ANI-Gamut: Benchmarking Agent Reliability across the Gamut of Agent–Network Interface Abstractions

Lorenzo Bracciale[1], Pierpaolo Loreti[1], Andrea Mayer[1], **Stefano Salsano**[1], Wim Henderickx[2]

[1] University of Rome Tor Vergata / CNIT, Italy · [2] Nokia, Belgium



Presenter: Stefano Salsano

- Professor of Telecommunications, University of Rome Tor Vergata; CNIT research unit lead
- Research: network programmability, SRv6, SDN/NFV, Linux networking; IETF contributor
- Current focus: agentic AI for network management and reproducible benchmarks
- Project page: **netgroup.github.io/ani-gamut** · stefano.salsano@uniroma2.it

22nd International Conference on Network and Service Management (CNSM 2026) · 27 October 2026, Alcalá de Henares, Spain

Funded by the European Union NextGenerationEU | **PRIN** Ministero dell'Università e della Ricerca | **Italia Domani** National Recovery and Resilience Plan | **PRIN 2022 NEWTON**

Key messages

1 **LLM agents operate networks through some interface.**
Nobody has measured which interface level works best.

2 **ANI-Gamut is an open-source playground for agentic network management;**
in this paper we change only the interface level and keep task, model and network fixed.

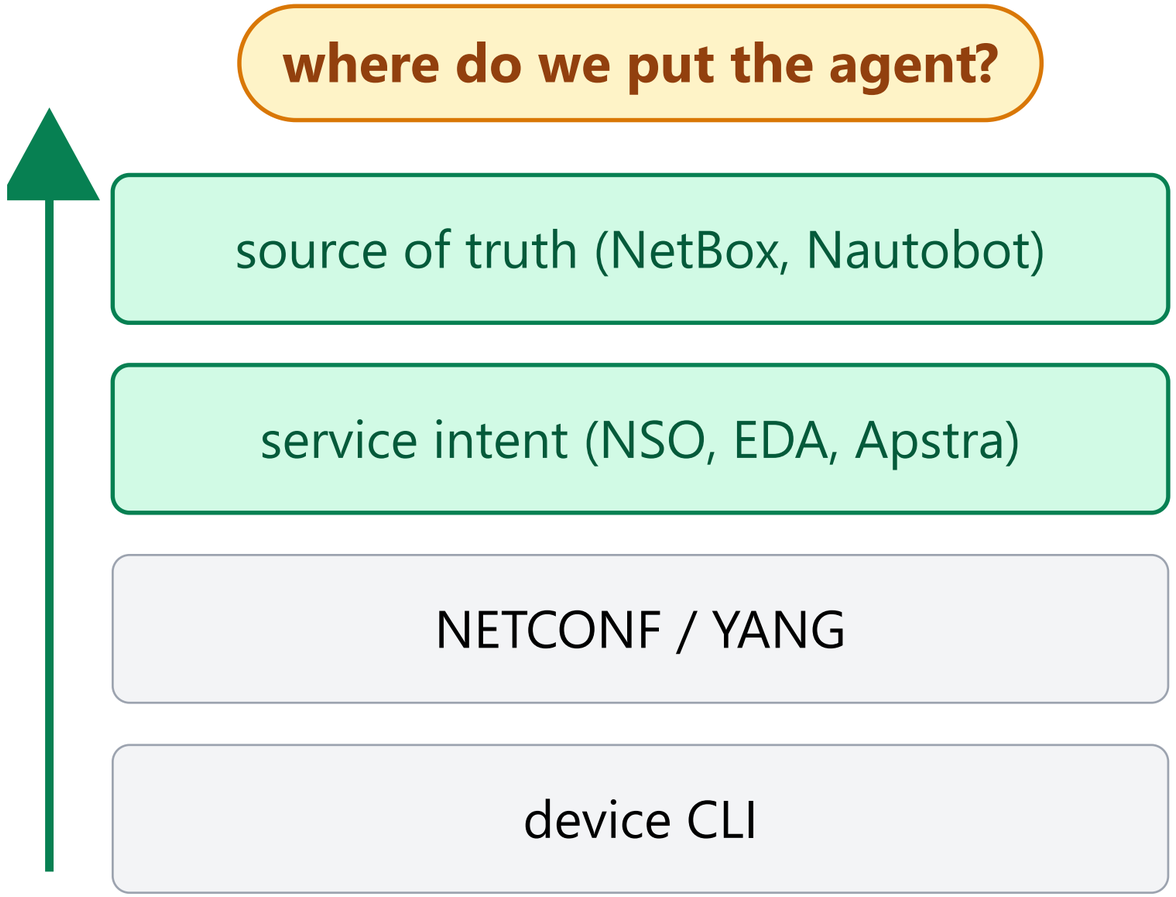
3 **In our pilot, a higher-level interface succeeds where the raw shell fails, at a tenth of the cost;**
no collateral damage at any level.

1 · THE QUESTION

At which level should the agent act?

- Attention goes to the **agent**; its **contract** with the network stays implicit
- Coding agents learned it: **the interface governs task success**
- **ANI: Agent–Network Interface** the contract the agent acts through
- Most existing work (e.g. IETF draft) fixes the ANI at **one level**

Operators are climbing this ladder today. Which rung is safe to hand to an agent is being decided in practice.

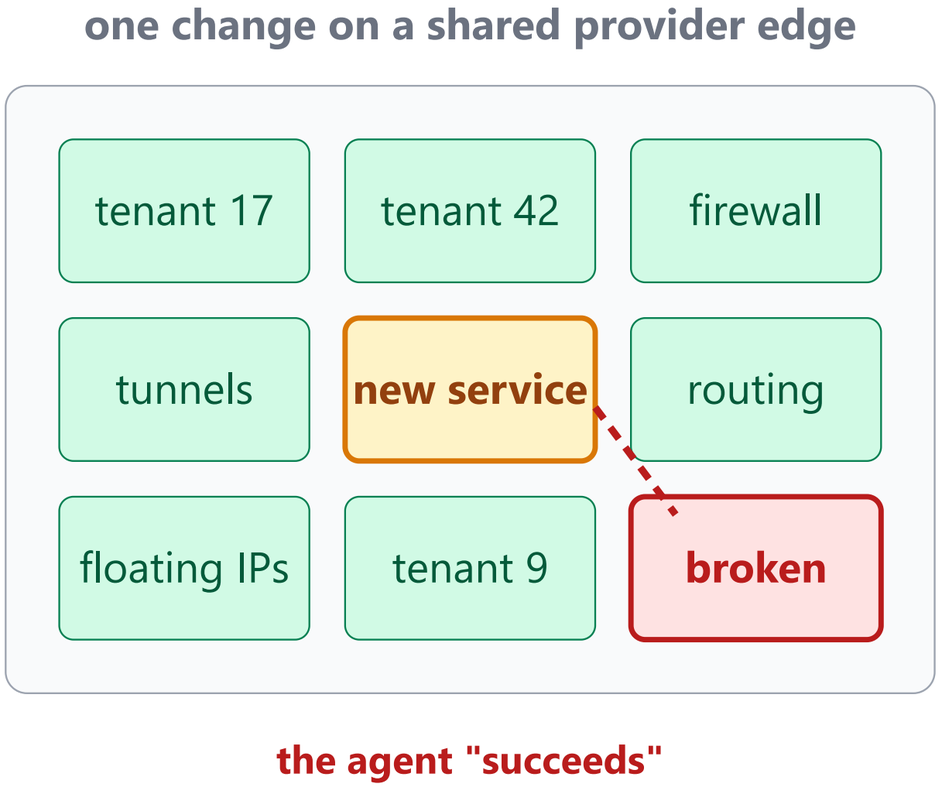


The failure that matters: collateral damage

- Task done, **unrelated service broken** = an outage
- Two legs: reach the goal **and** touch nothing else
- Carrier grade: **five nines**
- Only visible on a ***brownfield***:

  a network already full of other services

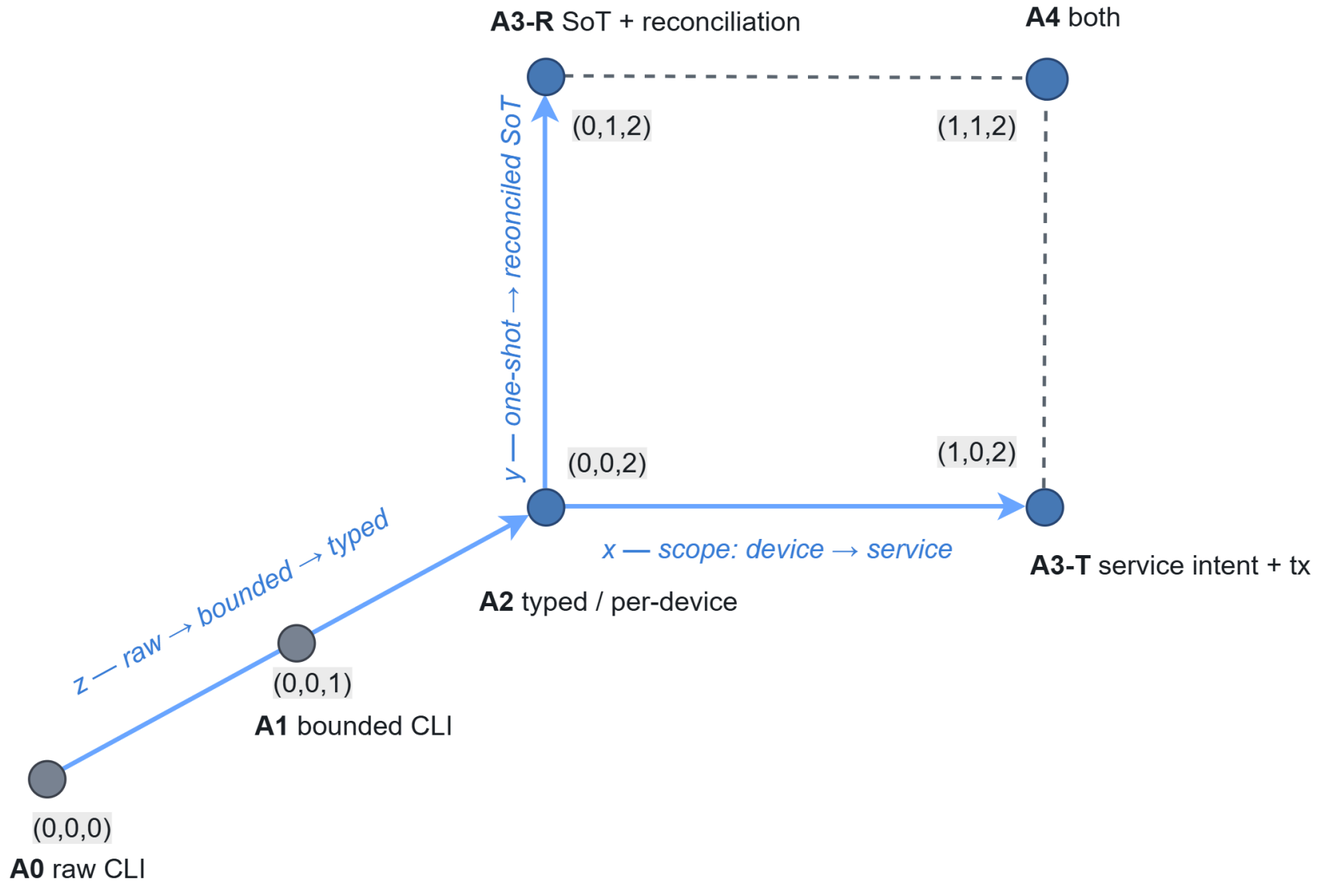
Toy playgrounds hide the interactions we need to stress.



The spectrum: A0 to A4

- **A0** raw shell · **A1** bounded CLI
- **A2** NETCONF/YANG, device commit
- **A3-T** service intent + transaction
- **A3-R** reconciled source of truth
- **A4** both: commercial controllers like Cisco NSO, Nokia EDA, Juniper Apstra

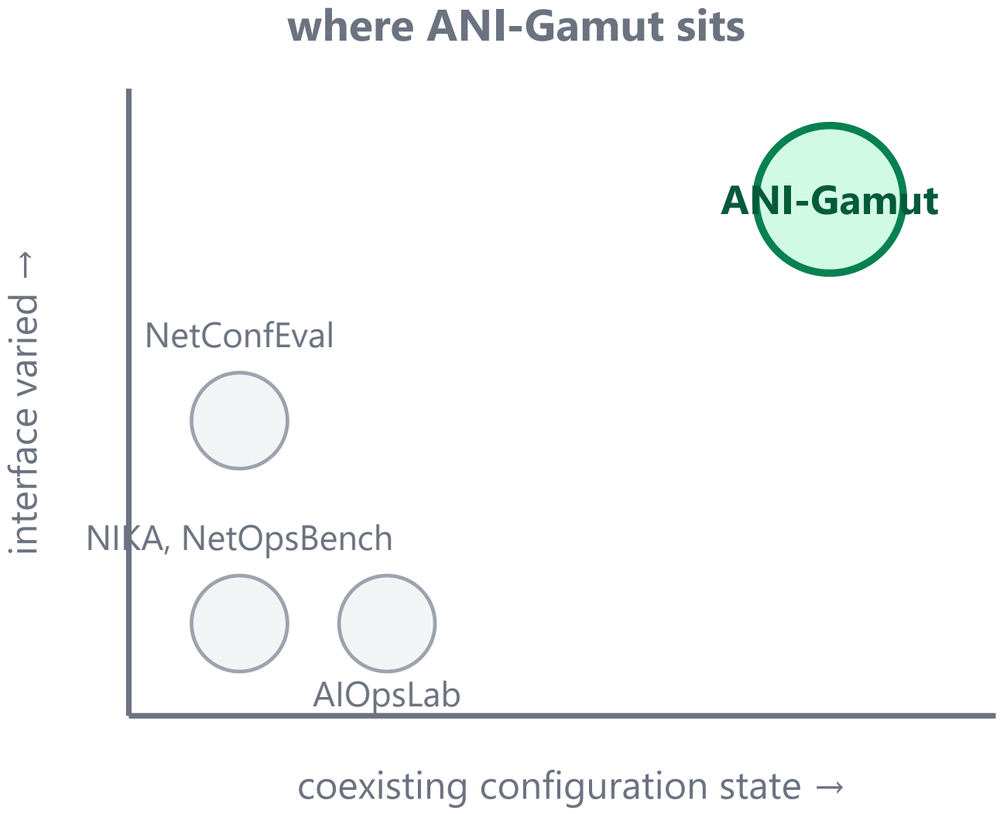
A ladder below, a plane above. A3-T and A3-R are two different bets on safety.



Nobody varies the interface

- Existing benchmarks: **one fixed interface**, mostly troubleshooting
- NetConfEval: levels compared, but as **static** translation
- OperAID: changing what the agent **sees** moves success 11% → 61%
- **ALL** populate **little coexisting state**

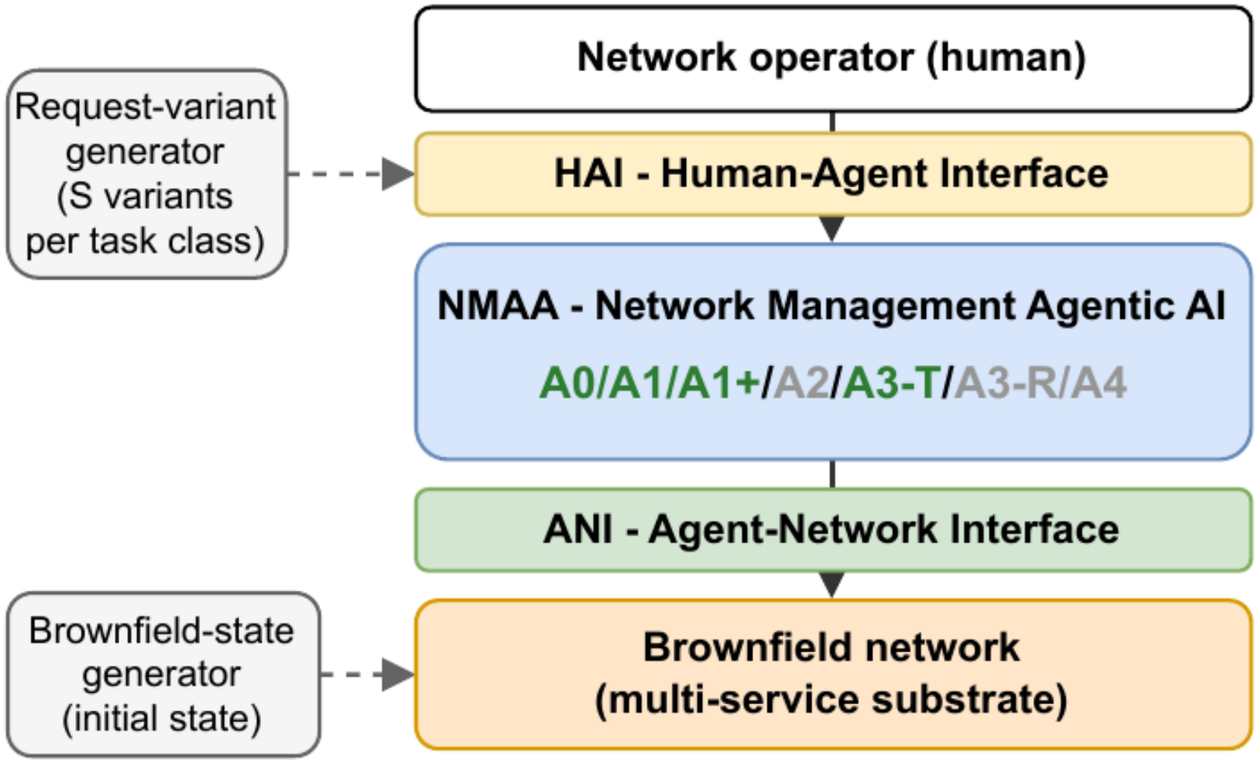
Production networks carry 10^5–10^6 config items. That density, not node count, is where damage happens.



One variable: the interface level

- **HAI**: the operator's request, fixed
- **The agent**: one model, fixed
- **ANI**: the level we vary
- Results checked from outside, same at every level

Every level gets the same agent budget. Same model, prompt, step and token caps, retries. The only thing that changes between two runs is the action interface.



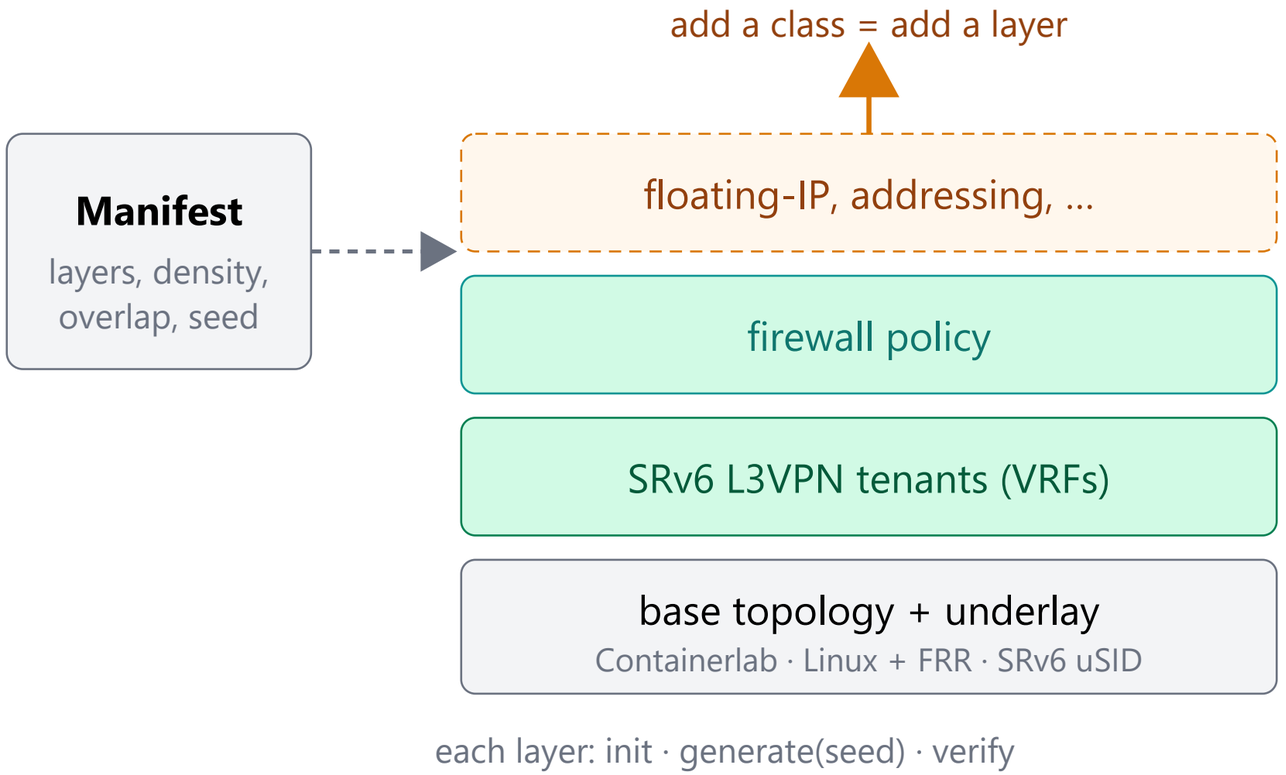
Same task, four implemented interfaces

Level	What the agent sees	Safety mechanism
A0	raw shell, every guardrail removed	none
A1	free-text CLI, bounded	allowlist + snapshot / rollback
A1+	A1 + hand-picked blocklist	ad-hoc prohibitions (control)
A3-T	typed **service** intent: `provision_service`	transaction + validation + rollback

A2, A3-R conceptual; A4 future work. Built from our configuration harness (A1) and the MAIN 2026 SRv6 playground (A3-T).

A brownfield built like a Docker image

- Topology base + **service-class layers**
- Manifest in YAML: a brownfield Dockerfile
- Each layer: **init · generate(seed) · verify**
- Each layer also writes the **checks** for its own services



~25 nodes, ~10 service classes, 10^3–10^4 items each. Density comes from configuration state inside the nodes, not from adding nodes.

Scoring: success vs. failure, safety vs. collateral damage

- Success = the requested service works
- Damage = a service that worked before no longer does
- Per run: damaged or not
- **Safety metric**: the fraction of damage-free runs (good = 100%)
- In the pilot we produced **failures**, never **damage**



3 · PILOT RESULTS

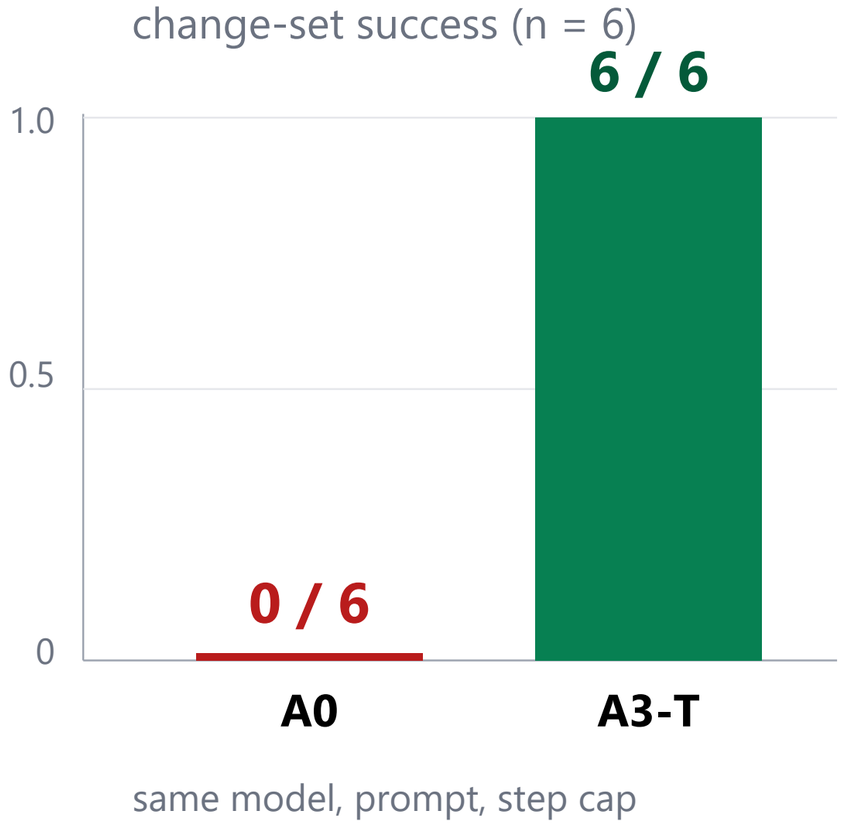
Task success follows the level PILOT

Change-set on a live 25-site tenant, n = 6 seeds:

task: add 2 sites, move 2, remove 1

Level	Success
A0 raw shell	0 / 6
A3-T typed intent	6 / 6

A0 gets ~85% of the sites connected on every seed, but never completes the task.

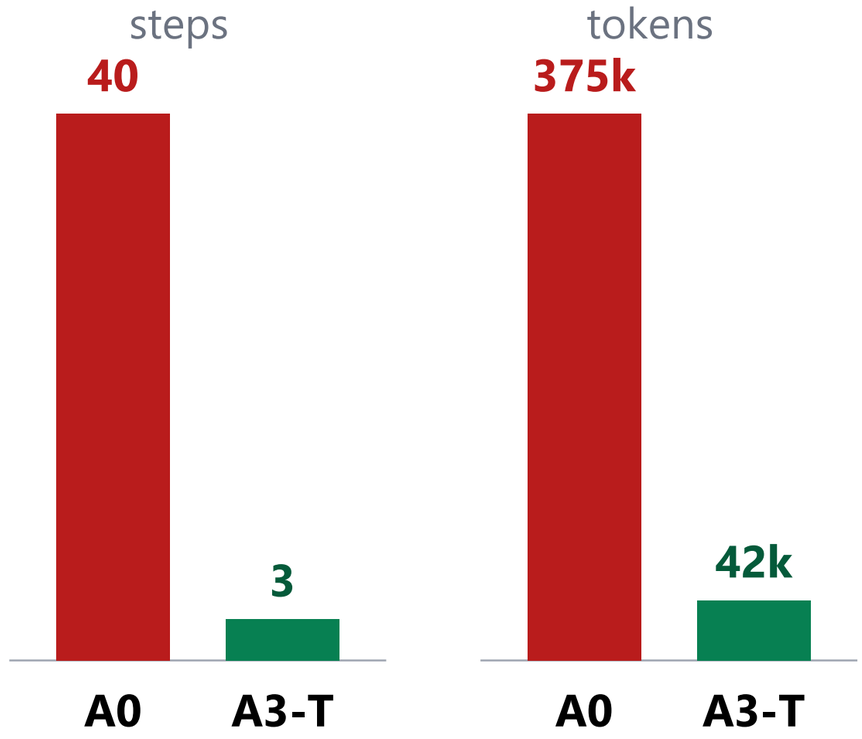


same model, prompt, step cap

Cost follows the level PILOT

- A0: **40 steps** (the maximum), **~375k tokens**
- A3-T: **3 steps**, **~42k tokens**
- Roughly **13×** steps, **9×** tokens
- Part of it is **budget exhaustion**: A0 fails *at* the cap

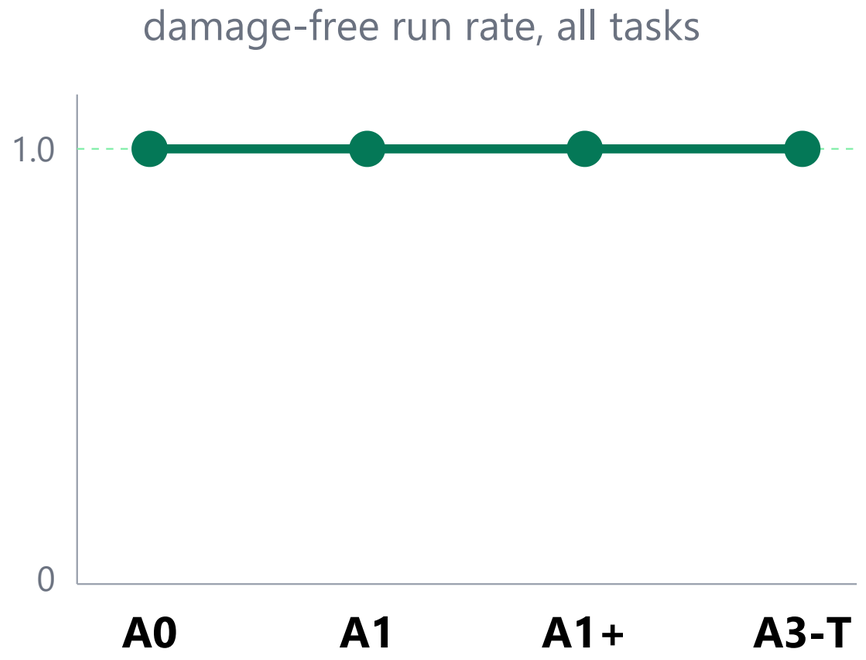
Why it is cheaper: the work moved from the agent to the automation layer. At A3-T the agent makes one `provision_service` call; the command sequence, validation and rollback live in the automation layer, written once. At A0 the agent rebuilds them at every run.



change-set task, matched budget

Collateral safety does *not* follow the level PILOT

- Damage-free runs: 100% at every level, on every task
- Agents **fail safe**: they give up,
  they don't break neighbours
- Tenants live in separate **VRFs**: nothing can leak



an honest null, with a cause: VRF isolation

In this testbed tenants are isolated by VRFs, so there is no shared state an agent could break. Ongoing follow-up work: a shared firewall ruleset across tenants, where a careless agent can hurt its neighbours (stay tuned!).

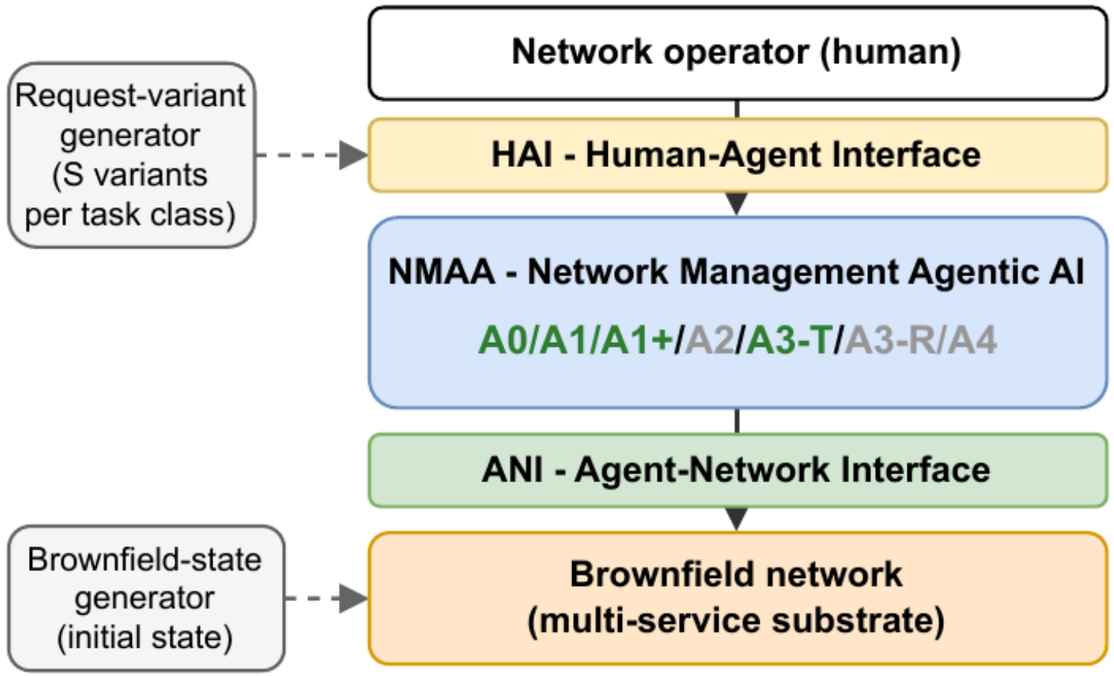
Limits of this pilot

- **Few runs**: 20 / 6 / 4 per level. “No damage seen” is not “no damage”
- **Few segments, not a curve**: only A0, A1, A1+ and A3-T were run;

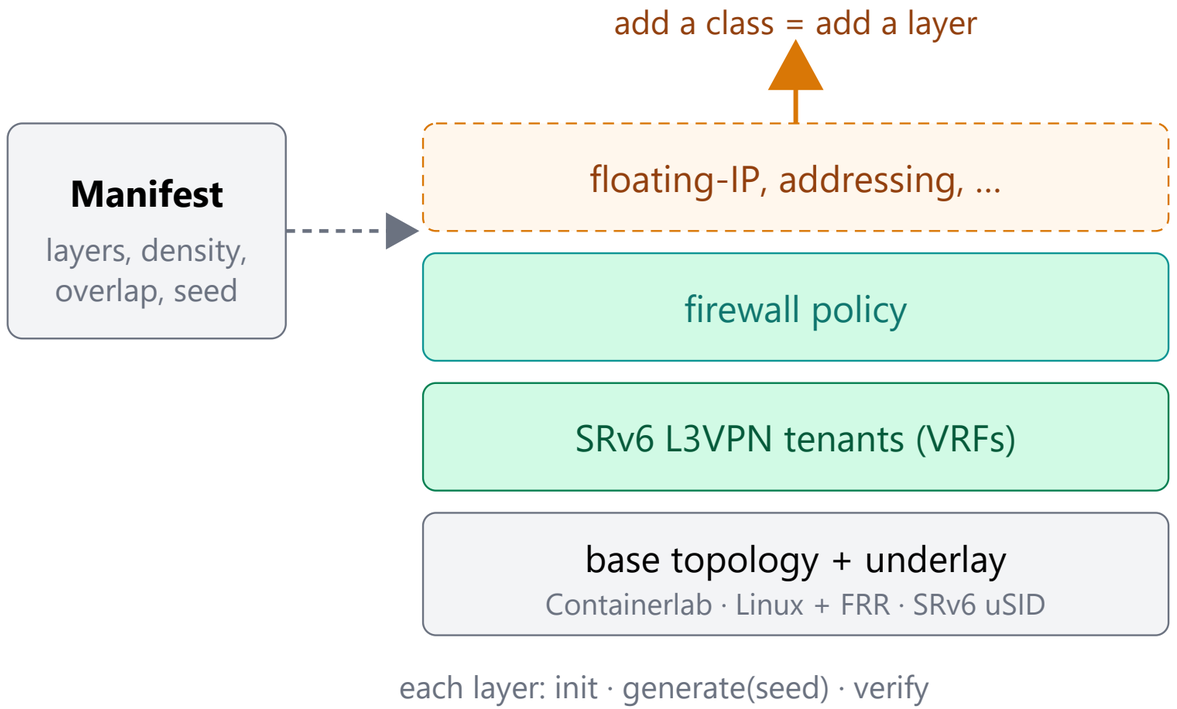
  A2, A3-R, A4 were not
- **Emulated network**: trust the gap between levels, not the absolute numbers

The gap between the ends is large and consistent; filling the middle is the next step.

Released as open source: build on it



Harness and per-level adapters: add a level, a model, a task class



Brownfield generator: add a service class, change density, replay a seed

github.com/netgroup/ani-gamut · playground, pre-registration and extended version at **netgroup.github.io/ani-gamut**

Take-home

1 **Treat the agent–network interface as a first-class experimental variable.**

2 **Task success and cost track it: 0/6 → 6/6 at a tenth of the cost.**

3 **Collateral safety was set by the network isolation, not by the agent.**

Next: shared state, A2 and A3-R implemented, density and fault sweeps.

4 **We need realistic testbeds, with complexity comparable to production networks.**

Thank you. Questions?

Playground, pre-registration, extended version: **netgroup.github.io/ani-gamut**

BACKUP

Which levels were run in which experiment PILOT

Experiment	A0	A1	A1+	A3-T
Benign onboarding, n = 20	0.20	0.35	0.55	not run
Change-set, live tenant, n = 6	0 / 6	not run	not run	6 / 6
Onboarding under faults, λ = 0.3, n = 4	0 / 4	2 / 4	1 / 4	4 / 4
Damage-free rate, all of the above	1.00	1.00	1.00	1.00

The headline contrast is A0 vs A3-T because the change-set task was run only at the two ends. A1 and A1+ appear in the benign and fault experiments; A2, A3-R and A4 were not implemented. Shared seeds at each level → paired McNemar test (change-set: p = 0.031 on six discordant pairs).

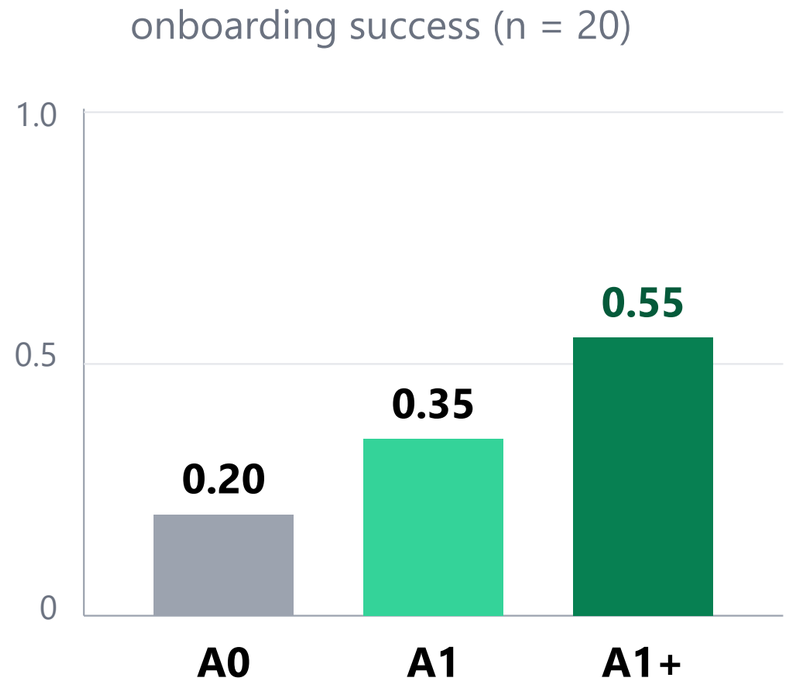
BACKUP

Benign onboarding, no faults PILOT

Level	Success	Wilson 95%
A0	4/20 · 0.20	[0.08, 0.42]
A1	7/20 · 0.35	[0.18, 0.57]
A1+	11/20 · 0.55	[0.34, 0.74]

- McNemar A0 vs A1+: p = 0.092 (trend); A1+ vs A1: p = 0.289
- Tokens +62% from A0 to A1; damage-free 1.00

Benign tasks show the trend, not the separation.



BACKUP · INJECTED FAULTS 1/3

Why we inject faults

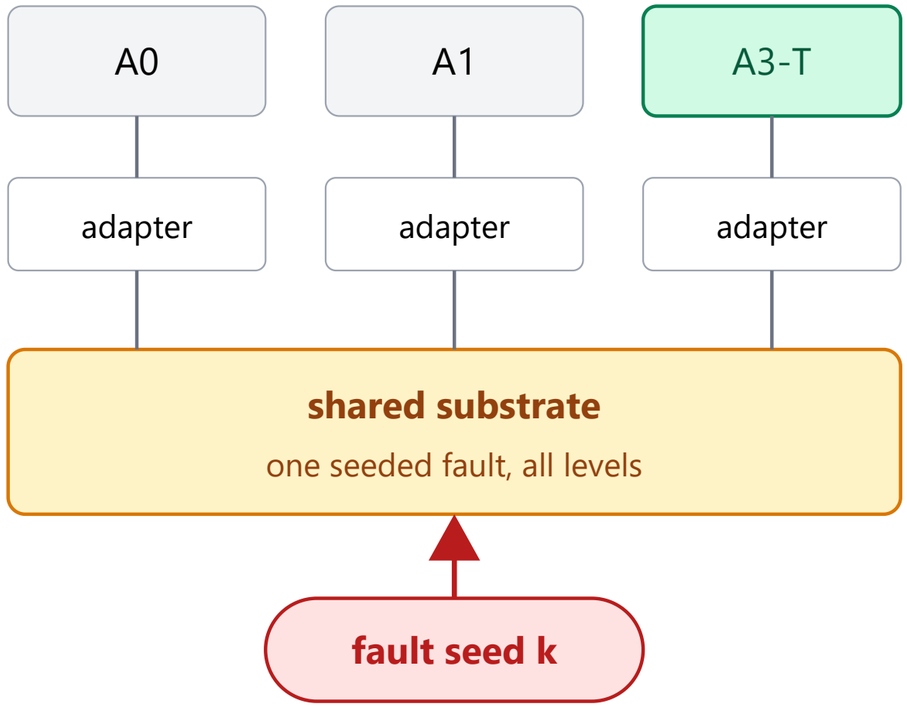
- On a healthy network, A0 and A3-T differ mostly in cost
- Real networks lose packets, lose links, freeze nodes
- Question: does the interface level still matter when the network misbehaves?
- Fault rate λ is a stress knob, set above realistic levels on purpose

Same task (tenant onboarding), same model, same budget, same seeds. Only the level changes.

BACKUP · INJECTED FAULTS 2/3

How faults are injected

- Faults: packet loss, latency, downed link, frozen node, daemon restart, held lock
- Injected in the **shared substrate**, below every adapter
- Never inside a level: that would test the level's own resilience
- **Seeded**: the same fault sequence is replayed at every level



Every level sees the same event at the same moment. Partial apply, double apply, lockout are outcomes we measure, not inputs.

BACKUP · INJECTED FAULTS 3/3

Results under injected faults

Onboarding task, λ = 0.3, n = 4 per level, one model, same budget:

Level	Success	Damage-free	Steps	Tokens
A0	**0 / 4**	100%	25.0	127k
A1	2 / 4	100%	25.0	228k
A1+	1 / 4	100%	25.0	239k
A3-T	**4 / 4**	100%	5.5	35k

- A3-T completes every run; A0 none; A1 and A1+ in between
- No level damaged a pre-existing tenant, even under faults
- A3-T sends a typed intent and does not need to read the degraded network;
  A0 and A1 do, and lose their way

Caveat. Four runs per level; and typed intent, transactions and the actuation channel change together, so this pilot does not say which one does the work.