VAAS-X SDK · ROBOTICS & AUTONOMY

Memory Inside a Live Control Loop, and an Honest Limit We Found

A simulated drone wired into the vaas-x SDK's ingest/query/outcome loop in real time -- plus a controlled ablation that found a genuine, reproducible limitation in how memory competed for the decision, and why.

2/2	1/232	400	0
ABLATION RUNS, BYTE-IDENTICAL	STEPS MEMORY WON THE DECISION	PTS: SINGLE COLLISION COST IN SCORING	GPU REQUIRED

VAAS-X Ltd (SC873624) · August 2026 · vaasx.com

A different kind of test: a closed control loop, not a static dataset

Every other whitepaper validates retrieval against a fixed dataset: ingest, then query, then check the answer. This one asks a harder question of the SDK: can it sit *inside* a real-time decision loop, where every query result immediately becomes an action that changes the world, which changes the next query? We built a simulated drone — 2D position and altitude, raycast obstacle sensing, wind and battery drain, walls it can actually crash into — and wired vaas-x into its control loop directly: canonicalize state to text, query memory for similar past situations, score the retrieved action alongside a local controller via a short rollout simulation, apply a deterministic safety veto that memory can never override, execute, and write the outcome back.

What Bootstrap actually does here

```
from vaasx import Bootstrap

brain = Bootstrap(api_key="YOUR_API_KEY", device_id="drone_01")

hits = brain.query("Drone navigation state. goal_dist=6.0m yaw_err=0.2rad
hazard=clear ...", k=7)

brain.ingest([{
    "state": {"text": "Drone navigation state. goal_dist=6.0m ..."},
    "action": {"text": "Control: thrust=0.72 yaw_rate=0.31 (left) climb=-0.20
(down)"},
    "outcome": {"text": "Outcome: success no_collision dist_to_goal=1.9
energy=0.013 arrived", "success": True},
}])

brain.outcome(hits[0]["id"], success=True, delta=0.12)
```

Same three calls as every other vertical — ingest, query, outcome — applied to a real-time control problem instead of a dataset.

The architecture: memory suggests, deterministic rules decide

Retrieved actions are one candidate among several — alongside a local heuristic controller, a handful of purpose-built maneuvers (settle near the goal, brake, escape a close obstacle), and small perturbations of the heuristic action. Every candidate is scored by simulating it forward a few steps and penalizing collisions, close clearance, and wasted energy. A hard-coded veto function rejects any candidate — retrieved or not — that violates a safety rule (thrust too high with an obstacle close

ahead, turning too hard with GPS degraded). This separation is deliberate: probabilistic retrieval proposes, deterministic rules and simulated rollout decide. It's the same principle behind every guardrail pattern in the other whitepapers, just running inside a live loop instead of a one-shot query.

An honest result: memory rarely wins the decision, and we found out why

We ran a controlled ablation: the identical 10-episode run, twice, on the same wall layout and the same random draws — once with memory allowed to compete for the decision, once with it excluded entirely (everything else in the candidate pool identical). First attempt: both arms produced byte-identical trajectories. The cause turned out to be in how we'd seeded memory — entirely from the same local heuristic controller it was competing against, so retrieval had nothing genuinely different to offer. We fixed that (24% of seed episodes now come from a distinct exploration policy, not the heuristic) and reran.

> **RESULT, REPORTED HONESTLY**
>
> Both arms still finished at **4 wins out of 10** — and still episode-for-episode identical. Across 20 full episodes (232 total control steps), the memory candidate won the internal decision exactly **once**. The reason: the scoring bonus intended to favor a recalled action is a small, fixed number, while the rollout cost function it competes against has a much larger dynamic range — a single collision alone costs 400 points in that scoring. A recalled action transplanted onto a new state essentially never beats a controller purpose-built for that exact state by less than the bonus, so the discount rarely if ever changes the outcome. This is a faithful port of the original prototype's own scoring formula, not something introduced in porting it — which means it's a genuine, previously unmeasured property of this specific design, not a flaw in retrieval itself. We can't even say whether what memory recalled was good; it essentially never got the chance to be chosen.

The generalizable lesson: in a "propose from memory, then score by simulation" pattern, the bonus that's supposed to let memory override a local controller has to be calibrated against the actual scale of the scoring function it's competing inside, or memory will structurally never participate no matter how good retrieval is. That's a real, useful thing to know before building this pattern into a production autonomy stack — and it's the kind of finding that only shows up when you run a real ablation instead of taking a demo's summary numbers at face value.

Scope, honestly

What this validates: the ingest/query/outcome path runs unmodified inside a real-time control loop against a resolver over the network, retrieval and safety-veto logic compose cleanly, and a controlled, reproducible ablation methodology can isolate exactly what a memory-augmented decision system is and isn't doing. What it does **not** show: that retrieval-augmented control

outperforms a well-tuned local controller in this environment — on this evidence, it didn't get the chance to. This is a simulated drone, not flight-tested hardware, and the scoring-calibration issue found here is specific to this rollout-scoring design, not a general claim about memory-augmented control.

Deployment model

Identical to every other vertical: a single API key against the hosted resolver, no local GPU or vector database. The control loop shown here ran unmodified against the same public endpoints used throughout this SDK — nothing control-loop-specific was added to make it work.

Get started

Free tier: instant API key, no credit card — `pip install vaas-x`, connect a source, query your first episode in under ten minutes.

Full guide: vaasx.com/docs/getting-started.html | Enterprise: hello@vaasx.com