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

Lorenzo Bracciale*, Pierpaolo Loreti*, Andrea Mayer*, Stefano Salsano*, Wim Henderickx†
*University of Rome Tor Vergata, Italy, {name.surname}@uniroma2.it
†Nokia, Belgium, wim.henderickx@nokia.com

Extended version of the paper presented at CNSM 2026, Alcalá de Henares, 27 October 2026 [1]

Code: https://github.com/netgroup/ani-gamut (tag `cnsm2026`) Project page: https://netgroup.github.io/ani-gamut/

***Abstract*—Large Language Model (LLM) agents are increasingly trusted to operate live networks: they read state, change configuration, and verify the result. A first-order question is left implicit: *at which level of abstraction* should the agent operate? We make the interface-abstraction level an explicit, controlled experimental variable, organizing agent–network interfaces into a spectrum from raw CLI (A0) through bounded wrappers (A1) and standardized model-driven configuration (A2) to typed transactional service intent (A3-T), reconciled source-of-truth automation (A3-R), and their combination (A4). We present ANI-Gamut, a reproducible, open-source playground that exposes the *same* task at several levels on a single, densely populated *brownfield* substrate, where many coexisting services share resources and collateral damage actually arises. We instantiate and measure four points of the spectrum and describe the others only at the conceptual level, and record three dependent variables as the substrate is stressed by injected faults: task reliability, collateral damage against pre-existing tenants, and operational cost. In a pilot with small run counts (n=20, n=6 and n=4 per level), the interface level moves reliability and cost sharply: on a live change-set task a raw-shell agent fails on all six seeds while a typed transactional interface succeeds on all six (paired McNemar p=0.031, on six discordant pairs), at roughly an order of magnitude less cost, as the engineering effort migrates from the agent to a reusable transaction layer. Collateral damage, by contrast, is absent at every level, whether benign or under faults: in a tenant-isolated substrate the agents fail safe, and the blast radius is held by the substrate's isolation, which moves the safety question from the agent to the substrate. ANI-Gamut realizes and extends the benchmarking-playground vision of [2], turning the interface-abstraction level from an implicit design choice into a measured variable of agent reliability and cost.**

***Index Terms*—agentic AI, network management, agent–network interface, abstraction level, reliability, safety, recovery, benchmarking, SRv6, NETCONF, source of truth, Model Context Protocol.**

I. Introduction

NETWORK operations are moving from human-typed commands to LLM agents that act on live infrastructure, observing the running network, changing it, and confirming the result. Attention has gone almost entirely to the agent, its model, prompting, and planning loop, while a prior question stays implicit: *at which level of abstraction* should the agent operate? The same change can be presented to the agent as a raw device shell, as a typed declarative service request, or as an edit to a reconciled source of truth, and these are very different contracts between the agent and the network.

Software engineering has already learned that this choice is first-order. How reliably a coding agent works is governed by the agent–computer interface, not the model alone. A constrained, well-shaped interface lets a given model succeed where a raw shell makes it fail [3], [4]. Networking has begun to borrow the vocabulary, and recent agent–network interface (ANI) work already uses the term, but at a single, fixed abstraction [5]; no study varies the level and measures what it buys. The question is not academic. Operators and vendors are climbing an abstraction ladder today, from device CLI to model-driven configuration (NETCONF/YANG [6], [7]), to service-intent orchestration (Cisco NSO, Nokia EDA, Juniper Apstra [8]–[10]), to reconciled source-of-truth automation (NetBox/Nautobot [11]). Which rung is safe to hand to an autonomous agent is being decided in practice, and it is the decision this paper turns into a measurement.

What makes the question sharp is the failure mode that matters in operations. A configuration agent that completes its task but silently breaks an unrelated, already-deployed service has, in practice, caused an outage. Deployability therefore has two legs: the agent must reach its goal *and* leave every pre-existing service intact, the latter at carrier-grade reliability. Collateral damage is a zero-tolerance property, so we score not only whether the agent finishes but whether a run leaves the network damage-free, and how both hold up as the surrounding state grows denser and faults intrude.

We make the **interface-abstraction level** an explicit, controlled experimental variable. We organize agent–network interfaces into a spectrum, A0 to A4 (Fig. 1), from the raw shell (A0) through bounded per-device wrappers (A1) and standardized model-driven configuration (A2) to a two-dimensional top: typed transactional service intent (A3-T), reconciled source-of-truth automation (A3-R), and their com-

bination (A4). Holding the task, the network, the model and the observation interface fixed, we change only the level the agent acts through, and read off reliability, safety, recovery and cost.

Measuring this well needs a substrate where collateral damage can actually occur. That calls for a *brownfield*: a network already densely populated with heterogeneous services that share resources, so that a careless change can shadow a prefix, reuse an identifier, or trip a global knob. Toy playgrounds with a handful of services hide the very interactions we want to stress, and a conclusion drawn at toy scale need not hold at production scale. The machine-learning fields that matured did so on shared benchmarks whose difficulty matched the real problem, such as ImageNet for image recognition [12] and the WMT shared tasks for machine translation [13]; agentic network configuration has no equivalent yet, and constructing one is part of the work.

This paper makes three coupled contributions.

- *The interface level as a controlled variable.* We define the A0–A4 abstraction spectrum and use it as the independent variable in a controlled study of agent reliability, tracing where each safety property enters. The principle is established in software engineering but not yet in networking under controlled conditions.
- *A reproducible brownfield playground.* We build, and release as open source, ANI-Gamut: a single multi-service substrate, a layered scenario description, and a seeded initial-state generator that populates a realistic brownfield, on which the same task is exposed at several levels. It instantiates the benchmarking playground envisioned in [2]; the release is public [14].
- *A methodology for the production-readiness gap.* Damage-free operation is a near-zero-probability, zero-tolerance target, and estimating it is not straightforward. The method combines a binary per-run safety outcome, proportion confidence intervals, run-budget sizing by the rule of three, paired comparisons over shared fault seeds, and stress sweeps over state density and fault rate. This lets an agent's collateral-damage probability, and its distance from the production regime, be estimated and compared across interface levels.

Existing agentic-network benchmarks differ from ours on two axes: they fix the agent–network interface as one harness, and they populate little coexisting configuration state, mostly to score troubleshooting and root-cause analysis [15]–[20]. We instead treat the interface level as the experimental treatment, and push toward realistic brownfield state heterogeneity and density for *safe configuration*, not repair. ANI-Gamut is released as open source [14] (tag `cnsm2026`: the harness with its per-level adapters, the scenario generator and the SRv6 substrate); this extended version adds the detailed methodology, the system architecture and the appendices to the conference paper [1].

In a pilot on a multi-tenant SRv6 brownfield the effect is sharp: at a matched budget, on a live change-set task a raw-shell agent fails on all six seeds where a typed transactional interface succeeds on all six, at an order of magnitude less cost. Collateral damage, in contrast, stays absent at every level: the tenant isolation of the substrate bounds the blast radius, a result we return to. The study measures four points of the spectrum (A0, A1, A1+ and A3-T); two points stay conceptual (A2, A3-R) and one is future work (A4), so what the data supports is a contrast between the ends of the implemented range, not a curve over the whole spectrum.

The rest of the paper is organized as follows. Section II reviews agent interfaces and network benchmarks; Section III defines the abstraction spectrum; Sections IV and V describe the framework and the brownfield playground; Section VI states the methodology; Section VII reports results; and Sections VIII and IX discuss implications and conclude.

II. Background and Related Work

A. Agent–computer and agent–network interfaces

The software-engineering community has shown that the interface exposed to an LLM agent, and not only the model behind it, governs how reliably the agent acts. SWE-agent introduces the *Agent–Computer Interface* (ACI) and shows through ablations that replacing a raw shell with a small set of agent-tailored actions changes task-resolution rates by a wide margin [3]. CODESTRUCT reports the same effect for code editing: moving from free-text patches to structured operations over syntax-tree entities sharply reduces invalid edits [4]. There is a safety counterpart to this reliability story: granting an agent executable tools rather than text alone measurably raises the rate of unsafe actions under identical prompts and policies [21]. In networking, the term *Agent–Network Interface* (ANI) and the use of the Model Context Protocol (MCP) to expose network operations as tools were introduced by the IETF NetConfBench draft [5]. We port the ACI principle to networking and add what the software-engineering setting does not address: operational safety and recovery on a live network.

B. LLM agents and benchmarks for network management

Several benchmarks evaluate LLM agents on network tasks. NetConfEval asks whether models can translate requirements into formal specifications, function calls, and low-level configuration, and already observes that higher-abstraction targets are easier than raw configuration [22]; it studies these as static translation tasks, not as an agent acting in a closed loop. NIKA [15] and NetOpsBench [16] are interactive arenas for troubleshooting and root-cause analysis on emulated networks, scoring detection and localization through bounded observability tools. The Confucius framework runs multi-agent LLMs in production and routes actions through validated intermediates rather than raw configuration [23]. Related efforts benchmark autonomous remediation in cloud and microservices settings [24] and survey the design and safety of agentic NetOps and AIOps [25]. Mani *et al.* make a complementary point outside benchmarking: having an LLM emit code against high-level APIs, instead of touching raw data, improves correctness and explainability [26]. Work at CNSM and TNSM has taken the intent route: policy generation from natural language for

intent-based application management [27], flow-rule generation for SDN with retry-based deployment validation [28], and LLNet, which instructs softwarized devices from intents through a small language model [29]. Our novelty is not intent-based management, which that line already pursues, but the controlled comparison of the abstraction levels at which an agent is allowed to act. In all of this work the interface through which the agent acts is fixed as a single harness; none treats it as an experimental variable.

C. Industrial network automation and autonomy levels

Our premise is that operator networks are already climbing an abstraction ladder, so an AI agent will increasingly have to act *through* these layers rather than around them. Standardized model-driven configuration is mature: NETCONF/YANG with candidate datastores and confirmed commit [6], [7], RESTCONF [30], and the vendor-neutral OpenConfig/gNMI stack [31]. Above the device, intent-based networking [32] and autonomic networking [33] define declarative, outcome-driven control, while source-of-truth–driven automation [11] and commercial controllers such as Cisco NSO [8], Nokia SR Linux/EDA [9], and Juniper Apstra [10] reconcile a versioned desired state against live device state. Liu *et al.* organize policy languages by their level of abstraction along the intent-refinement pipeline [34].

Operator and vendor bodies have also *classified* how far this automation has progressed. The TM Forum Autonomous Networks programme defines six levels, from Level 0 (manual) to Level 5 (fully autonomous), with an evaluation methodology (IG1252) that scores five cognitive dimensions, Intent/Experience, Awareness, Analysis, Decision, and Execution, as owned by personnel or by the system [35]. 3GPP SA5 standardizes an equivalent grading in TS 28.100 [36], and ETSI ZSM specifies the closed-loop, intent-driven architecture these levels assume [37]. These frameworks classify *autonomy*: how much of the decision loop the system, rather than a human, owns. Our spectrum (Section III) measures a different and complementary quantity, the abstraction of the interface the agent acts *through*. The two axes are orthogonal: an agent at any autonomy level still acts on the network through some interface-abstraction level. To avoid clashing with these autonomy levels, and with OSI layer numbering where L2 and L3 already carry fixed meanings, we denote our interface-abstraction spectrum A0–A4 (Section III) and leave "L" to its established uses.

D. The gap

Two gaps cut across this work. First, the agent–network interface is always a single fixed harness, whether a typed MCP toolset, an agent–cloud interface, or an emulated CLI [15], [16], [18]: the interface abstraction level itself is never treated as a controlled variable, held constant across task, network, and model and correlated with reliability, safety, and recovery. NetConfEval [22] does compare abstraction levels, but as independent static translation tasks (requirements to specification to low-level configuration), not as one task an agent carries out through interfaces held constant and varied across runs on a live network; the IETF NetConfBench draft [5] coins the term "agent–network interface" yet fixes it at a single device-CLI abstraction. The closest a benchmark comes to manipulating the interface is OperAID [38], whose tool-access ablation moves a 5G-core remediation agent from 11% to 61% success; but it toggles read-only *observation* tools at a fixed raw action interface, confirming that the interface matters while leaving the abstraction level of the agent's *actions*, the variable we study, untouched.

Second, what existing benchmarks under-populate is not device count but the global configuration state that makes configuration hard. A production backbone or datacenter carries, at the same time, tens to about 10^2 heterogeneous service classes and on the order of 10^5 to 10^6 configuration items in total; it is the size and heterogeneity of this coexisting state, not the number of nodes, that produces the resource sharing and contention where collateral damage occurs. Existing arenas populate far less of it, and even NETPRESS's [17] thousands of nodes form a single-purpose static capacity-planning graph rather than a dense, heterogeneous live configuration. ANI-Gamut does not reach production scale either, but it approximates it from below far more closely than toy substrates: on a modest fabric (a few tens of nodes) we populate on the order of ten coexisting service classes (SRv6 east-west and north-south tunnels, firewall policy, underlay routing, floating-IP, addressing, VRFs), each with 10^3 to 10^4 rules or configuration items. Appendix C gathers the public evidence behind these orders of magnitude, and Appendix D compares existing benchmarks on this state-complexity metric.

Our own prior playground [2] supplies the starting point: the SRv6 substrate and the typed transactional interface that becomes A3-T here. What is new in ANI-Gamut is the interface level as a controlled variable, the layered brownfield generator with its co-emitted oracle, and the zero-tolerance damage metric with the statistics that go with it.

By treating the interface abstraction level as the independent variable on this dense, multi-service brownfield substrate, ANI-Gamut closes both gaps.

III. The Interface-Abstraction Spectrum

We organize agent–network interfaces along a spectrum, shown in Fig. 1, from the rawest device access to fully reconciled source-of-truth automation. The spectrum is the conceptual contribution of this paper: it names *where* an agent acts, independently of which model or agent architecture is used. Table I summarizes the six levels and the aligned subset ANI-Gamut evaluates.

A. The ordered base: A0 to A2

At the bottom of the spectrum the agent operates on individual devices. **A0** is the raw device shell, an unbounded action space with no transactional semantics. **A1** wraps per-device configuration behind a bounded interface (read the running configuration, apply a change, run a validation command), still expressed as free-text commands. **A2** is standardized,

TABLE I
THE INTERFACE-ABSTRACTION SPECTRUM AND THE SUBSET ANI-GAMUT EVALUATES (IMPL. IMPLEMENTED, CONC. CONCEPTUAL, FUT. FUTURE WORK).

Level	Interface	Key safety / transaction property	Eval.
A0	raw shell	none; unbounded action space	impl.
A1	bounded CLI	generic guardrail + manual recovery	impl.
A2	NETCONF/YANG	device candidate commit + validation	conc.
A3-T	service intent	workflow transaction + validation + rollback	impl.
A3-R	reconciled SoT	drift detection + reconciliation	conc.
A4	intent + SoT	both high-level axes (Pareto-dominant)	fut.

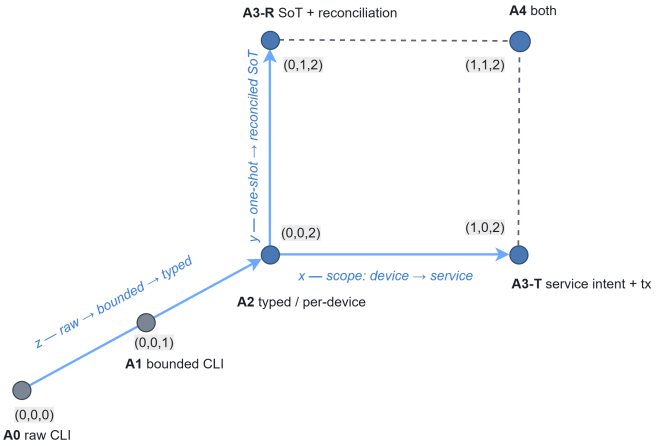


Fig. 1. The interface-abstraction spectrum. The ordered base A0–A1–A2 runs along z (representation maturity); the high-abstraction interfaces form a 2D top (Axis A: scope; Axis B: state/control), with A2 as origin, A3-T and A3-R as the two incomparable corners, and A4 their combination.

model-driven configuration over NETCONF/YANG: typed and vendor-neutral, with candidate datastores and confirmed commit that provide device-level transactional safety [6], [7]. These three levels are ordered by how far the interface moves the agent from raw mutation and by how much safety it interposes.

B. The two-dimensional top: A3-T, A3-R, and A4

Above A2 the spectrum is not a single ladder but a plane spanned by two orthogonal axes. The first axis is *scope*, from per-device configuration to service-level, cross-device intent. The second is the *state and control model*, from one-shot, episodic actions to a persistent, versioned source of truth continuously reconciled against live state. **A3-T** (transactional service intent) advances the scope axis: the agent issues a typed declarative service request that a workflow transaction manager applies with active validation and rollback; this is the interface realized by the SRv6 playground [2]. **A3-R** (reconciled source of truth) advances the control axis: the

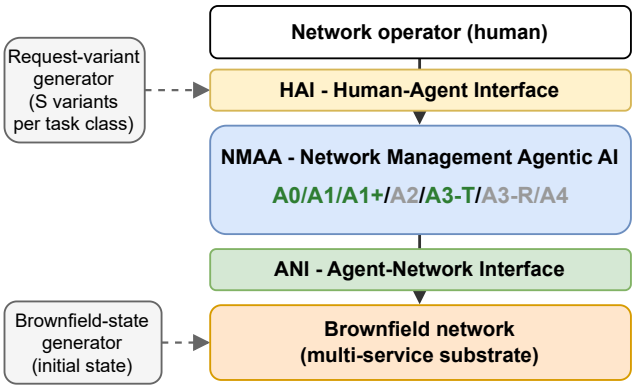


Fig. 2. ANI-Gamut at a glance. The NMAA system sits between the human–agent interface (HAI), where the operator states a configuration requirement, and the agent–network interface (ANI), through which it acts. Only the ANI level varies (green: implemented; grey: conceptual); the HAI request, the model and the observation are held constant. Two generators drive each run: the request variants of a task class and the seeded brownfield state.

agent edits an authoritative, versioned desired state, and a reconciliation loop detects drift and remediates [11]. A3-T and A3-R are not ordered with respect to each other; each provides a different kind of safety. **A4** occupies the remaining corner, combining service intent with reconciliation, and is where commercial systems such as Cisco NSO, Nokia EDA, and Juniper Apstra already operate [8]–[10]. Placing A2 at the origin, A3-T, A3-R, and A4 are the points $(1,0)$, $(0,1)$, and $(1,1)$: A4 Pareto-dominates both A3 variants, which is why it carries the higher number even though A3-T and A3-R are mutually incomparable.

C. Abstraction exposed, not implementation

A level denotes the abstraction *exposed to the agent*, not the stack used to enforce it, and a level need not be built on the one below. A3-T does not require A2: the SRv6 playground enforces its typed `encap/decap` intent through ordinary kernel routing commands, with no NETCONF involved, and a configuration-domain A3-T can likewise be realized by serializing typed intent into device commands. This decoupling lets us study the interface the agent sees while leaving the realization substrate free, and it is what allows the same task to be exposed at several levels (Section IV) on one emulated testbed (Section V).

IV. ANI-GAMUT: LEVELS AS A CONTROLLED VARIABLE

ANI-Gamut makes the interface level the one variable that changes between runs. The system under test is a *network-management agentic-AI* (NMAA) system that sits between two interfaces: it receives a configuration requirement from a human operator over a *human–agent interface* (HAI), a natural-language prompt with zero or more attached files, and acts on the network through the agent–network interface (ANI) (Fig. 2).Appendix A expands this with the scenario generators and the oracle. The ANI level is an internal choice of the NMAA system, invisible to the operator: the same HAI request

is met by A0 with a raw shell and by A3-T with typed intent. We therefore hold the HAI request, the observation interface and the model fixed and vary only the ANI level, so two NMAA systems compared here are identical except for the abstraction at which they act. We build the harness by unifying two existing codebases: NetAiBench, a configuration harness developed in our group that already sits at A1, and the SRv6 playground [2], which already realizes an A3-T service interface. The full A0–A4 spectrum is the conceptual contribution; the demonstrator implements the aligned subset A0, A1 and A3-T, leaving A2 and A3-R conceptual for the reasons given in Section VI.

A. A unified harness and a common ANI

The harness derives from NetAiBench's modular architecture. Every task carries one specification: a natural-language intent, the topology, the expected end state, machine-checkable test cases, and the *baseline invariants* that the pre-existing services and global settings must keep satisfying. A common ANI sits between the agent and the substrate, with one adapter per level that translates the agent's actions into substrate operations. The agent is a controlled factor, not the object of study: single-turn, ReAct and planner–executor modes, with and without recovery, run unchanged across levels. A shared evaluator scores every run identically and observation is held constant, so that what differs between two runs of the same task is only the level the agent acts through. The substrate is the unified Containerlab testbed of Section V, on which both source codebases already run.

B. Per-level adapters (same task, different interface)

At **A0** the adapter is a raw shell on the node: an unbounded action space with no transactional semantics, the naive path obtained by removing every guardrail. **A1** keeps the same free-text representation and adds two separable mechanisms. The first is *bounding*: a configuration-command allowlist, a block on out-of-scope shell, and protection of the management plane. The second is *recovery*: an auto-captured per-device snapshot the agent can roll back. This bounding is a *generic* guardrail, keeping the agent inside the configuration surface and the device reachable, not an anti-damage blocklist; hand-picked prohibitions against specific destructive commands would make the level safe by our own tuning, so we move them into the **A1+** ablation (Section VI). In the configuration domain A0 and A1 are deliberately close, differing only by the allowlist, since both expose raw CLI; the substantive jump is to typed intent. **A3-T** exposes typed declarative *service* intent: the agent issues a cross-device service request that a workflow transaction manager applies with active validation and compensating rollback. It already exists in the SRv6 playground as schema-checked `encap`/`decap` messages with a per-direction transaction and an ICMP-in-SRv6 validation probe [2]; we generalize it to the configuration domains by promoting NetAiBench's internal typed model and inverse serializers, today used only for rollback, to a public interface. **A2** (standardized NETCONF/YANG with candidate datastores and confirmed commit [6], [7]) and **A3-R** (a versioned source of truth reconciled against live state [11]) are defined but left conceptual: no mature open-source orchestrator gives an honest, measurable A2 agent path on this substrate, and a reconciliation loop is a journal-phase extension. Attribution does not depend on them; it rests on contrasts we can run, the A0-versus-A1 bounding and recovery decomposition and A3-T-internal ablations, detailed in Section VI.

V. A Reproducible Brownfield Playground

The playground is the substrate where collateral damage can occur, and we release it as a reproducible open-source artifact with the paper: a single multi-service substrate, a layered scenario description, and a seeded initial-state generator, available at https://github.com/netgroup/ani-gamut [14].

A. A unified multi-service substrate

All services coexist on the same devices, as on a real provider edge: SRv6 east-west and north-south tunnels, firewall policy, underlay routing, floating-IP and addressing share each node's namespaces. That shared occupancy is what makes resource contention, and therefore collateral damage, real. It is feasible because both source codebases run on Containerlab with Linux and FRR, so their labs merge onto one node image (the SRv6 node, with `seg6` and FRR, extended with `nft` and `iptables`); NetAiBench's SR Linux and OpenBSD drivers add device heterogeneity. The SRv6 service class runs on a NEXT-CSID (micro-SID) data plane whose locators follow the F3216 addressing recommendations of the IETF SRv6ops draft [39] (a single 32-bit block, with 8-bit set and node fields and loopbacks taken from the locator), so the testbed reflects current operator practice rather than an ad-hoc scheme. Topology and state are two independent scales. The number of emulated nodes is bounded by host resources, but the pre-existing state (tenants, VRFs, SIDs, routes, rules) is kernel and configuration data, not extra containers, so it grows independently. Our hypotheses live in the *state*, so we keep a modest fabric, six PEs, three IGWs, two EGWs and twelve hosts, about twenty-five nodes, and populate it densely. On it we pre-load on the order of ten coexisting service classes with 10^3 to 10^4 items each, an under-approximation of the production orders of magnitude of Section II. State heterogeneity and density are parameterized and swept: the prediction is that the safety advantage of higher abstraction grows with density, hidden on a sparse substrate and revealed on a dense one.

B. A layered, composable scenario generator

The brownfield is built like a container image. A topology base, the Containerlab description, carries a stack of service-class layers declared in a composition manifest, a brownfield "Dockerfile" written in YAML. Each service class is a plug-in layer with three phases. An `init` phase establishes the class prerequisites, enabling `seg6`, reserving VRF and SID space, or setting up base firewall chains. A `generate` phase writes the class's $T{=}0$ instances as real kernel and device

state, together with the baseline invariants for those instances, so the oracle is emitted alongside the state. A `verify` phase checks every instance of the class, exhaustively or sampled. The manifest lists the layers, their dependencies, the per-class parameters (instance counts, prefix overlap, load skew, distribution shapes) and the seed; topology and manifest together determine the scenario. Phases run globally, `init` then `generate` then `verify`, with layers ordered by dependency, so the underlay routing is in place before the overlays that ride on it. The oracle is therefore distributed across layers, each class verifying its own instances, and the agent's target is a held-out instance of a task class, declared in the manifest and placed to interact with existing state, on the same PE, an adjacent prefix, or a shared SID pool; the same generator draws the random variants of that class (Section VI) that make up a cell's paired runs. A realistic initial state, rather than a merely dense one, means several things at once. Every pre-existing service passes its invariants at $T{=}0$, or the scenario is regenerated. Service sizes and node load follow plausible skewed distributions, shared-resource dependencies and prefix adjacency or overlap occur at a tunable rate so that shadowing and collisions are genuine traps, the per-class mix is controlled, and the target sits in the populated context rather than in isolation. How much the generator's realism affects the conclusions is itself a question, which the density sweep turns into a follow-up sensitivity study.

C. Substrate-level fault injection

Faults are injected in the shared substrate, identically for every level, never inside a per-level adapter. Injecting "operation X fails with probability p" at A3-T would bake the level's own resilience into the stimulus and destroy comparability; instead we inject the cause low, below the ANI adapters, and measure how the effect propagates through each interface. The admissible primitives act at node and link level and strike all levels alike: packet loss, latency and jitter, a downed link, an unresponsive (frozen) node, a daemon restart, a held route-table or datastore lock, a concurrent writer mutating shared state during the task, and transient exhaustion of a shared SID or address pool. Transport-specific faults are not admissible as primitives, because transports differ by level; they are modeled as node or link faults that hit every transport equally. Partial application, double-apply on retry, and lockout are measured outcomes, not inputs: they emerge when a primitive strikes at the wrong moment, so the same physical event can leave A0 with half-applied state while an A3-T transaction rolls back or retries idempotently. Faults are seeded and the same seed is replayed across levels, common random numbers made possible precisely because the fault lives in the substrate; observation is out-of-band, run by the harness rather than the agent's level-specific validation, so success and safety are measured identically everywhere. The fault rate is a tunable stress knob, set deliberately above realistic levels, and the gap between levels as a function of it is a headline analysis alongside the density sweep.

D. A reproducible open-source release

Adding a service class is adding a layer with its three phases; the playground composes the layers and derives the oracle automatically. This realizes and extends the single extensible benchmarking playground envisioned by the accepted SRv6 paper [2]. The tooling, the layered scenario description and the initial-state generator with the harness, is released as open source with the paper [14]: `playground/` holds the harness, the per-level adapters, the generator and the oracle (Apache-2.0, a derived version of NetAiBench); `substrate/` holds the SRv6 multi-service substrate (GPL-3.0) ; designing it for reuse is part of the contribution. Large campaigns are made practical by engineering the testbed for scale, through parallel scenario execution and a reduced per-node memory footprint.Appendix B reports the parallelization scheme and the footprint reduction.

VI. Experimental Methodology

A. Hypotheses

We test six directional hypotheses, some of which may be refuted; that is what makes this measurement rather than advocacy. **H1 (safety):** higher-abstraction interfaces lower the rate of collateral damage relative to the A0/A1 baseline, at constant task and model. **H2 (recovery):** higher levels return the network to a safe, correct state more often after a committed error. **H3 (expressiveness trade-off):** functional success is not monotonic in abstraction, because very high levels can cap what is expressible on some tasks. **H4 (cost):** higher levels reach a correct-and-safe state at lower cost and lower variance. **H5 (model $\times$ level):** the benefit of abstraction is larger for weaker models, the interface compensating for model capability. **H6 (attribution):** the reliability gain comes from transactional and recovery semantics entering at specific points, not from typed representation alone.

B. Design and controlled factors

The interface level is the treatment in a factorial design that crosses it with the task class, the model, and the agent mode, swept over two stress parameters: the state density of the brownfield and the substrate fault rate λ. A *task class* is a configuration requirement stated on the HAI, a prompt template with zero or more attached files; the task classes span the service classes that populate the substrate, and two classes that request the same reconfiguration in different words isolate the effect of expression from the effect of the situation. For each cell the generator draws S random *variants* of the task class, each a concrete request (tenant, prefixes, target) placed in a freshly generated brownfield; the prompt text and the file structure are identical across a class's variants, so what varies is the situation, not the wording. These S variants are the per-cell runs over which the statistics below are estimated, and each variant, bundling the request, the brownfield, and the fault seed, is replayed across the levels, which is what makes the comparison paired.

We compare interface levels under a matched agent budget. The execution budget is held constant across levels, with

the same model, system prompt, agent mode, step and token caps, retry policy, and observation interface, so that only the action interface differs. The development budget is equalized by a single shared agent with no level-specific tuning beyond exposing each interface. We additionally report cost (tokens, messages, steps, wall-clock) as a dependent variable (H4): the question is not only whether a higher level is safer at equal budget, but whether it reaches a safe state at lower budget.

The slice implemented here is A0, A1 and A3-T, with A1+ as an ablation, while A2, A3-R and A4 stay conceptual. This still answers the question: the claim concerns the *trend* from the low-abstraction baseline to the high-abstraction interfaces, not a continuous monotone curve, and attribution is preserved through within-level ablations rather than through the absent A2/A3-R contrasts.

C. Metrics: a zero-tolerance safety outcome

Agents are usually scored on mean task completion, but moving a success rate from 95% to 97% leaves a configuration agent far from deployable. A deployable agent must reach a success probability indistinguishable from one *and* damage no pre-existing service at carrier-grade reliability, the "five nines" of network practice, a collateral-damage probability on the order of 10^{-5}. No current system is near this bar on either leg; the playground measures how far a system is and which knobs move it closer. This fixes the role of every metric: task completion and the breadth of damage do not rank near-deployable systems, since none exists, but show that any system for which they are non-trivially measurable is by that fact far from production, and chart the directions in which the knobs (level, model, agent mode, stress) move it.

The unit of damage is the *service instance*, one deployed service such as a single SRv6 tunnel, grouped into service classes by type. An instance is damaged if any of its baseline invariants fails after the agent acts, counted once however many invariants it carries. Per run we record the per-class damage fraction B_c over the whole gamut of pre-existing services (SRv6 east-west and north-south, firewall, routing, floating-IP) and the global invariants (backbone reachability, forwarding, default-route removal, VRF integrity). Since collateral damage must be zero, incidence matters more than magnitude: the per-run outcome is binary, the damage vector $\mathbf{B} = (B_c)$ is null or not, and the primary safety statistic over a set of runs is the **damage-free run rate** $P(\mathbf{B} = \mathbf{0})$. Magnitude is kept as a secondary, diagnostic measure: the macro-average $\bar{B}$ across classes (equal class weight, insensitive to differing instance counts) and the per-class vector when damage occurs, with pooled micro-averages reported only as an operational footprint. The functional-success rate S_{tc}, the fraction of test cases the newly configured service passes, is the completion leg, measured but not the safety headline. Each run also carries an outcome label: clean success, dangerous success (works but damages), clean failure, or destructive failure. Recovery (H2) uses the same zero-damage lens: the fraction of fault-hit runs that still end damage-free and correct. Observation is out-of-band, so success and safety are scored identically at every level; the oracle reuses NetAiBench's invariant checks and the SRv6 ICMP-in-SRv6 probe, with the generator co-emitting the per-instance invariants (Section V).

D. Statistics and attribution

The primary outcome is binary, so the estimand is a probability: per cell we estimate $\hat{p}(\text{damage}) = k/S$ with a Wilson or Clopper–Pearson interval, since the normal approximation fails near zero, which is exactly the A3-T regime. Sample size follows from the zero claim, not from comparing means: by the rule of three, zero damages in S runs give an upper 95% bound near $3/S$, so asserting $p(\text{damage}) < 1\%$ for A3-T needs S on the order of a few hundred per cell, and the pilot fixes it. Common fault seeds are replayed across levels, which makes the comparison paired and lets McNemar's test work on the discordant pairs where the same physical event damaged the low level but not the high one, more direct than comparing marginal rates. Secondary continuous metrics (cost, time-to-safe-state, magnitude when damage occurs) use confidence intervals and mixed-effects regression with task and model as random effects, including the model$\times$level interaction (H5). We report success and cost per level with interval estimates. The density and λ sweeps are not part of this pilot and are part of the ongoing follow-up work. The hypotheses of Section VI-A and the analysis plan are published on the project page [40], together with the part of the plan that the pilot executed.

Attribution (H6) uses contrasts that probe one mechanism at a time, because A3-T and A3-R advance different axes rather than forming one staircase. A1 bundles *bounding* (allowlist, out-of-scope-shell block, management protection) and *recovery* (snapshot and manual rollback); we vary them as a 2×2 factor at constant free-text representation, separating how much *preventing* buys from how much *undoing* buys, which a single A0-versus-A1 contrast conflates. The A1-to-A3-T step then isolates service-scope typed intent with one-shot workflow transactions. The way an interface enforces consistency is itself a sub-axis of increasing principledness: an ad-hoc prohibition (the A1+ blocklist), predetermined domain-specific templates, standardized validation and commit (A2), and typed transactional intent (A3-T) approach the same guarantee by different routes, and we compare them empirically instead of assuming the typed route wins.

VII. Results

These results are a pilot: n=20 per level on the benign task, n=6 on the change-set, n=4 under faults; we rely on paired tests and explain the mechanism behind each one. Two service families are exercised, a benign onboarding task and a change-set on a live tenant, each run with and without injected faults, with one model held fixed across levels and a matched budget (same model, prompt, and step cap; only the action interface changes). Interface abstraction moves reliability and cost; it does not move the rate of collateral damage.

Reliability is governed by abstraction. On the benign onboarding task success grows monotonically over the low levels (Table II); A1+ above A1 is noise, since the Tier-2

TABLE II

CAPABILITY ON THE BENIGN ONBOARDING TASK (NO FAULTS; n=20 PER LEVEL; ONE MODEL, MATCHED BUDGET). WILSON INTERVALS ARE WIDE AND OVERLAP AT THIS n. PAIRED MCNEMAR: A0 AGAINST A1+ p=0.092; A1+ AGAINST A1 p=0.289.

Level	Success	Rate	Wilson 95%	Dmg-free
A0	4/20	0.20	$[0.08, 0.42]$	1.00
A1	7/20	0.35	$[0.18, 0.57]$	1.00
A1+	11/20	0.55	$[0.34, 0.74]$	1.00

TABLE III

HEADLINE CAPABILITY ON THE CHANGE-SET TASK (LIVE 25-SITE TENANT; TWO ADDS, TWO MOVES, ONE REMOVE; n=6 SEEDS, NO FAULTS; ONE MODEL, MATCHED BUDGET). BOTH LEVELS ARE DAMAGE-FREE; EXACT PAIRED MCNEMAR p=0.031.

Level	Success	Wilson 95%	Steps	Tokens	Dmg-free
A0	0/6	$[0.00, 0.39]$	40	∼375k	1.00
A3-T	6/6	$[0.61, 1.00]$	3	∼42k	1.00

blocklist can only forbid, not help solve. The separation becomes clean once the high level is in play. On a live change-set task (two adds, two moves and one remove on a 25-site tenant, n=6 seeds, Table III) A0 solves $0/6$ and A3-T $6/6$, with non-overlapping Wilson intervals and a significant paired test (exact McNemar p=0.031 on six discordant pairs); A0 reaches about 85% of the mesh on every seed yet never completes it, while A3-T always reaches 100%. The gap survives stress: under injected packet loss and node isolation on the onboarding task (λ=0.3, Table IV), A3-T solves $4/4$ where A0 collapses to $0/4$. The mechanism is that A3-T applies a typed intent through its actuation channel and does not depend on probing the data plane, which the faults have degraded, whereas A0 and A1 must read a broken network to choose each next command, and lose their way.

Cost is governed by abstraction. Solving through one typed `provision_service` call costs far less than composing raw commands step by step: on the change-set, A3-T uses 3 steps and about 42k tokens against A0's 40 steps (capped) and about 375k tokens, roughly $13\times$ fewer steps and $9\times$ fewer tokens; on onboarding the token cost already rises $+62\%$ from A0 to A1. Part of the ratio is budget exhaustion rather than inefficiency: A0 is capped at 40 steps and fails at the cap, so its cost is the budget it was given, not the cost of a completed run. The engineering moves into the layer instead of into each run: the transaction recipe is written and validated once there, so the per-run agent budget falls as the level rises.

Collateral damage is *not* governed by abstraction. The damage-free rate is 1.00 at every level, on both tasks, benign and under faults: no run damages a pre-existing tenant. The null has a clear cause. The agents fail safe, abandoning the task instead of corrupting their neighbors, and the substrate keeps tenants in separate VRFs, so even overlapping addresses on shared routers cannot leak; the substrate's isolation bounds the blast radius, and the agent's abstraction has no part in it. What decides collateral safety is where shared state lives,

TABLE IV

FAULT CALIBRATION (λ=0.3, ONBOARDING TASK, n=4 PER LEVEL, ONE MODEL, MATCHED BUDGET). *Success* IS THE FRACTION OF RUNS THAT REACH THE NEW SERVICE; *damage-free* IS THE RATE $P(\mathbf{B}=\mathbf{0})$ OF RUNS WITH NO COLLATERAL DAMAGE TO PRE-EXISTING TENANTS.

Level	Success	Wilson 95%	Dmg-free	Steps	Tokens
A0	0/4	$[0.00, 0.49]$	1.00	25.0	127k
A1	2/4	$[0.15, 0.85]$	1.00	25.0	228k
A1+	1/4	$[0.05, 0.70]$	1.00	25.0	239k
A3-T	4/4	$[0.51, 1.00]$	1.00	5.5	35k

and on this substrate the A1/A1+ blocklist, built to prevent damage, is inert, which is why its ablation reads as noise. It would bite on services with shared mutable state that tenant isolation does not protect, such as a stateful middlebox with a common ruleset that a careless flush can break; surfacing that is the natural next study.

The fault-robustness gap is confounded. Part of A3-T's advantage under faults is that its actuation is out-of-band from the data-plane disturbance, so while the actuation channel is the same for every level, the low levels also pay for having to observe a degraded data plane. Typed representation, transactional semantics, the out-of-band actuation channel and the engineering that went into the adapter vary together here; this pilot does not separate them, and the matched budget equalizes the agent's allowance, not these mechanisms. Controlling the actuation channel (A0/A1 with out-of-band actuation, or A3-T forced to observe in band) is the first ablation of the ongoing follow-up work.

VIII. DISCUSSION

A3-T and A3-R are different bets on safety: A3-T narrows what the agent can express to a typed service request and wraps it in a transaction, while A3-R lets it edit a versioned desired state that a reconciliation loop drives toward. Our slice tests the first and leaves the second conceptual, so which of them buys more safety is a question the framework is built to answer rather than one we settle here.

The expressiveness trade-off (H3) is the obvious objection to "higher is safer": a typed service interface cannot express a configuration its schema does not anticipate, so on tasks such as dynamic-routing tuning a leaner interface that stays close to the device can win on raw success. We keep such tasks in the corpus precisely so the expressiveness ceiling appears as a result rather than hiding as a gap.

Where does collateral safety come from? A hand-curated blocklist (A1+) forbids the specific destructive commands an operator has been burned by, while a structural interface makes whole classes of damage unrepresentable; on our substrate neither was exercised, because no level damaged a neighbor. The data points instead to the substrate: tenant isolation held the blast radius at zero on every run, so on a VRF-isolated brownfield collateral safety is a property of where shared state lives. The structural-versus-ad-hoc contrast becomes measurable only on services whose shared state the isolation does not cover, which is where we take the study next.

For an operator the question is one of placement, at which rung an agent acts reliably and cheaply on a given service given the model in hand and the stress on the surrounding state, and the framework turns that into the success rate and the budget at each rung rather than a matter of judgment.

The operator–agent interface (HAI) has its own abstraction spectrum, from a terse ticket to a fully structured intent; we hold it fixed so that the agent–network interface is the sole treatment, and leave that symmetric axis to future work.

Several threats temper the conclusions. The substrate is emulated, so absolute rates will differ on production hardware; we therefore read level-to-level differences, which the paired design protects, rather than absolute values. The generator argues for realism but does not prove it, and how much its choices move the conclusions is left to the density sweep of the ongoing follow-up work. A2 and A3-R are conceptual here, so the high end of the spectrum rests on A3-T alone; closing that is the first item of future work.

A subtler concern is the engineering that moves into the shared infrastructure: the transaction manager, active validation, and rollback are built once for all levels. This is the agent–computer-interface premise at work, and we attribute the resulting reliability to specific mechanisms through the ablations. The comparison therefore holds at a fixed, modest agent budget; the asymptotic regime, in which unbounded per-level engineering is poured into a low-level agent, is out of scope and operationally irrelevant, since operators do not invest person-years of engineering per agent.

IX. Conclusion and Future Work

We treated the abstraction level of the agent–network interface as an experimental variable and built ANI-Gamut to measure what each level buys. In this pilot, and between the two implemented ends of the spectrum, the interface level is a measurable determinant of how reliably and how cheaply an agent operates: on a dense brownfield, raising it carries a raw-shell agent from failing to succeeding at a fraction of the cost. Collateral safety, by contrast, did not move with the interface; on a tenant-isolated substrate it was a property of the isolation model, which relocates the safety question from the agent to where shared state lives. We still score collateral damage as a binary, zero-tolerance, damage-free run rate, and we supply the statistics, proportion intervals, rule-of-three budgeting, and paired tests, that let a near-zero damage probability be bounded and a small-sample capability gap be tested. The playground, the layered scenario description, the initial-state generator and the harness, is released as a first reproducible open-source benchmark with this paper (https://github.com/netgroup/ani-gamut, tag `cnsm2026`).

Future work fills in the spectrum and grows the benchmark. An implemented A2 (NETCONF/YANG on SR Linux) and a real A3-R (a reconciliation loop over a source of truth) would complete the high end and let A3-T and A3-R be compared directly; the A4 corner, where commercial systems already operate, follows. On the measurement side, the ongoing follow-up work quantifies how far undersized playgrounds distort conclusions and opens the release into a shared brownfield challenge.

Appendix A
Detailed System Architecture

Figure 3 expands Fig. 2 with the two scenario generators, the per-instance oracle, and the multi-service substrate.

Appendix B
Testbed Footprint and Parallel Scalability

A measurement campaign runs the same task across many cells (interface level, seed, state density, fault rate), so throughput is set by how many isolated worker labs run in parallel on one host. We engineered the testbed to raise that number and, by measuring it, found that the lab footprint is not the binding constraint. This appendix reports the progression of the per-lab RAM footprint and what actually bounds parallelism.

a) What runs per node.: Routing is static; FRR and haproxy are present in the node image but never started, so they cost disk, not memory. The per-router helper daemon connects to a broker that the parallel worker labs omit, so it exits a few seconds into boot and is absent during the long agent phase. A node at steady state is therefore close to an idle shell, and the dominant per-lab cost is the Docker/containerd per-container overhead times the container count, plus one host-side agent process per worker.

b) From one container per host to an aggregated lab.: The original layout uses one container per host: a worker lab is 23 containers (12 hosts and 11 routers). Collapsing the twelve host endpoints into a single aggregator container, with each host leg kept in its own in-container network namespace so that the data-plane semantics are unchanged (including the isolation of overlapping-tenant addresses), brings the lab to 12 containers; the 11 routers stay separate, since each is a distinct SRv6 node and is the agent's action surface. On the deploy VM (8 vCPU, 16 GB) the resident cost of a lab, measured as the free-memory delta around a deployment, drops from 129 MB to 78 MB, about 40%, and the all-green baseline is reproduced identically (38 invariants, including the hard overlap case). At roughly 5 MB per container, however, the lab substrate sits far below the 16 GB ceiling.

c) Estimated move to Alpine.: The node image is currently Ubuntu with a large Python virtual environment (e.g. datapizza, numpy, scipy), none of which a node needs at steady state; the built image is on the order of 2–3 GB. A lean Alpine variant carrying only the data-plane tools (bash, iproute2, iputils-ping, nftables, iptables) would be roughly 80–150 MB, a 15–30$\times$ reduction on disk, with faster build, pull and boot. Its effect on resident RAM is small: only the PID-1 shell shrinks (a few MB per node), while the per-container runtime overhead that dominates is distro-independent. Feasibility is confirmed: Alpine 3.21 ships iproute2 6.11, well above the version needed for the NEXT-CSID uSID data plane on the shared 6.8 kernel. The Alpine image is thus a clean win on disk, build and boot, not a memory multiplier.

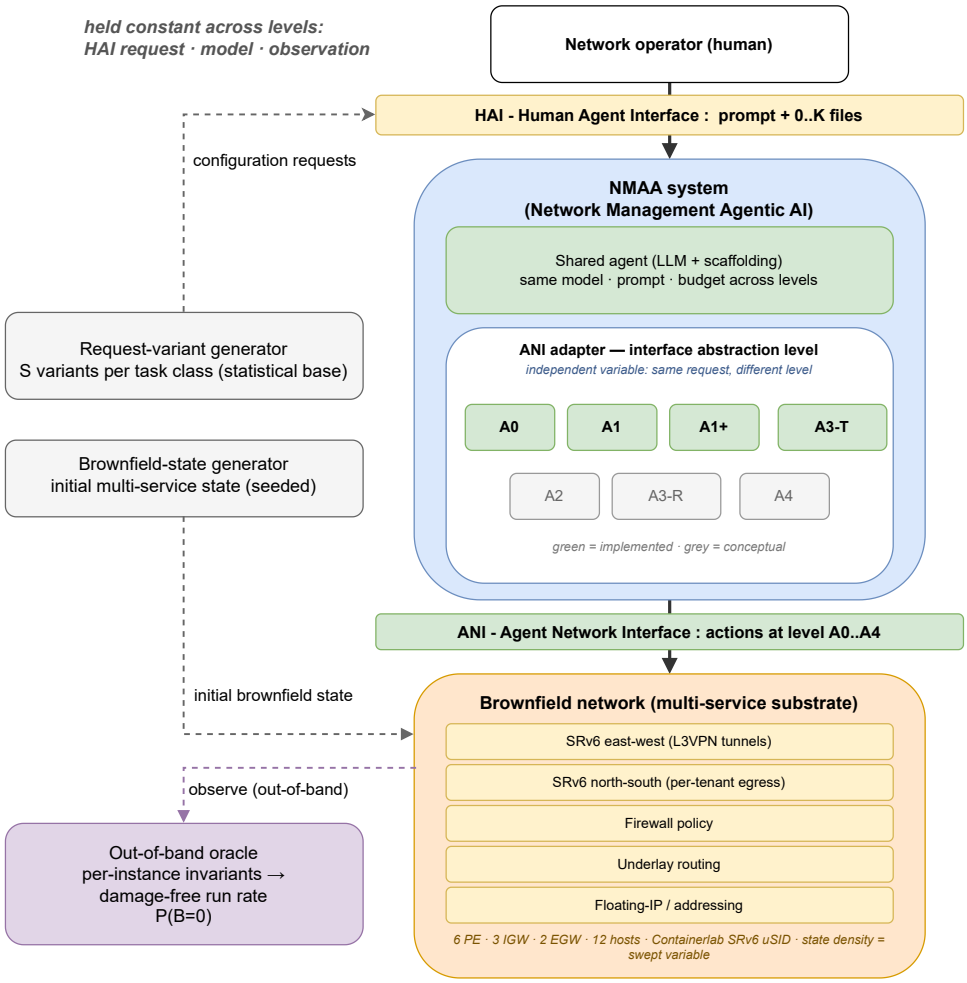


Fig. 3. Detailed ANI-Gamut architecture. The shared agent acts through one ANI adapter per level (green: implemented A0/A1/A1+/A3-T; grey: conceptual A2/A3-R/A4). The request-variant generator produces the configuration requests carried over the HAI; the brownfield-state generator seeds the multi-service substrate; the out-of-band oracle checks per-instance invariants and yields the damage-free run rate $P(\mathbf{B} = \mathbf{0})$. The HAI request, the model and the observation are held constant across levels.

TABLE V
PER-LAB FOOTPRINT ON THE DEPLOY VM (8 VCPU, 16 GB). RAM/LAB IS THE FREE-MEMORY DELTA AROUND ONE DEPLOYMENT; THE ALPINE ROW IS AN ESTIMATE.

Node image / layout	Cont./lab	RAM/lab	Image (disk)
One container per host	23	129 MB	∼2–3 GB
Host aggregator (current)	12	78 MB	∼2–3 GB
Lean Alpine aggregator (est.)	12	∼75 MB	∼0.1 GB

d) What actually bounds parallelism.: Putting the measured pieces together, a worker costs about 0.3–0.5 GB at steady state (the lab plus one host-side agent process, hundreds of MB of Python), so memory alone would allow on the order of 25–45 parallel workers; the bursty CPU load of simultaneous redeployments caps the practical number lower, around 4–6 in transient. Neither is the true ceiling: in practice the binding resource is the LLM API account (reachability, credit, and rate limit). Halving the container count and shrinking the image is a clean win on deployment time and host overhead, but the lever for higher useful throughput is the API quota, not the testbed footprint.

APPENDIX C
REAL-WORLD CONFIGURATION-STATE SCALE

This appendix substantiates the scale premise of Section II: the relevant measure of playground complexity is the size and heterogeneity of the global configuration state rather than the device count. As orders of magnitude, a production backbone or data-center network coexists on the order of 10^2 heterogeneous service classes and carries on the order of 10^5 to 10^6 configuration and state items. Table VI gathers public evidence, keeping *service classes* (heterogeneity) separate from *items per class* (density).

The density figure is well supported. A single backbone router holds on the order of 10^6 forwarding entries, namely

TABLE VI
PUBLIC EVIDENCE FOR REAL-WORLD CONFIGURATION-STATE SCALE.

Quantity	Order of magnitude	Source
Global BGP table (IPv4 FIB)	$\sim 10^6$ routes ($\approx$950k)	[41]
VRFs per device	up to $\sim 10^3$	[42]
BGP peers per device	up to $\sim 5\times 10^2$	[42]
ACL entries per device	10^3–10^4	[42]
Configuration size per file	10^2–10^3 lines	[43]

the global routing table [41], together with up to roughly 10^3 VRFs and 10^3 to 10^4 filter entries [42]; a network of tens to hundreds of such devices therefore reaches 10^5 to 10^6 distinct configuration and state items. The heterogeneity figure is more sensitive to definition. Counting distinct configuration feature-domains or coexisting service types, such as L3VPN, L2VPN and EVPN, Internet and peering, QoS, multicast, traffic engineering, segment routing, packet filtering, address translation, and management, and using the number of data models a platform exposes as a proxy,[1] the count lands in the tens to low hundreds, consistent with 10^2 at the lower bound. Counting service *instances* such as per-tenant VPNs instead raises the figure to 10^2 to 10^4. We thus phrase the heterogeneity as "tens to about 10^2 coexisting service classes" and treat 10^2 as an upper estimate rather than a typical value.

APPENDIX D
CONFIGURATION-STATE SCALE OF EXISTING AGENTIC BENCHMARKS

We re-examine existing agentic benchmarks on the same metric: the number of heterogeneous service classes in the initial state, and the configuration-state size per class. Table VII reports figures or estimates with sources, and flags values the original work does not state. Two groups emerge. Network-configuration arenas run on a live or emulated substrate but expose $O(1)$ to $O(10)$ service classes with small per-class state. Site-reliability and AIOps arenas score fault diagnosis and remediation over microservice or Kubernetes stacks; on the configuration-state metric they are off-axis, since their reported scale is a count of fault cases rather than coexisting configuration heterogeneity.

Across both groups, no benchmark combines many heterogeneous coexisting service classes with large per-class configuration state on a live substrate. The only entry reaching thousands of elements, NetPress, does so as a static capacity-planning graph of 5,493 nodes rather than a live operated network [17]; we therefore bind the toy-scale observation to live, interactive substrates. For reference, the substrate of Section V carries on the order of 10 coexisting service classes and 10^3 to 10^4 configuration items, already above the live benchmarks surveyed here, while production networks (Appendix C) sit two to three orders of magnitude higher again.

[1]Vendor YANG model inventories run to several hundred modules per release; see https://github.com/YangModels/yang.

ACKNOWLEDGMENT

This work has received funding from the Italian MUR PRIN NEWTON project.

REFERENCES

[1] L. Bracciale, P. Loreti, A. Mayer, S. Salsano, and W. Henderickx, "ANI-Gamut: Benchmarking agent reliability across the gamut of agent–network interface abstractions," in *Proc. 22nd International Conference on Network and Service Management (CNSM)*, Alcalá de Henares, Spain, Oct. 2026.

[2] S. Salsano *et al.*, "A playground for benchmarking agentic AI in network management," in *Proc. Mediterranean Artificial Intelligence and Networking Conf. (MAIN)*, 2026.

[3] J. Yang *et al.*, "SWE-agent: Agent-computer interfaces enable automated software engineering," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[4] M. Kim *et al.*, "CODESTRUCT: Code agents over structured action spaces," in *Proc. Annual Meeting of the Association for Computational Linguistics (ACL)*, 2026, arXiv:2604.05407.

[5] Y. Cui *et al.*, "A framework to evaluate llm agents for network configuration," IETF Internet-Draft, Informational draft-cui-nmrg-llm-benchmark-01, December 2025.

[6] R. Enns *et al.*, "Network configuration protocol (NETCONF)," IETF, RFC 6241, 2011.

[7] M. Bjorklund *et al.*, "Network management datastore architecture (NMDA)," IETF, RFC 8342, 2018.

[8] Cisco Systems, "Cisco network services orchestrator (NSO)," https://developer.cisco.com/docs/nso/, 2024.

[9] Nokia, "Nokia SR Linux and event-driven automation (EDA)," https://docs.eda.dev/, 2024.

[10] Juniper Networks, "Juniper Apstra: Intent-based networking for the data center," https://www.juniper.net/, 2024.

[11] Network to Code, "Nautobot: Network source of truth and golden config," https://networktocode.com/nautobot/, 2024.

[12] J. Deng *et al.*, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2009, pp. 248–255.

[13] O. Bojar *et al.*, "Findings of the 2014 workshop on statistical machine translation," in *Proc. Ninth Workshop on Statistical Machine Translation (WMT)*, 2014, pp. 12–58.

[14] S. Salsano, L. Bracciale, P. Loreti, A. Mayer, L. Proia, M. Federico, and P. Caliandro, "ANI-Gamut: playground, scenario generator and harness (release `cnsm2026`)," https://github.com/netgroup/ani-gamut, 2026, tag `cnsm2026`, commit 4d44d84. Apache-2.0 (playground) and GPL-3.0 (substrate).

[15] Z. Wang *et al.*, "A network arena for benchmarking ai agents on network troubleshooting," *arXiv preprint arXiv:2512.16381v1*, 2025.

[16] Y. Yang and H. Xu, "NetOpsBench: Open arena for NetOps in AI infrastructure," https://github.com/NetX-lab/NetOpsBench, 2026.

[17] Y. Zhou *et al.*, "NetPress: Dynamically generated LLM benchmarks for network applications," *arXiv preprint arXiv:2506.03231*, 2025, also referred to as NetArena.

[18] Y. Chen *et al.*, "AIOpsLab: A holistic framework to evaluate AI agents for enabling autonomous clouds," *arXiv preprint arXiv:2501.06706*, 2025.

[19] Y. Wang *et al.*, "Cloud-OpsBench: A reproducible benchmark for agentic root cause analysis in cloud systems," *arXiv preprint arXiv:2603.00468*, 2026.

[20] Y. Zhou *et al.*, "MeshAgent: Enabling reliable network management with large language models," in *Proc. ACM SIGMETRICS (Proc. ACM Meas. Anal. Comput. Syst.)*, 2026.

[21] S. Yu, F. Carroll, and B. L. Bentley, "The causal impact of tool affordance on safety alignment in LLM agents," 2026, proc. ICECET; arXiv:2603.20320.

[22] C. Wang *et al.*, "Netconfeval: Can llms facilitate network configuration?" *Proceedings of the ACM on Networking*, vol. 2, no. CoNEXT2, pp. 1–25, 2024.

[23] Z. Wang *et al.*, "Intent-driven network management with multi-agent LLMs: The Confucius framework," in *Proc. ACM SIGCOMM*, 2025.

TABLE VII

CONFIGURATION-STATE SCALE OF EXISTING AGENTIC BENCHMARKS, MEASURED AS HETEROGENEOUS SERVICE CLASSES AGAINST STATE PER CLASS. "EST." MARKS OUR ESTIMATE WHERE THE SOURCE IS SILENT.

Benchmark	**Service classes**	**State size per class**	**Substrate**	**Ref.**
NIKA	$\sim$5 scenario types	11–101 nodes; 54 issue types; 10^2–10^3 config items (est.)	live emul.	[15]
NetOpsBench	$O(1)$ scenarios	tens of devices, 4 scales (est.)	live emul.	[16]
NetArena/NetPress	10 node types	5,493-node graph (static); live tasks small	static/live	[17]
NetConfEval	4 task families	small topology; static translation	static	[22]
NetConfBench	40 tasks	few nodes per task; intent, initial config, tests	live emul.	[5]
MeshAgent	3 graph applications	DSL queries over a network graph	graph	[20]
AIOpsLab	2–3 microservice apps	up to 28 microservices; 1000 fault scenarios	live (K8s)	[18]
Cloud-OpsBench	K8s stack	452 fault cases, 40 root-cause types	live (K8s)	[19]
OperAID	1 app (Open5GS 5GC)	11 NFs + RAN sim; 3 fault scenarios; 7 read-only kubectl tools	live (KinD)	[38]
MicroRemed	microservice apps	microservice topologies (est.)	live	[24]
ITBench	3 IT domains	$\sim$94 scenarios (SRE/CISO/FinOps)	live	[44]
SREGym	cloud-native stack	90 SRE problems; multi-layer faults	live	[45]

[24] L. Zhang *et al.*, "MicroRemed: Benchmarking LLMs in microservices remediation," *arXiv preprint arXiv:2511.01166*, 2025.

[25] M. Bilal *et al.*, "Large language models for agentic NetOps and AIOps: Architectures, evaluation, and safety," *arXiv preprint arXiv:2605.12729*, 2026.

[26] S. K. Mani *et al.*, "Enhancing network management using code generated by large language models," in *Proc. ACM Workshop on Hot Topics in Networks (HotNets)*, 2023.

[27] K. Dzeparoska *et al.*, "LLM-based policy generation for intent-based management of applications," in *Proc. 19th Int. Conf. Network and Service Management (CNSM)*, 2023, pp. 1–7.

[28] A. El Hachimi *et al.*, "Flow-rule generation for SDN using LLMs with retry-based deployment validation," in *Proc. 21st Int. Conf. Network and Service Management (CNSM)*, 2025, pp. 1–5.

[29] A. Angi, A. Sacco, and G. Marchetto, "LLNet: An intent-driven approach to instructing softwarized network devices using a small language model," *IEEE Trans. Netw. Service Manag.*, vol. 22, no. 4, pp. 3403–3418, 2025.

[30] A. Bierman, M. Bjorklund, and K. Watsen, "RESTCONF protocol," IETF, RFC 8040, 2017.

[31] OpenConfig, "OpenConfig: Vendor-neutral, model-driven network management (gNMI/gNOI)," https://openconfig.net/, 2024.

[32] A. Clemm *et al.*, "Intent-based networking – concepts and definitions," IRTF, RFC 9315, 2022.

[33] M. Behringer *et al.*, "A reference model for autonomic networking," IETF, RFC 8993, 2021.

[34] X. Liu *et al.*, "Generic intent-driven networking paradigm with different levels of policy abstraction," *IEEE Communications Magazine*, vol. 63, no. 4, pp. 162–168, 2025.

[35] TM Forum, "Autonomous networks levels evaluation methodology," TM Forum, Introductory Guide IG1252 v1.2.0, 2024.

[36] 3GPP, "Management and orchestration; levels of autonomous network," 3GPP, Technical Specification TS 28.100, 2024.

[37] ETSI ISG ZSM, "Zero-touch network and service management (ZSM); reference architecture," ETSI, Tech. Rep. GS ZSM 002, 2019.

[38] A. Góes de Castro *et al.*, "OperAID: Benchmarking LLM agents for autonomous Kubernetes fault remediation," in *Proc. IEEE Int. Conf. Network Softwarization (NetSoft) Workshops*, 2026.

[39] J. Horn *et al.*, "Addressing recommendations for SRv6 NEXT-CSID," IETF Internet-Draft, Informational draft-horn-srv6ops-srv6addressing-01, March 2026, work in progress.

[40] L. Bracciale, P. Loreti, A. Mayer, S. Salsano, and W. Henderickx, "ANI-Gamut: hypotheses and analysis plan," https://netgroup.github.io/ani-gamut/, 2026, https://netgroup.github.io/ani-gamut/preregistration.html.

[41] G. Huston, "CIDR report: BGP routing table analysis (as131072)," https://www.cidr-report.org/, 2026, iPv4 table approx. 950k routes; accessed June 2026.

[42] Cisco Systems, "Cisco Nexus 9000 Series NX-OS verified scalability guide," https://www.cisco.com/c/en/us/td/docs/dcn/nx-os/nexus9000/, 2024, e.g., up to 1000 VRFs and 512 BGP peers per device.

[43] T. Benson, A. Akella, and D. A. Maltz, "Unraveling the complexity of network management," in *Proc. USENIX Symp. on Networked Systems Design and Implementation (NSDI)*, 2009, pp. 335–348.

[44] S. Jha *et al.*, "ITBench: Evaluating AI agents across diverse real-world IT automation tasks," in *Proc. Int. Conf. on Machine Learning (ICML)*, 2025, arXiv:2502.05352.

[45] J. Clark, Y. Su *et al.*, "SREGym: A live benchmark for AI SRE agents with high-fidelity failure scenarios," *arXiv preprint arXiv:2605.07161*, 2026.