confidence is not evidence.

Build the harness that asks for evidence. Where this goes:

1. Evals report variance and replayability, not just success rate.
2. Regulated work requires deterministic replay of agent actions.
3. The LLM moves down the stack; verifiers become the moat.
4. "I can't do this as specified" becomes a scored capability.

Hugging Face
huggingface.co/maureesewilliams

arXiv paper
arxiv.org/abs/2607.08894

Blog
maureesewilliams.io

ADDENDUM · LIVE RESEARCH

GATS since the preprint

Re-run end to end, tested on a live model, methods tightened. Here's where the numbers stand today.

A1 · 100-TASK PLANNING SUITE, CURRENT NUMBERS

On a real model, the gap widens

	REPRODUCIBLE PIPELINE	LIVE LLAMA 3.2 (3 SEEDS)
GATS (b=10)	100%	100%
GATS (b=5)	75%	
LATS (b=10)	87.4%	0.3-0.7%
ReAct	51.6%	36.3%
LLM calls (ReAct / LATS b=5 / b=10)	16.2 / 21.4 / 46.2	

Deterministic, about 2 minutes, no GPU, no keys. These supersede preprint v1's Table 1. GATS doesn't move with the model because it never calls the model while planning.

A2 · WHAT'S BEEN REFINED

Tighter methods, stronger evidence

- **Reproducible by anyone:** every number now comes from a seeded, deterministic pipeline in the repo.
- **Live-model baselines:** v1's baselines ran on a simulated backend; on Llama 3.2, GATS holds 100% while LATS falls below 1%.
- **Methods detail tightened:** seed counts, call accounting, node counts, scaling language, and the λ setting are now stated precisely.
- **Still open:** one unseeded v1 LATS run (92%) lands at 87-90% today.

Holds

- Stress test, exactly: 100 / 88.9 / 23.9
- 0 LLM calls during planning (50 bootstrap)
- 0 variance across seeds
- Plan quality independent of model quality

Locked PlanBench (v2.1)

The first locked run surfaced three evaluator flaws. With them fixed: weighted best-first **78/90** vs BFS **32/90**, **+51.1 pts** (95% CI 43.3-58.9).

Graph memory cuts model queries **79.9%** with identical plans.