CLOUDGYM: CAN AI AGENTS MANAGE SHARED CLOUD INFRASTRUCTURE?

Anonymous authors
Paper under double-blind review

ABSTRACT

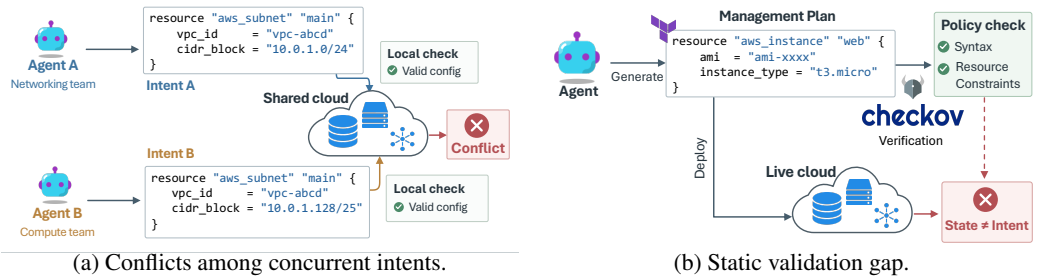
Managing cloud infrastructure requires substantial expertise and operational effort, motivating automation with AI agents. As cloud infrastructure is inherently shared, cloud management agents must coordinate with other participants whose actions can invalidate their plans or block their operations. Existing evaluations largely focus on static plans or isolated execution, leaving coordination in shared environments insufficiently tested. We present CLOUDGYM, a benchmark for evaluating cloud management agents in live, shared cloud infrastructure, with 102 cases spanning 19 AWS services and 65 resource types. Each case introduces controlled concurrent participants whose actions create incompatible requirements, ambiguous targets, or temporary execution conflicts. Semantic oracles use initial and final cloud states and the record of activated participants to check whether outcomes satisfy the task intent under a specified resolution policy. To support reliable evaluation, an automated pipeline generates these cases from existing management tasks and certifies their behavior through execution against AWS. Our evaluation shows that frontier agents perform poorly on CLOUDGYM, with the top-performing model, Opus 5.5, achieving an overall pass rate of 34.3% compared with 97.1% in isolated execution. This gap highlights coordination as a major obstacle to agentic cloud management. We open-source the CLOUDGYM dataset and evaluation framework (Anonymous Authors, 2026) to enable future research.

1 INTRODUCTION

Cloud computing has transformed IT infrastructure, with modern services relying on large collections of interconnected cloud resources (Armbrust et al., 2010). Keeping the cloud infrastructure aligned with constantly changing service demands requires continuous management (Verma et al., 2015; Hummer et al., 2013). Deploying a virtual machine, for example, involves connecting its network interface and configuring access rules so that it can communicate with other services. Cloud tenants express infrastructure intents through cloud-provider APIs and management tools, including command-line interfaces (CLIs) and infrastructure-as-code (IaC) (Artac et al., 2017), which translate intents into resource-level operations (Qiu et al., 2023). The substantial knowledge this work demands and the risk of manual errors have long motivated cloud automation to minimize human effort (Yin et al., 2011; Gunawi et al., 2016). Recent advances in AI agents are enabling *agentic cloud management*, in which agents determine how to realize and carry out high-level management intents through management tools (Yang et al., 2025a; Chen et al., 2025; Jha et al., 2025).

At scale, *cloud management agents* operate in shared cloud environments across teams with different resource domains, operational roles, and ownership boundaries. One team may repair network topology, another scale compute capacity, and another configure security policies, each using its own management interfaces. As shown in Figure 1, each party acts through observations and operations scoped to its own task, without a complete view of the tenant's shared state or ongoing activities. As a result, plans and deployments that satisfy an intent in isolation may become infeasible when another party operates in the shared cloud environment. Even if every individual operation succeeds, the combined changes may leave the cloud infrastructure in a state that fails to satisfy one or more parties' intents. Establishing success therefore requires checking the final cloud state against each parties' initial intent as well as the agreed resolution of any conflicts.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.



(a) Conflicts among concurrent intents. (b) Static validation gap.

Figure 1: Challenges in agentic cloud management.

We therefore argue that cloud management agents must be evaluated in shared cloud infrastructure, and must address both *coordination* with other parties' intents and in-flight operations, and *validation* of whether the intended outcome has been realized. While existing work has explored cloud management tasks, evaluation has largely focused on static analysis of generated configuration plans or task execution in isolated cloud environment without concurrent participants Tiwari & Gupta; Zhang et al. (2025); Yang et al. (2026); Jana et al. (2026). For example, IaC-Eval Kon et al. (2024) statically checks generated IaC programs for intent fulfillment, while NSync Yang et al. (2025b) focuses on drift reconciliation for a single management party.

CLOUDGYM bridges this gap with the first benchmark for evaluating cloud management agent in shared environment. Each task begins with an initial state and a target intent given to the evaluated agent. During execution, CLOUDGYM watches cloud state and the agent's API activity to trigger *distractors* at controlled points. Each distractor represents a party pursuing its own intent through operations that change the shared infrastructure, requiring the agent to respond to their effects with a *resolution policy*. Once the concurrent operations complete, CLOUDGYM evaluates the final cloud state against a *semantic oracle* of the required outcome. Our benchmark contains 102 cases spanning 19 AWS services and 65 resource types, certified through execution against AWS. CLOUDGYM tackles three key challenges in benchmarking cloud management agents:

Exercising realistic coordination under concurrent management intents. Evaluation in a shared cloud environment requires controlled interference; otherwise, concurrent activity can cause intermittent failures that are difficult to reproduce. The interference must express another party's plausible intent, arise at an appropriate point in the agent's execution, and present a meaningful reasoning problem. To compare agents across different coordination challenges, CLOUDGYM defines three classes of coordination problems. In a *semantic conflict*, two participants require outcomes that cannot hold simultaneously; In An *intent ambiguity*, a participant makes the intended outcome of another unclear; In an *execution conflict*, concurrent operations temporarily blocks progress toward an otherwise compatible goal.These cases require agents to determine which requirement takes precedence, clarify the intended target, or wait for an ongoing operation to complete, respectively.

Validating intent fulfillment in live cloud infrastructure. Validating outcomes in a shared cloud environment poses three challenges. First, neither a static configuration nor a final deployment snapshot alone establishes whether the agent's intent was fulfilled without violating other parties' intents. Second, validation must account for which distractors were triggered during execution, since their activation determines which requirements apply in each run. Third, validation must accept benign variations among correct deployments, making exact matching against a reference state overly restrictive. CLOUDGYM therefore uses semantic oracles over the initial and final provider states and the record of activated distractors to check intent fulfillment under the task's resolution policy.

Generating high-quality coordination tasks automatically. Generating reliable coordination tasks requires the infrastructure, concurrent intents, and success criteria to be consistent. CLOUDGYM uses a dedicated generation agent to extend screened single-intent tasks into coordination cases with executable distractors, resolution policies, and semantic oracles, with each generation stage checked for consistency. For example, live control runs on real cloud verify that the main intent can be completed and accepted by the oracle, while interference runs verify that distractors trigger, modify the infrastructure, and become observable during execution.

Finally, we conduct a detailed evaluation against a range of state-of-the-art agents using a variety of cloud management interfaces. Agents achieve 82.8%–97.1% pass rates on isolated cloud tasks, but only 18.6%–34.3% when they must coordinate with concurrent participants and obtain the required

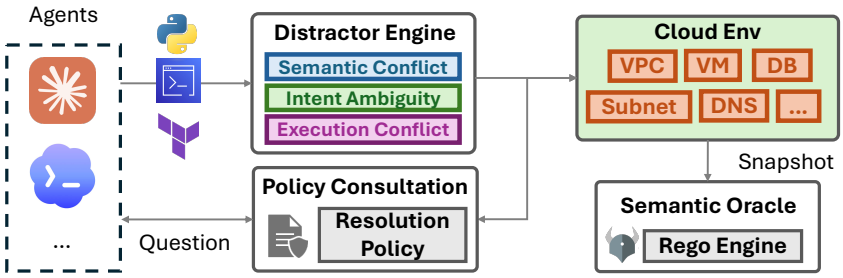


Figure 2: CLOUDGYM evaluation workflow overview.

resolution policy through consultation. Even the strongest model, Opus 5.5, drops from 97.1% to 34.3%, highlighting coordination as a major challenge for current agents.

2 CLOUDGYM

To our knowledge, CLOUDGYM is the first benchmark to exercise agents in live infrastructure with concurrent participants. As shown in Figure 2, CLOUDGYM first setup initial state in a real cloud environment and presents the main task intent to the evaluated agent. As the agent observes the environment and issues cloud operations, a distractor engine evaluates the agent's operations along with the current cloud state. Once a predefined trigger condition is satisfied, the distractor engine activates a distractor that represents another party in the shared environment with a distinct intent and introduces a controlled change to the shared cloud environment. The evaluated agent must then observe the resulting change, infer its implications, and replan/reconcile as per a resolution policy. In the end, CLOUDGYM extracts the final cloud state and applies an executable symbolic oracle written in Rego (Open Policy Agent) to determine whether the intents have been successfully realized.

2.1 PROBLEM DEFINITION

A task in CLOUDGYM defines a live cloud-management scenario in which an agent must realize a management intent despite concurrent interference. Formally, each problem is a five-tuple $P = (S_0, I, O, D, R)$ consisting of:

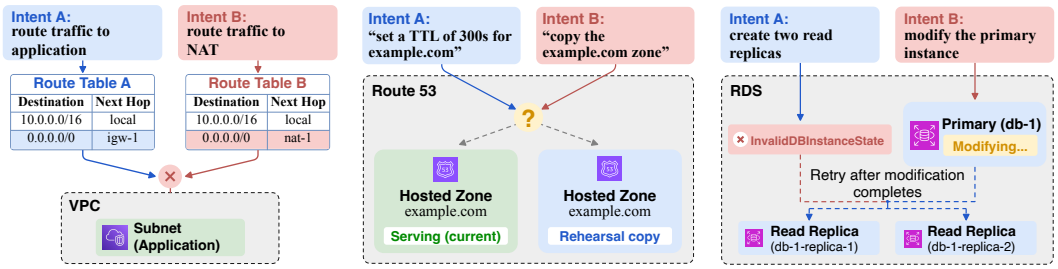
- **Initial cloud state S_0.** A configuration written in Terraform (HashiCorp, n.d.), a popular IaC-style cloud management tools, that instantiates the cloud environment from which the episode begins.
- **Intent I.** A natural-language specification of the desired change to the cloud environment.
- **Semantic oracle O.** An executable Rego policy that specifies the properties the resulting cloud state must satisfy for the intent to be considered fulfilled.
- **Distractor set D.** A set of trigger–action pairs $D = \{(t_j, a_j)\}$, where each trigger t_j specifies when concurrent participants should be introduced and each action a_j specifies the corresponding cloud operation(s).
- **Resolution policy R.** A set of natural-language rules specifying how conflicts or ambiguities should be resolved when the intended outcome is not uniquely determined by I.

For each problem, the input and expected output of the evaluated agent are defined as follows:

- **Input.** The agent receives the intent I and can observe and modify the cloud environment initialized from S_0. No additional information about the initial state is provided.
- **Expected output.** After the agent completes the task, CLOUDGYM extracts the final cloud state S_f and evaluates it using O. The agent succeeds if and only if the oracle evaluates to true.

2.2 COORDINATION TASKS

Distractor Taxonomy. Let S denote a valid cloud state and ϕ_i the desired post-condition of any intent i. We classify coordination interference into three categories based on how two concurrent intents A and B interact with each other.



(a) Semantic Conflict (b) Intent Ambiguity (c) Execution Conflict

Figure 3: Examples of each taxonomy category.

(1) *Semantic conflict (SC).* Two clearly defined intents require incompatible post-conditions. Each may be feasible alone, but no valid cloud state can satisfy both. The incompatibility may concern a resource attribute, an association, a connected set of resources, or an environment-wide constraint:

$$\nexists S : \; S \models \phi_A \wedge \phi_B.$$

Example. As shown in Figure 3a, Intent A and Intent B require the same subnet to use different route tables, while the cloud permits only one route-table association for that subnet. The two target states are therefore mutually exclusive and therefore requires a proper resolution.

(2) *Intent Ambiguity (IA).* An intent leaves some target or value to be determined from cloud state, so its concrete meaning may change as the state changes. In particular, there may exist two different valid states under which the same intent resolves differently:

$$\exists S, S' : \; S \neq S', \qquad S \models \phi_A \wedge \phi_B, \qquad S' \models \phi_A \wedge \phi_B.$$

Example. In Figure 3b, an agent is asked to set a TTL for `example.com` records. If another participant creates a second public hosted zone with the same domain during execution, the phrase no longer identifies a unique target, and the intent may resolve to different cloud states.

(3) *Execution conflict (EC).* The intents have fixed meanings and compatible post-conditions, so there exists a valid cloud state in which both can hold. Their executions may nevertheless temporarily interfere due to provider constraints: the provider may reject an operation c_A while another change c_B is in progress at t_0, yet accept the same operation at a later time t_1 when c_B completes.

$$\exists S : \; S \models \phi_A \wedge \phi_B, \qquad \neg \mathsf{Concurrent}(\mathsf{c_A}, \mathsf{c_B})$$

Example. Figure 3c shows a scenario that an agent creates read replicas of an existing database while another participant modifies the primary instance. Replica creation is rejected by the provider while the instance is in the `modifying` state, but the same request succeeds once the modification completes, and both intents remain compatible in the final state.

Runtime distraction injection. CLOUDGYM uses distractors to introduce controlled concurrent interference into a cloud task, thereby simulating a realistic cloud environment with multiple parties. Each distractor represents another participant with its own intent and actions, allowing the benchmark to test whether the agent can adapt when shared cloud state changes or when another operation overlaps with its execution. To make such interference both realistic and reproducible, CLOUDGYM realizes each distractor $D_j = (t_j, a_j)$ as an event-driven trigger–action binding.

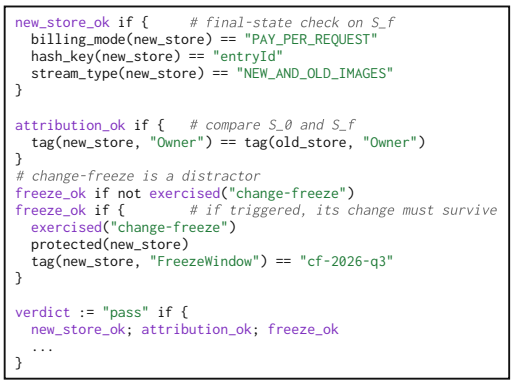
CLOUDGYM supports both state-based and API-based triggers because different conflict types become observable at different points during execution. State-based triggers condition on live cloud-state snapshots and are designed to capture coarse-grained conditions that persist in cloud state, such as the existence of a resource. API-based triggers operate on agent-issued API events and capture fine-grained execution windows that may exist only around a specific request, which is necessary for reproducing transient execution conflicts. During execution, the runtime evaluates t_j over cloud-state snapshots and agent-issued API events and fires the corresponding interference action a_j.

To condition on agent-issued API events, CLOUDGYM implements a proxy that intercepts all requests from agents to the live cloud environment. When an API-based trigger fires, the proxy may



```
{
  "AWS::DynamoDB::Table": {
    "ledger-entries-global": {
      "BillingMode": "PAY_PER_REQUEST",
      ...
      "KeySchema": [{
        "AttributeName": "entryId",
        "KeyType": "HASH"
      }],
      "StreamSpecification": {
        "StreamViewType": "NEW_AND_OLD_IMAGES"
      },
      ...
      "DeletionProtectionEnabled": true,
      "Tags": {
        "Owner": "ledger-platform",
        "FreezeWindow": "cf-2026-q3"
      }
    } ...
  } ...
}
```

(a) Final snapshot S_f.



```
new_store_ok if {     # final-state check on S_f
  billing_mode(new_store) == "PAY_PER_REQUEST"
  hash_key(new_store) == "entryId"
  stream_type(new_store) == "NEW_AND_OLD_IMAGES"
}

attribution_ok if {   # compare S_0 and S_f
  tag(new_store, "Owner") == tag(old_store, "Owner")
}
# change-freeze is a distractor
freeze_ok if not exercised("change-freeze")
freeze_ok if {        # if triggered, its change must survive
  exercised("change-freeze")
  protected(new_store)
  tag(new_store, "FreezeWindow") == "cf-2026-q3"
}

verdict := "pass" if {
  new_store_ok; attribution_ok; freeze_ok
  ...
}
```

(b) Rego oracle over $\langle S_0, S_f, D_c \rangle$.

Figure 4: Example pair of cloud-state snapshot and Rego oracle.

briefly hold the matching agent request while launching the distractor, then release it once the distractor has entered the required execution phase. This further provides deterministic control over short-lived races. Together with live cloud-state snapshots, the proxy also gives CLOUDGYM visibility into both the operations being issued and their resulting state changes.

Resolution policy and consultation. In many cases, the intent I alone may not determine a unique correct resolution. Each problem therefore includes a resolution policy R, which specifies the additional rules needed to determine the intended outcome. For example, in a *semantic conflict* case, the policy specifies which of two conflicting intents should be abandoned. In an *intent ambiguity* case, it specifies how to disambiguate among multiple plausible interpretations.

In real cloud-management workflows, discovering such resolution policies is often part of the task itself and requires careful coordination among teams. An agent should therefore not be given all resolution information beforehand. Instead, it must recognize when the observed cloud state is insufficient to resolve the task, determine what information is missing, identify the appropriate cloud participant to consult, obtain the relevant resolution policy, and incorporate that policy into its subsequent actions. To simulate this interaction while keeping policy discovery deterministic, CLOUDGYM provides a *consultation mechanism* to the agent through an MCP tool (Model Context Protocol, 2025). The agent can issue a specific query associated with an observed cloud resource. Resources created by distractors carry distinctive fingerprints, allowing CLOUDGYM to identify which concurrent participant the agent is referring to and return the resolution policy associated with that distractor's intent. This design ensures that policy discovery remains grounded in the agent's observations of the live cloud environment rather than hidden coordination information directly.

2.3 INTENT VALIDATION

CLOUDGYM evaluates an agent against the materialized live cloud state. For each case c, CLOUDGYM captures cloud-state snapshots over a predefined set of relevant resource types $\mathcal{T}_c$. After deploying the initial configuration, CLOUDGYM records an initial snapshot S_0. During execution, it captures intermediate observations. After the agent terminates and all activated distractors complete, the runtime records the final snapshot S_f.

Formally, a snapshot at time t is a typed resource map:

$$S_t = \{\tau \mapsto \{i \mapsto P_{t,\tau,i}\} \mid \tau \in \mathcal{T}_c\},$$

where τ denotes a provider resource type, i is a provider-side resource identifier, and $P_{t,\tau,i}$ is the corresponding property object. Because the snapshot is captured from the live cloud state rather than Terraform state files or other interface-specific representations, it provides a more authoritative view of the deployed infrastructure and naturally covers resources created through different modalities, including SDKs, CLIs, and other management interfaces (Yang et al., 2025a).

Since a single intent may have multiple correct realizations, validation should not rely on exact state matching. Instead, after capturing the final snapshot, CLOUDGYM evaluates task completion using a case-specific Rego oracle, as illustrated in Figure 4. The oracle takes the tuple $\langle S_0, S_f, D_c \rangle$, where

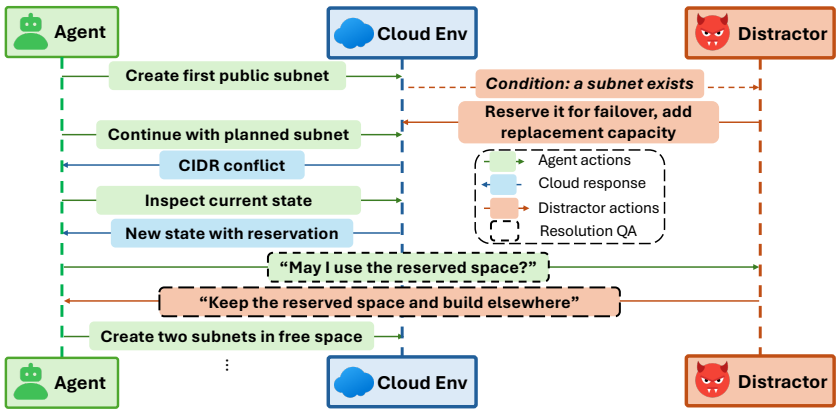


Figure 5: An example *semantic conflict* case with simplification.

D_c records the set of activated distractors in the case. The oracle specifies semantic conditions for successful intent completion as predicates over these inputs, including resource existence, cardinality, attributes , etc. Distractor-dependent postconditions are checked only when the corresponding distractors are triggered, allowing the oracle to account for actual interferences that occurred during execution. The initial state snapshot S_0 is also used to check consistency with S_f, ensuring that the agent preserves unaffected state and does not introduce unnecessary changes. Consequently, this design accepts multiple final states as long as they satisfy the intended semantic conditions.

2.4 An End-to-End Example

Figure 5 shows an end-to-end example from a real CloudGym run. An agent intends to add at least two public subnets to an existing VPC and connecting them to the public Internet. During execution, another engineer concurrently reserves an empty subnet as failover headroom and allocates replacement capacity within the same address space. This reservation happens to target the first subnet created by the agent. When the agent proceeds with its original plan, it encounters a CIDR conflict and observes that the subnet has been reserved for failover. Because the resolution policy is not initially available, the agent asks the responsible engineer whether the reserved space may be reused. The engineer instructs it to preserve the reservation and use free address space instead. The agent then replans, creates two public subnets in unallocated ranges, and connects them through the existing shared egress path. This example illustrates how CloudGym injects a concurrent intent in response to execution progress, exposes its effects through live cloud states, and evaluates whether the agent can detect the resulting conflict, obtain the missing resolution policy, and replan without violating the other participant's intent.

2.5 Benchmark Generation Workflow

Figure 6 summarizes how CloudGym constructs concurrent cloud-management benchmark cases. CloudGym begins from a corpus of single-intent infrastructure tasks. In the current implementation, we use IaC-Eval Kon et al. (2024) as one such seed source, but the generation pipeline is not specific to IaC-Eval and can operate on other task collections. For each seed, CloudGym first performs eligibility screening to exclude ones that are unsafe or impractical for repeated live execution, such as configurations involving unsupported resources, excessive provisioning cost or latency, or resources outside the benchmark's state-extraction coverage. This stage preserves the service and resource diversity of the source dataset while restricting generation to cases that can be exercised repeatedly in a controlled cloud sandbox account.

For each eligible seed, a generation agent synthesizes a CloudGym case through a sequence of gated steps. The reference configuration is first mutated into a pair of an initial state and an intended target state, defining the main participant's state transition. The generator then constructs concurrent intents according to the interference taxonomy, together with their resolution policy and executable distractors. Finally, it produces the state-based semantic oracle described in the previous section and a suite of synthetic fixtures covering both successful resolution and representative negative states.

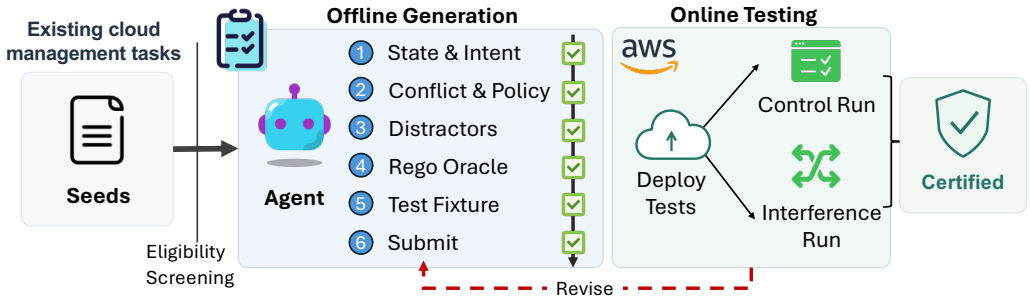


Figure 6: Benchmark generation flow.

Each stage is checked before generation can advance, preventing internally inconsistent distractors, triggers, or oracles from reaching live-cloud execution.

Passing synthetic checks alone does not establish that a case behaves as intended under real provider semantics (Qiu et al., 2024). Therefore, every candidate is certified against the real cloud provider before entering the benchmark. Certification first repeats the offline checks, then deploys the initial configuration and validates the oracle and trigger predicates against the resulting real cloud state. Then, A control run without interference verifies that the main intent can be completed and accepted by the oracle. Only after that, an interference run verifies that the intended distractors can be triggered, successfully modify the live environment, and become observable during agent execution. The final oracle outcome under the interference run is recorded rather than used as a certification gate, since resolving that interference is precisely the capability the benchmark is intended to measure. Failed certification returns execution evidence to the generator for revision and only candidates that pass the full certification loop are published as benchmark cases.

3 RESULTS

Experiment settings. We evaluate Claude Code and Codex on 102 CLOUDGYM cases using five model backends: Claude Haiku 4.5, Claude Opus 5.5, Claude Fable 5, GPT-5.5, and GPT-5.6 Sol. For each case and agent–model configuration, we run 2 independent trials per experimental condition. Our primary experiment uses the default hybrid tool modality and compares three conditions:

(i) **Control:** no distractors, isolated single task.
(ii) **Prompted policy:** distractors enabled, with resolution policies in the agent prompt.
(iii) **Consulted policy:** CLOUDGYM default policy, distractors enabled, with resolution policies withheld from the agent prompt and made available only through gated consultation.

We further evaluate how cloud management modality affects agent performance by comparing SDK, IaC, CLI, and hybrid interfaces under (iii) consulted policy. For SDK, IaC, and CLI runs, agents are instructed to use Python scripts with boto3, Terraform, and command-line tools, respectively. For hybrid runs, agents are free to choose among these modalities.

Evaluation metric. We evaluate agent performance using five primary metrics:

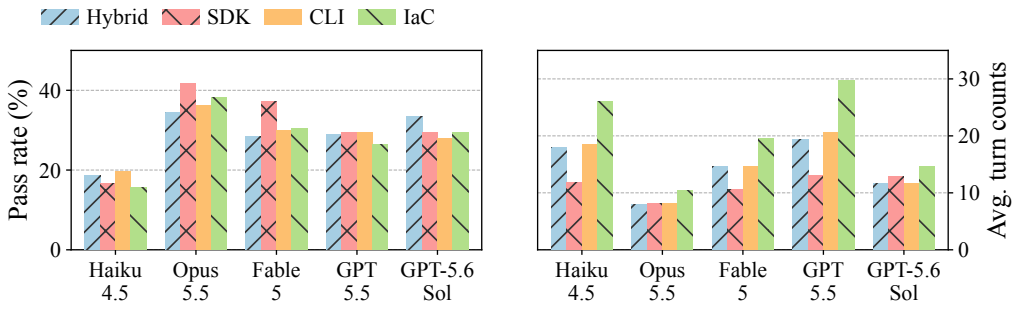
- **Pass rate:** the fraction of runs in which the agent successfully completes the task according to the symbolic oracle.
- **Turn count:** the number of agent interaction turns taken during each trial.
- **Token usage:** the total number of model tokens consumed during the agent run.
- **Inference cost:** the monetary cost of model inference during the agent run.
- **Relative slowdown:** the ratio of end-to-end completion time under interference of distractors to that of the corresponding conflict-free control run.

3.1 OVERALL RESULTS

CLOUDGYM exposes a substantial gap between completing a cloud-management task in isolation and resolving it in the presence of another participant. As shown in Table 1, under the control condition (i) without concurrent participants, agents achieve overall pass rates from 82.8% to 97.1%. With concurrent participant enabled, performance drops substantially. Under the condition (ii) prompted

Table 1: Results under the hybrid modality. Tokens, turns, and costs are averaged per case.

Condition	Model	Pass rate (%)	SC (%)	IA (%)	EC (%)	Tokens (K)	Turns	Inference cost ($)
(i)	Haiku 4.5	82.8	85.9	77.6	86.1	461.4	14.3	0.09
	Opus 5.5	97.1	98.9	97.4	91.7	207.3	6.1	0.12
	Fable 5	96.1	95.7	94.7	100.0	279.7	11.4	0.66
	GPT-5.5	96.6	97.8	93.4	100.0	200.3	16.2	0.27
	GPT-5.6 Sol	96.6	98.9	93.4	97.2	189.5	9.2	0.21
(ii)	Haiku 4.5	19.6	5.4	13.2	69.4	540.3	17.6	0.11
	Opus 5.5	55.4	40.2	57.9	88.9	255.1	7.3	0.16
	Fable 5	52.0	38.0	50.0	91.7	360.0	13.3	0.91
	GPT-5.5	38.7	19.6	38.2	88.9	290.3	20.6	0.38
	GPT-5.6 Sol	45.6	27.2	47.4	88.9	234.5	10.8	0.25
(iii)	Haiku 4.5	18.6	4.3	9.2	75.0	592.9	18.0	0.11
	Opus 5.5	34.3	17.4	32.9	80.6	282.7	8.0	0.16
	Fable 5	28.4	4.3	30.3	86.1	406.8	14.6	0.98
	GPT-5.5	28.9	8.7	26.3	86.1	273.6	19.4	0.35
	GPT-5.6 Sol	33.3	17.4	25.0	91.7	264.4	11.7	0.27



(a) Pass rates.

(b) Average turn counts.

Figure 7: Effects of cloud management modality on agent performance and efficiency.

policy, the overall pass rates fall to 19.6%–55.4%. Under the default and most realistic condition (iii) consulted policy, the range further degrades to 18.6%–34.3%.

Model differences. The degradation is consistent across the evaluated models, but its magnitude varies across models. Claude Opus 5.5 achieves the highest overall pass rates under condition (iii). Fable 5 consults at a similar frequency to Opus 5.5 but achieves a lower pass rate and shows a slightly larger decline from condition (ii), suggesting that consultation alone is insufficient for successful execution. GPT-5.6 Sol consults more frequently than GPT-5.5 and performs better overall under condition (iii). Haiku 4.5 has the lowest overall pass rate and consults in only 11.3% of runs. Qualitative trace inspection shows instances where it notices state changes but proceeds to overwrite them without seeking resolution guidance.

Cost. Concurrent interference also increases execution and inference costs. Across all evaluated models, both conditions (ii) and (iii) generally incur high token usage, turn counts, and inference costs than the control condition. However, higher resource consumption does not consistently correspond to higher pass rates. Under condition (iii), Haiku 4.5 consumes the most token but achieved the lowest overall pass rates, while Fable 5 incurs the highest inference cost, averaging $0.98 per case, but achieves a lower passing rate than Opus 5.5. Opus 5.5 achieves the highest overall pass rates with the fewest turns, averaging 8.0 turns and $0.16 per case.

3.2 Results by coordination tasks and management modalities

Task differences. Partitioning results by case taxonomy, *EC* cases retain higher pass rates than *SC* and *IA* cases under interference, ranging from 75.0% to 91.7% under (iii). This relative robustness is consistent with transient execution conflicts being recoverable through waiting and retrying, whereas *SC* and *IA* cases require reconciling conflicting requirements or resolving ambiguous intent.

Successful *EC* recovery nevertheless incurs additional time. Among *EC* case–trial pairs passing both (i) and (iii), median slowdown relative to control is 1.59× for Haiku, 1.67× for Opus, 1.41× for Fable, 1.21× for GPT-5.5, and 1.25× for Sol. Opus traces show a cautious approach to concurrent changes: waiting for ongoing operations to complete, verifying resource state, and retrying blocked updates, which contributes to this overhead.

Modality differences. Figure 7 shows that cloud-management modality affects both task success and interaction efficiency under condition (iii). Across all cases, SDK increases the pooled pass rate from 28.7% under the hybrid modality to 30.9%, while reducing the average turn count from 14.3 to 11.3. Our execution traces show that agents group multiple cloud reads and writes in a single SDK script, reducing the number of interaction turns. IaC on the other hand achieves 28.0% pass rate but requires the most turns for inspecting the initial cloud states, averaging 20.1. CLI remains close to the hybrid baseline, with a pass rate of 28.6% and an average of 14.7 turns.

These effects vary across models. Opus 5.5 improves from 34.3% under the hybrid modality to 41.7% with SDK and 38.2% with IaC, whereas GPT-5.5 improves slightly from 28.9% to 29.4% with SDK, and GPT-5.6 Sol declines from 33.3% to 29.4%. IaC requires the most turns for every evaluated model, but does not consistently improve task success. Overall, specifying a modality can improve aggregate performance, but the gains depend on the model and do not eliminate the difficulty of resolving concurrent intents.

4 Related Works

Cloud infrastructure benchmarks. Existing cloud-infrastructure benchmarks primarily evaluate LLMs on generating or modifying infrastructure-as-code (IaC). IaC-Eval Kon et al. (2024) evaluates natural-language-to-Terraform generation with executable intent specifications, while SWE-InfraBench Tarasova et al. (2026) evaluates modifications to existing AWS CDK projects using executable tests. Other benchmarks extend this setting to multi-format IaC mutation, deployability, ambiguity resolution, security, and infrastructure drift Davidson et al. (2025); Zhang et al. (2025); Yang et al. (2026); Tiwari & Gupta; Yang et al. (2025b). Overall, these benchmarks remain centered on configuration artifacts or predefined infrastructure changes, rather than end-to-end intent realization by agents operating through heterogeneous interfaces in a shared live cloud.

Site reliability engineering (SRE) benchmarks. Similar to CloudGym, some recent SRE benchmarks such as AIOpsLab Chen et al. (2025), ITBench Jha et al. (2025), and SREGym Clark et al. (2026) also evaluate AI agents in live system environments. These benchmarks deploys cloud applications, injects faults and workloads, exposes telemetry, and evaluates SRE agents across the incident life cycle. CloudGym focuses on cloud management, not SRE; it addresses a fundamentally different problem from failure diagnosis and recovery. Note that existing SRE benchmarks only consider single SRE participant, while in reality there could also be concurrent parties. CloudGym inherently captures such coordination challenge and can be extended to include SRE agents.

5 Discussion and Conclusion

We presented CloudGym, a benchmark for cloud management agents in shared cloud environment. Our evaluation shows that agents that perform strongly on isolated cloud tasks degrade substantially when other participants act on the same infrastructure, highlighting coordination as a central challenge for autonomous cloud management.

We envision several extensions that would further broaden the scope of CloudGym. Extending its current AWS setting to multiple cloud providers would capture dependencies and operations that span provider boundaries. Moving beyond infrastructure management toward cloud application management would bring application-level goals, service dependencies, and runtime behavior into evaluation. Supporting multiple evaluated agents would enable the study of inter-agent communication and collaborative conflict resolution. Building on the current seed-based generation pipeline, future work could also generate complete benchmark instances directly from high-level management objectives, including the initial environment, concurrent intents, and evaluation criteria.

AI USE STATEMENT

In this work, we used generative AI tools to assist with software development, generate synthetic benchmark cases for CLOUDGYM, and assist with manuscript writing and language editing. The authors manually reviewed the AI-assisted manuscript content and tested the relevant code. We validated the generated benchmark cases through staged consistency checks, synthetic tests, and live AWS certification runs (§2.5). The authors take full responsibility for the final content of this work, including the text, claims, code, and benchmark artifacts produced with the aid of generative AI.

REFERENCES

Anonymous Authors. CloudGym: Anonymous repository. https://anonymous.4open.science/r/iclr-2027-cloudbench-4475/, 2026.

Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, April 2010.

Matej Artac, Tadej Borovsak, Elisabetta Di Nitto, Michele Guerriero, and Damian Andrew Tamburri. DevOps: Introducing Infrastructure-as-Code. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pp. 497–498. IEEE, 2017.

Yinfang Chen, Manish Shetty, Gagan Somashekar, Minghua Ma, Yogesh Simmhan, Jonathan Mace, Chetan Bansal, Rujia Wang, and Saravan Rajmohan. Aiopslab: A holistic framework to evaluate ai agents for enabling autonomous clouds. *Proceedings of Machine Learning and Systems*, 7, 2025.

Jackson Clark, Yiming Su, Saad Mohammad Rafid Pial, Yifang Tian, Lily Gniedziejko, Hans-Arno Jacobsen, Yinfang Chen, and Tianyin Xu. Sregym: A live benchmark for ai sre agents with high-fidelity failure scenarios. *arXiv preprint arXiv:2605.07161*, 2026.

Sam Davidson, Li Sun, Bhavana Bhasker, Laurent Callot, and Anoop Deoras. Multi-iac-eval: Benchmarking cloud infrastructure as code across multiple formats. *arXiv preprint arXiv:2509.05303*, 2025.

Haryadi S. Gunawi, Mingzhe Hao, Riza O. Suminto, Agung Laksono, Anang D. Satria, Jeffry Adityatama, and Kurnia J. Eliazar. Why does the cloud stop computing? lessons from hundreds of service outages. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*, SoCC '16, pp. 1–16. Association for Computing Machinery, 2016.

HashiCorp. What is Terraform? https://developer.hashicorp.com/terraform/intro, n.d. Accessed 2026-09-25.

Waldemar Hummer, Florian Rosenberg, Fábio Oliveira, and Tamar Eilam. Testing idempotence for infrastructure as code. In *Middleware 2013*, volume 8275 of *Lecture Notes in Computer Science*, pp. 368–388. Springer Berlin Heidelberg, 2013.

Prithwish Jana, Sam Davidson, Bhavana Bhasker, Andrey Kan, Anoop Deoras, and Laurent Callot. TerraFormer: Automated infrastructure-as-code with LLMs fine-tuned via policy-guided verifier feedback. In *Proceedings of the IEEE/ACM 48th International Conference on Software Engineering: Software Engineering in Practice*, pp. 578–589. ACM, 2026.

Saurabh Jha, Rohan Arora, Yuji Watanabe, Takumi Yanagawa, Yinfang Chen, Jackson Clark, Bhavya Bhavya, Mudit Verma, Harshit Kumar, Hirokuni Kitahara, et al. Itbench: Evaluating ai agents across diverse real-world it automation tasks. *arXiv preprint arXiv:2502.05352*, 2025.

Patrick T Kon, Jiachen Liu, Yiming Qiu, Weijun Fan, Ting He, Lei Lin, Haoran Zhang, Owen M Park, George S Elengikal, Yuxin Kang, et al. Iac-eval: A code generation benchmark for cloud infrastructure-as-code programs. *Advances in Neural Information Processing Systems*, 37: 134488–134506, 2024.

Model Context Protocol. Tools: Model Context Protocol specification. https://modelcontextprotocol.io/specification/2025-11-25/server/tools, 2025. Specification version 2025-11-25. Accessed 2026-09-25.

Open Policy Agent. Policy language. https://www.openpolicyagent.org/docs/policy-language. Accessed: 2026-09-26.

Yiming Qiu, Patrick Tser Jern Kon, Jiarong Xing, Yibo Huang, Hongyi Liu, Xinyu Wang, Peng Huang, Mosharaf Chowdhury, and Ang Chen. Simplifying cloud management with cloudless computing. In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*, HotNets '23, pp. 95–101. Association for Computing Machinery, 2023.

Yiming Qiu, Patrick Tser Jern Kon, Ryan Beckett, and Ang Chen. Unearthing semantic checks for cloud infrastructure-as-code programs. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, SOSP '24, pp. 574–589. Association for Computing Machinery, 2024.

Natalia Tarasova, Enrique Balp-Straffon, Aleksei Iancheruk, Yevhenii Sielskyi, Nikita Kozodoi, Liam H Byrne, Jack Butler, Dayuan Jiang, Marcin Czelej, Andrew Ang, et al. Swe-infrabench: Evaluating language models on cloud infrastructure code. *arXiv preprint arXiv:2606.05249*, 2026.

Tarun Tiwari and Radhika Gupta. Iacbench: How reliably do large language models generate secure infrastructure-as-code? a multi-language benchmark and empirical study of frontier models. *A Multi-Language Benchmark and Empirical Study of Frontier Models*.

Abhishek Verma, Luis Pedrosa, Madhukar R. Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at Google with Borg. In *Proceedings of the Tenth European Conference on Computer Systems*, EuroSys '15, pp. 1–17. Association for Computing Machinery, April 2015.

Zhenning Yang, Archit Bhatnagar, Yiming Qiu, Tongyuan Miao, Patrick Tser Jern Kon, Yunming Xiao, Yibo Huang, Martin Casado, and Ang Chen. Cloud infrastructure management in the age of AI agents. *ACM SIGOPS Operating Systems Review*, 59(1):1–8, August 2025a.

Zhenning Yang, Hui Guan, Victor Nicolet, Brandon Paulsen, Joey Dodds, Daniel Kroening, and Ang Chen. Automated cloud infrastructure-as-code reconciliation with ai agents. *arXiv preprint arXiv:2510.20211*, 2025b.

Zhenning Yang, Kaden Gruizenga, Tongyuan Miao, Patrick Tser Jern Kon, Hui Guan, and Ang Chen. Ambig-iac: Multi-level disambiguation for interactive cloud infrastructure-as-code synthesis. *arXiv preprint arXiv:2604.02382*, 2026.

Zuoning Yin, Xiao Ma, Jing Zheng, Yuanyuan Zhou, Lakshmi N. Bairavasundaram, and Shankar Pasupathy. An empirical study on configuration errors in commercial and open source systems. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pp. 159–172. Association for Computing Machinery, 2011.

Tianyi Zhang, Shidong Pan, Zejun Zhang, Zhenchang Xing, and Xiaoyu Sun. Deployability-centric infrastructure-as-code generation: An llm-based iterative framework. *arXiv e-prints*, pp. arXiv–2506, 2025.