Automated Cloud Infrastructure-as-Code Reconciliation with AI Agents

ZHENNING YANG, University of Michigan, USA
HUI GUAN, Amazon Web Services, USA
VICTOR NICOLET, Amazon Web Services, USA
BRANDON PAULSEN, Amazon Web Services, USA
JOEY DODDS, Amazon Web Services, USA
DANIEL KROENING, Amazon Web Services, USA
ANG CHEN, University of Michigan, USA

Cloud infrastructure is managed through a mix of interfaces—traditionally, cloud consoles, command-line interfaces (CLI), and SDKs are the tools of choice. Recently, Infrastructure-as-Code/IaC frameworks (e.g., Terraform) have quickly gained popularity. Unlike conventional tools, IaC frameworks encode the infrastructure in a "source-of-truth" configuration. They are capable of automatically carrying out modifications to the cloud—deploying, updating, or destroying resources—to bring the actual infrastructure into alignment with the IaC configuration. When IaC frameworks are used together with consoles, CLI, or SDKs, IaC is unaware of changes through these non-IaC interfaces, and the IaC configuration no longer captures the intended state. This is called *infrastructure drift*. IaC frameworks will revert non-IaC changes based on the outdated IaC configuration, leading to misconfigurations or failures.

We propose NSync, an automated system for *IaC reconciliation*, which aims to propagate out-of-band changes back to the IaC program in the form of an update. Our key insight is that infrastructure changes via IaC, consoles, CLI, or SDK eventually all occur via cloud API invocations—the lowest layer for cloud management operations. Hence, NSync gleans insights from API traces to detect drift (i.e., non-IaC changes) and reconcile it (i.e., update the IaC configuration to capture the changes). This is a challenging task—identifying the intended change from low-level, noisy API traces is not easy; moreover, because of the criticality of cloud infrastructure, NSync cannot directly test the synthesized updates in a live environment. NSync addresses these challenges using an agentic design. It infers high-level infrastructure change intent from cloud API sequences with the help of LLMs, and synthesizes targeted IaC updates using domain-specific context management with customized agent tooling; it further maintains an evolving knowledge base of past successful reconciliation runs, reusing prior insights to achieve higher accuracy on future tasks. In addition to system design, we contribute a novel evaluation pipeline for injecting drift to cloud infrastructure and assessing reconciliation attempts, by sourcing from authoritative cloud operation examples and transplating them into an IaC-centric framework. Experiments across five real-world Terraform projects and 372 drift scenarios show that NSync outperforms the baseline both in terms of accuracy (from 0.71 to 0.97 pass@3) and token efficiency (1.47× improvement).

Additional Key Words and Phrases: Cloud Computing, Infrastructure-as-Code, AI Agents

1 Introduction

Cloud management—creating, maintaining, and updating cloud resources—is a crucial task. Management tasks take place through a mix of approaches. Infrastructure-as-Code (IaC) frameworks are gaining popularity: Terraform [23] leads the market, and similar tools like Pulumi [35] and OpenTofu [32] are quickly rising. These frameworks provide declarative high level specifications of the infrastructure in readable and reusable IaC configurations. This reliable "source of truth"

Authors' Contact Information: Zhenning Yang, znyang@umich.edu, University of Michigan, USA; Hui Guan, huiguan@amazon.com, Amazon Web Services, USA; Victor Nicolet, victornl@amazon.com, Amazon Web Services, USA; Brandon Paulsen, bpaulse@amazon.com, Amazon Web Services, USA; Joey Dodds, jldodds@amazon.com, Amazon Web Services, USA; Daniel Kroening, dkr@amazon.co.uk, Amazon Web Services, USA; Ang Chen, chenang@umich.edu, University of Michigan, USA.

offers DevOps teams have a centralized view of resources, ensures reproducibility, and enables code maintenance through version control. In theory IaC frameworks should complement traditional imperative management methods such as cloud consoles, command-line interfaces (CLI), and SDKs; these imperative approaches perform direct mutative actions on infrastructure and enable quick scripting and issue remediation, but do not offer the benefits of a declarative interface [54]. DevOps teams would like the benefits of both: the abstract declarative view of IaC with the easily programmable, mutative power of imperative modifications.

Unfortunately, mixing IaC with imperative interfaces introduces a critical problem: *infrastructure drift*, where the live infrastructure state diverges from the IaC configuration. Drift arises when infrastructure provisioned by an IaC framework is modified via other interfaces. An operator does this because imperative changes are easier than modifying declarative descriptions. For example, it's easier to call a single API that adds a tag to an instance than it is to find the module that describes an instance and make the modification in the appropriate place (§2.2). Since IaC frameworks regard their configurations as the canonical source of truth, they will overwrite out-of-band modifications in the next IaC update. This can lead to serious consequences such as downtime, compliance issues, and financial penalties [38]. We propose to provide the best of both declarative and imperative worlds by automatically incorporating out-of-band changes into the IaC configuration to consolidate the central view. We call this the *IaC reconciliation* problem.

Currently, addressing the IaC reconciliation problem relies on either manual intervention or commercial tools. With manual intervention, DevOps engineers need to inspect the live infrastructure, identify discrepancies, and retrofit updates into existing IaC configurations [22]. This manual process is challenging, slow, and error-prone. While IaC frameworks can detect changes to resources they already manage (e.g., a non-IaC tool modifying an IaC-managed VM attribute), they cannot automatically discover new resources (e.g., a new subnet created via non-IaC tools). This leaves DevOps engineers to identify these unmanaged resources themselves: this is not scalable and may overlook subtle yet critical changes, leave dangling resources, and can eventually cause failed deployments and misconfigurations. Even if DevOps engineers successfully identify all changes, they must then translate the changes into valid IaC updates. This not only requires understanding of both the structure and logic of the existing codebase, but also mastering the correspondence between deployed cloud resources and their declarative IaC representations.

Several commercial tools have emerged to mitigate the problem. Tools like Firefly [17] and Env0 [16] attempt to maintain a resource inventory by frequently scanning the entire cloud infrastructure. While this approach can, in theory, discover unmanaged resources, keeping the inventory accurate and up to date demands constant API calls across all regions and resource types—incurring significant operational overhead, risking API rate-limit violations, and introducing latency in drift detection. Other tools like Spacelift [41] focus on managing drifts within existing IaC-managed infrastructure but cannot handle resources created outside of IaC. Moreover, these commercial solutions are not free, introducing significant operational costs to many organizations.

We propose NSync, the first automated system for IaC reconciliation. The key insight behind our approach is to *operate on the cloud APIs* that both IaC and imperative tools rely on. RESTful APIs are the lowest-level abstraction for cloud management, and all higher-level modalities (IaC, CLI, SDK, consoles) invoke these APIs to perform tasks [54]. Furthermore, cloud providers offer API monitoring services, such as AWS CloudTrail [4] and Azure Activity Logs [31]. NSync uses this central vantage point to capture complete traces of all infrastructure changes, IaC or imperative. It jointly analyzes API traces and the original IaC configuration in order to generate code patches that reconcile the drift, while preserving the IaC configuration's structure and abstractions.

NSync casts IaC reconciliation as a *program repair* task, but the domain of cloud management presents novel challenges. In traditional program repair, typically a specification or a set of test

cases are given as explicit instructions for the repair. However, in our context, the repair instructions are implicitly captured in the API traces. The repair task needs to first perform *intent identification*, summarizing long and noisy cloud API traces into a concise description of the change. After identifying the intent, the *patch generation* step is another departure from conventional program repair. Due to the criticality of cloud operations, NSync cannot push a synthesized patch into a live deployment to test correctness. Rather, it has to perform sophisticated static evaluations, using a set of IaC-native tools to assess the match. This evaluation requires nuanced reasoning across both APIs and IaC, which is difficult to capture symbolically. To this end, we adopt an agentic approach built on large language models (LLMs), enabling the system to interpret opaque cloud state and reason about IaC structure and style. Further, the agent can continuously learn from experience by retaining knowledge from successful reconciliation runs and applying these insights to future tasks.

Besides system design, another contribution we make in this paper is a novel evaluation framework for testing IaC reconciliation. We source from realistic changes that are typical for DevOps tasks, but transform these changes using an IaC-based drift injection method to obtain ground truths. We have applied this methodology to generate 372 drift scenarios across five complex, real-world Terraform projects, ranging from tens to over a thousand resources.

We have built NSync in an agentic framework and will release it in open source. Our evaluation shows that NSync can successfully reconcile the IaC codebase, achieving 0.95 pass@3 accuracy, outperforming the off-the-shelf Claude agent (0.71) while being 1.72× more token-efficient. Continuous learning further improves the accuracy to 0.97 pass@3, while remaining 1.47× more efficient than baseline solutions. Under the stricter pass@1 metric, NSync achieves an average accuracy of 0.80, outperforming the baseline (0.49). To summarize, this paper makes the following key contributions:

- We present a new task called *IaC reconciliation*, and devise a novel solution that gleans insights from cloud API traces and casts this task as a program repair problem.
- We develop NSync, an agentic system that reconciles out-of-band infrastructure drift from API traces, substantially outperforming the baseline. We will release our system in open source.
- We propose an evaluation pipeline for generating realistic, assessable drift scenarios, and curate the first IaC reconciliation dataset with 372 validated cases across five diverse Terraform projects.
- We perform a detailed empirical study to assess the effectiveness and robustness of our system across diverse drift scenarios.

In the rest of this paper, we describe these contributions in detail.

2 Motivation

IaC frameworks operate on a declarative view of the resources in IaC configurations. These configurations which are compiled to cloud APIs that eventually modify the infrastructure. Our work focuses on Terraform, which is the dominant IaC solution with 62% market share as of 2025 [17]. In this section, we explain how this IaC framework works, describe why engineers use a mix of interfaces, how drifts happen, and motivate the IaC reconciliation problem. Finally, we describe the challenges and solutions in automated IaC reconciliation in our NSync design.

2.1 Background: IaC and Terraform

Cloud infrastructure has three forms of representation. (1) In Terraform, DevOps engineers write *configuration files* (`.tf`), which is the *declarative state* that defines the desired infrastructure. Terraform codebases are often organized in the same way as large code projects, with subdirectories and multiple configuration files—e.g., each subdirectory defining a set of related resources. (2) When Terraform compiles and deploys an IaC program to the cloud, it also produces a *local state* file (`.tfstate`), which records the IaC-managed resources (e.g., compute instances) and their current attributes as found in the cloud (e.g., machine image, ID). This is the *runtime state from the IaC point*

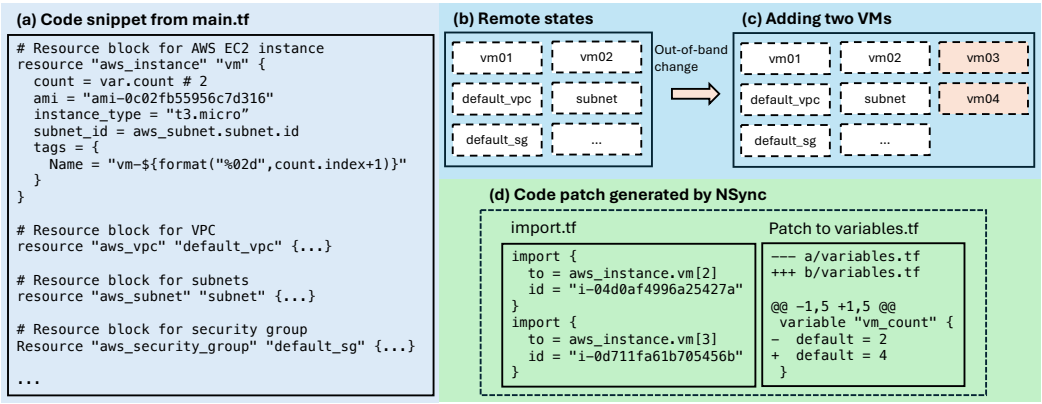

Fig. 1. A concrete IaC example written in Terraform. (a) Code snippets of a Terraform configuration file; (b) Cloud infrastructure provisioned by the configuration files; (c) Cloud infrastructure after out-of-band change that adds two VMs; (d) Code patch generated by NSync

of view, derived from the declarative state. (3) Finally, the cloud does not have a central view of the infrastructure state, but exposes resource-specific APIs to retrieve the *remote state* per resource.

A typical cloud operation workflow is as follows. First, DevOps engineers invoke `terraform plan` on a Terrraform workspace. This will generate a set of actions that, if performed, will modify the cloud infrastructure to the state encoded in the IaC program. The `terraform apply` command executes the actions based on the plan, modifying the cloud's remote state. While IaC as "source-of-truth" is one of its most valued properties, it assumes that the infrastructure is entirely managed by IaC. Consider the example in Figure 1(a), which gives a Terraform configuration snippet (`main.tf`)—the declarative state. This configuration provisions a number of VM instances specified by `var.count`, along with other networking components such as a Virtual Private Cloud (VPC), subnets, and a security group (SG) for the VM instances. Figure 1(b) illustrates the cloud's remote state after deployment. Unlike IaC, the cloud lacks a central state file; each resource exposes its own state, yielding a decentralized and fragmented view. In order to construct the remote state, one has to use state retrieval APIs at the resource level—i.e., one has to know that a certain VM exists before calling state retrieval APIs on that VM.

2.2 Problem: Infrastructure Drift and IaC Reconciliation

Operations are easier if DevOps engineers only used IaC frameworks to manage their infrastructure. However, in real-world operations, they often make *valid* changes outside the IaC workflow via consoles, CLIs, or SDKs. Our interviews with a major cloud provider reveal several representative situations where engineers bypass IaC workflows, which can introduce drift. *(1) Incident response:* when troubleshooting production issues, engineers need to apply fixes immediately. To act quickly, engineers bypass IaC and use faster API-driven tools such as AWS Systems Manager Automation [5] or Azure Sentinel Playbooks [29]. Drift arises when these emergency changes are never synchronized back into the IaC codebase. *(2) Performance tuning:* engineers frequently experiment with configuration parameters (e.g., scaling factors). While such modifications can be performed with IaC, doing so requires writing and deploying code for every trial, which is far more cumbersome than clicking a button in a dashboard or running a single CLI command. To iterate quickly, they bypass IaC and modify resources interactively. Drift arises when these exploratory updates are not synchronized back into IaC once an optimal configuration is chosen. *(3) Debugging and observability:* during live debugging, engineers often enable additional resources

such as logging, tracing, or temporary dashboards. IaC provides little support for live diagnostics, so these are typically set up directly through consoles or APIs. Similarly, drift arises when these changes are not synchronized in the IaC codebase.

In all these examples, infrastructure drift could occur. The IaC team lacks immediate visibility into when and why changes occur. Without direct operational context, they may miss relevant updates, struggle to identify which resources need to be managed or are temporary ones, and face difficulties in determining the complete scope of resource changes across different cloud services. Hence, infrastructure drift is fundamental to cloud operations at scale, and unfortunately, there is no fully automated solution to address the IaC reconciliation problem today. Next, we describe the three challenges in automated IaC reconciliation and our insights in NSync.

2.3 Challenge #1: Inferring Intent for IaC Reconciliation

The first challenge is to infer what changes must be reconciled into the IaC configuration. When resources are created through non-Terraform interfaces, Terraform cannot detect these changes since these resources are not part of its managed state – they are invisible from Terraform's perspective. The current practice is that DevOps engineers need to manually discover those resources—they must sift through SDK code commits, tickets, and verbose cloud audit logs (e.g., AWS CloudTrail); inspect multiple cloud services and regions; and attempt to understand the relationships between newly created resources and existing infrastructure. This is difficult and time-consuming, hindering operations at scale.

The insight in NSync is that modern cloud platforms provide auditing services that record all infrastructure modifications, regardless of the interface used to perform them. Examples include AWS CloudTrail [4], Azure Event Hub [31], Google Cloud Audit [18], and Alibaba ActionTrail [3]. The API call traces logged in these services offer a structured view of infrastructure modifications. Hence, these traces reflect operational behaviors, including out-of-band changes that bypass the IaC framework, providing opportunities for automated IaC reconciliation. If we could interpret API traces to synthesize the corresponding IaC code updates, this will bridge the gap and translate any cloud modifications into a desired IaC update.

API-Driven Reconciliation. Based on this insight, NSync designs an API-driven IaC reconciliation mechanism. Recent advances in large language models (LLMs) make it feasible to infer high-level semantic intent from a collection of API calls. The challenge is to identify intent from noisy traces. API traces are verbose and include redundant or transient operations (e.g., resources that are created and then deleted) as well as events that may be logged out of order. Hence, NSync needs to filter out noise from API traces to identify the actual changes, and understand the intended infrastructure modifications, without explicit repair instructions (e.g., specifications, test cases).

2.4 Challenge #2: Generating IaC Patches from Intent

However, even with precise knowledge of the intended infrastructure changes, patching IaC configurations to reflect those changes is challenging. In the cloud, there is no centralized place to query the complete infrastructure state; instead, the state is fragmented and only accessible through resource or service-specific APIs. Today, DevOps engineers must manually query resource states through CLI or consoles. These interfaces return flat, segmented descriptions populated with ephemeral runtime values, which are difficult to interpret or reuse. To bring these updates into the IaC codebase, engineers then need to inspect the existing configurations and manually infer not only the correct representation for each cloud resource, but also the dependencies among them. This task is made harder by the frequent evolution of both cloud APIs and IaC schemas, which requires constant tracking and re-interpretation. As infrastructures grow larger, this workflow quickly becomes unmanageable. These limitations motivate a more adaptive, agentic approach

powered by LLMs, and we target a different goal of reconciling IaC configurations as a update patch. An LLM agent with tools can access up-to-date cloud and IaC information, generalize across resource types, and infer the structure of arbitrary user IaC codebases.

Pursuing this approach, however, requires conquering two key technical barriers. The first stems from the difficulty in testing an IaC patch. Unlike conventional code-generation settings, there is no safe way to perform live testing. While tools like `terraform plan` can preview potential changes, these dry-run capabilities are limited to only IaC-managed resources and cannot fully validate changes involving out-of-band resources. At the same time, another barrier is that LLMs have limited context window, but IaC repositories can be arbitrarily large, while only a small fraction of the code is relevant to a given drift. These stand in contrast to conventional code-generation agents, which rely on rich environment feedback and have a scoped context—the agent can execute code, observe failures, and refine outputs using precise runtime signals as guidance [10, 28, 39, 56]. Such feedback is significantly weakened in the IaC setting.

IaC Patch Generation without Live Testing. Our design is to guide the agent by carefully managing context to isolate only the IaC fragments relevant to a drift, and developing specialized tools that help the agent extract relevant context and reason about changes safely, without manipulating live infrastructure. In the absence of execution feedback, the agent can rely only on our read-only IaC tools, e.g., checking IaC syntax, previewing prospective changes, for safe operations.

2.5 Challenge #3: Leveraging LLMs Efficiently for IaC Reconciliation

A third challenge lies in how to efficiently leverage LLMs for reconciliation. In most domains, adapting LLMs to a specialized task relies on fine-tuning, which is resource-intensive and poorly suited to cloud infrastructure operations—providers rapidly evolve APIs and semantics, while IaC frameworks also change frequently. Hence, fine-tuning would mean that in order to maintain accuracy, the model needs to be frequently re-trained. Beyond fine-tuning, recent research explores lifelong learning for agents, where systems continuously accumulate knowledge and improve through repeated interactions with their environment [44, 46]. These approaches have shown promise, but almost all operate in sandboxed domains, such as simulated games or controlled benchmarks, where agents can freely explore, fail, and recover at little cost. Cloud infrastructure presents a fundamentally harder setting: cloud infrastructure does not provide a safe environment for experimentation and learning. Safe and effective lifelong learning for IaC reconciliation remains an open challenge.

Learning and Reuse Across Reconciliations. Our approach to enabling continual learning is based on the observation that IaC reconciliation is a repeated process. Cloud infrastructure is long-lived, and agents may repeatedly encounter similar drift patterns as it evolves. Hence, even without free exploration, knowledge from past reconciliation attempts can be carried forward and reused safely and naturally. Specifically, the correspondence between API traces and IaC structures, or lessons from failed attempts, need not be rediscovered each time. Likewise, once an agent has learned the structure and conventions of a given codebase, such knowledge should be leveraged in future reconciliations for the same infrastructure. We design NSync to incorporate prior experience, enabling safe knowledge accumulation for improving reconciliation accuracy and robustness. Our design also addresses the downside that incorporating prior knowledge enlarges context and increases inference cost, making efficiency itself a central challenge.

Together, these three challenges motivate the design of NSync, an agentic system that makes IaC reconciliation safe, efficient, and continually improving over time.

3 Design of NSync

NSync is an agentic system that uses API call traces that record cloud infrastructure events, and updates the original IaC configuration to reflect the current infrastructure state. Figure 2 illustrates the architecture of NSync. The **intent identification** component analyzes noisy API call traces from the cloud provider to determine if there are any infrastructure changes, and analyze the nature of those changes. When infrastructure changes are detected, the second component, **patch generation**, synthesizes an update with the help of specialized IaC reconciliation tools, and is given access to the source configuration, the identified change intent, and a self-organized evolving knowledge base per IaC project. It outputs updated configurations that accurately reflect the current state of the infrastructure, ensuring alignment between the IaC code and the actual deployed resources. Finally, our agent uses **continuous learning** to record past experience to achieve better performance over a longer time horizon.

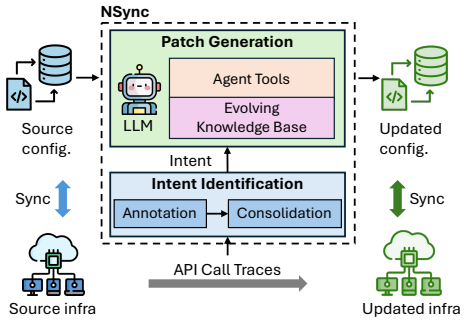

Fig. 2. Overview of NSync.

3.1 Intent Identification

This step aims to identify the infrastructure change intent from raw API traces. One naïve solution is to have the LLM directly process API traces, but this presents several challenges. First, the traces contain substantial noise which is irrelevant to infrastructure changes, such as read-only operations, retry attempts, and user-specific account information. Second, API structures vary significantly across different cloud services, using inconsistent naming conventions and parameter formats, making uniform analysis difficult. Third, auditing services can populate API events out of order [7, 30] due to asynchronous collection across services and regions and low timestamp resolution. This means the system should not rely on the sequential order of API calls in traces; instead, it must reason about the semantics of operations and distill the outcome of mutating actions while tolerating imprecise arrival order. Our intent identification method focuses on the content and relationships of events rather than their raw sequence in the trace.

Before intent identification, we preprocess the traces to extract a clean sequence of mutating events—API calls that alter infrastructure state. We discard read-only calls (e.g., `Describe*`, `List*`), deduplicate retries by keeping only the final success, and remove irrelevant fields such as timestamps or request IDs, thereby reducing trace volume while preserving all state-changing operations. The intent identification process then proceeds in two stages: *annotation* and *consolidation*.

The key idea is to consolidate the API events and identify the corresponding IaC-level resource blocks. Recall that imperative API calls eventually trigger changes on cloud states, which need to be encoded as IaC configuration updates. Inferring IaC resource-level changes (create, delete, update) from the API calls is not easy. For instance, while the `CreateVpc` call corresponds to creating a new VPC block, a `CreateTags` call on a VPC does not create a new resource but rather updates the VPC. To capture such distinctions, we normalize API calls into a consistent schema using an LLM.

3.1.1 Step 1: Annotation. Our key contribution in this step is a neurosymbolic annotation procedure that uses LLMs to produce consistent, schema-aligned labels for heterogeneous API calls. We design a fixed schema that the LLMs must adhere to; further, we encourage consistent labeling across event categories and resource types using two mechanisms: (1) the annotator maintains memory

Table 1. Annotation schema for API calls. Fields not applicable are set to `null`.

Field	Description
`category`	Operation type: `create`, `delete`, `attach`, `detach`, `update` or `unknown`
`type`	Primary resource type (e.g., `Vpc`, `Instance`) for create/delete/update
`id`	Primary resource identifier (e.g., `vpc-123`, `i-456`), else `null`
`type1`	First resource type in attach/detach relation, else `null`
`id1`	First resource identifier in attach/detach relation, else `null`
`type2`	Second resource type in attach/detach relation, else `null`
`id2`	Second resource identifier in attach/detach relation, else `null`

(T) of previously inferred resource types to ensure coherence across a trace, and (2) annotation is executed in batches with retries to guard against occasional LLM errors.

Table 1 shows the standard schema that we define, which labels each API call into one of five canonical categories: *create*, *delete*, *update*, *attach*, and *detach*. Each annotated event captures only the essential information: operation type, resource type, and unique resource identifier(s). For example, an EC2 instance launch (`RunInstances`) is represented as *create*(instance, i-1234), while a volume attachment becomes *attach*(volume-1, instance-2).

Algorithm 1: LLM-Based API Annotation

Input: API calls C, schema S, batch size b, retries r
Output: Annotated calls A
Init:
$A \leftarrow \emptyset$ (annotations) ;
$T \leftarrow \emptyset$ (resource types) ;
foreach $B \subseteq C,\ |B| = b$ **do**
  $k \leftarrow 0$;
  while $k \leq r$ **do**
    $\hat{B} \leftarrow \mathrm{LLM}(B, S, T)$;
    if $|\hat{B}| = |B|$ **then**
      break
    $k \leftarrow k + 1$;
  if $|\hat{B}| \neq |B|$ **then**
    $\hat{B} \leftarrow \{\emptyset\}^{|B|}$
  $A \leftarrow A \cup \textsc{Merge}(B, \hat{B})$;
  $T \leftarrow T \cup \textsc{Types}(\hat{B})$;
return A;

Our LLM annotator reasons about the semantics of an event in context. For example, `CreateTags` may look like a new creation, but it is actually an *update*, and the target of the update depends on parameters—adding a tag could modify a VM, a VPC, or another resource type. Likewise, `AuthorizeSecurityGroupIngress` appears to authorize something new, but in fact it updates the configuration of an existing security group. By contrast, `RunInstances` might seem to update the state of a VM, yet it actually provisions a new instance and should be classified as a *create*. These examples show why simple regex or keyword matching would be brittle. LLMs, on the other hand, having been trained on cloud documentation including API usage examples and descriptions, understand these nuances; moreover, because cloud APIs continue to evolve, LLMs can adapt to these changes. To improve scalability, we annotate API calls in batches rather than individually. Algorithm 1 takes API calls C and processes them in batches B of size b, using the schema S and previously collected resource types T as context. For each batch, the LLM attempts up to r retries to produce annotations $\hat{B}$. The results are merged into the annotation set A, and the newly discovered types from $\hat{B}$ are added to T for subsequent batches.

3.1.2 Step 2: Consolidation. After annotation, each API call is in standardized form (e.g., `create(instance, i-1234)` or `attach(volume-1, instance-2)`), giving us a labeled trace of mutating events. Reconciliation only requires reasoning about *persistent drift*, i.e., the infrastructure changes that remain at the end of the trace. Relative to that goal, the trace still contains noise: there may be many transient operations, such as redundant or overwritten updates, or creation of temporary resources followed by deletion. Consolidation identifies the set of persistent drifts that need to be reconciled in IaC.

Given an annotated API trace, we first organize the events by their resource identifiers. An identifier is a unique number generated by the cloud when a resource (e.g., VM) is created, and

remains consistent throughout its lifecyle. For each *node identifier*, we collect all events that occurred on this node/resource—e.g., `subnet-456: [create, update]` means that a unique ID `subnet-456` was created and then updated. For each *edge* that connects two resources, we collect all operations that have been performed on this edge, written as ⟨`id1, id2`⟩. For instance, ⟨`volume-1, instance-2`⟩`: [attach, detach, attach]` denotes three operations on this edge. We then apply the following reduction rules independently to each node or edge identifier:

- **Persistent Create.** If a resource is created and never deleted, we retain only the create event and drop intermediate updates. Creates take precedence as they introduce new resources, while configuration details can be recovered later.
- **Persistent Delete.** If a resource was deleted, we retain only the delete and remove all associated updates prior to the deletion. Deletes, like creates, are higher-precedence persistent drifts, as they eliminate resources entirely from the infrastructure.
- **Persistent Update(s).** If a resource was only updated (with no create or delete), we conservatively retain the update. This ensures that updated attributes, even if they are not defined in the original IaC configuration, will be explicitly captured.
- **Persistent Attach/Detach.** Balanced attach/detach pairs cancel out, while any net imbalance is retained as a conservative signal of persistent drift.

The output of this reduction is a compact set of persistent drifts, organized by resource and edge IDs, and it is deliberately conservative to avoid discarding meaningful information.

3.2 Patch Generation

Next, NSync generates a patch to fix the persistent drift. Generating IaC patches with LLMs presents unique challenges compared to traditional automated program repair (APR). In APR, repair correctness can be validated through test execution and runtime signals. However, for cloud management, validating patch correctness is more challenging. Direct execution of a patched IaC project is not feasible as it would modify live infrastructure. While tools like `terraform plan` can preview potential changes after a patch is applied, it is designed for previewing planned infrastructure updates from a known terraform state to a new desired state, rather than validating drift reconciliation patches where resources have been created or modified outside of terraform's knowledge. This makes it difficult to assess patch correctness before actual deployment.

- `drift_report`: a safe, read-only operation that previews the alignment between the current patch and the cloud state (inspired by `terraform plan`; see later).
- `self_critique`: inspired by reflection and self-refinement methods [28, 39], this tool lists all edits made so far and requires the agent to reason about their correctness and necessity. By critiquing its own patch history, the agent avoids hallucinated changes, scope creep, and untracked divergences from the IaC codebase.

The patch generation workflow in NSync is an iterative patch–evaluate–refine loop. The agent first generates a candidate patch, then evaluates it using feedback tools, and finally critiques its own edits before refining. This mirrors the trial-and-error workflow of program repair agents, but here it is realized entirely through safe, read-only operations tailored to IaC.

3.2.1 Drift Report. `Terraform plan` is not suitable for patching because it assumes the outdated IaC configuration is the "source of truth" and generates update plans that propose reverting cloud-side changes to match stale configurations An agent reading this information effectively has to "reverse engineer" the changes with substantial ingenuity. Worse, `Terraform plan` exposes attribute-level differences for each managed resource, including runtime fields marked as "known after apply," resulting in verbose outputs that inflate context length and overwhelming LLMs.

This combination of misleading guidance and excessive verbosity effectively poisons the agent's reasoning, pushing it toward incorrect or irrelevant patches.

`drift_report` reframes planning into reconciliation-oriented feedback. Instead of emitting a full update plan, it trims outputs to only those resources that actually drifted and annotates them with their locations in the IaC codebase. This targeted feedback helps NSync generate and refine patches more effectively. The agent first analyzes the condensed API trace to create an initial patch, then uses `drift_report` in an iterative patch-evaluate-refine loop to improve the solution. Currently, `drift_report` assumes all detected infrastructure changes should be incorporated into the IaC patch. However, in practice, some drifts are intentional and should be preserved, while others should be reverted. Distinguishing between them remains future work, as this study focuses on automatically reconciling all drifted resources under IaC management so that DevOps engineers can subsequently perform operations directly through IaC.

3.2.2 Self Critique. Another challenge in the agentic workflow is maintaining focus across multiple edits. After reasoning over the drift and applying a sequence of patches, the agent can lose track of its overall reconciliation objective, leading to hallucinated changes or oversized edits that diverge from the intended fix [27]. Prior work suggests that reflection can improve agent reliability [28, 39, 56], but existing methods depend on execution feedback or test cases—signals unavailable in IaC reconciliation, where live execution is unsafe.

We design `self_critique`, a safe reflection mechanism that operates without execution, which operates at a coarser granularity than `drift_report`. Whereas `drift_report` helps the agent verify whether a candidate patch has fully reconciled the infrastructure by checking if any drift remains. `self_critique` periodically reviews the accumulated edits and prompts the agent to reason about whether its progress still aligns with the drift intent. By surfacing reconciliation at this higher level, the tool enforces consistency and helps the agent stay focused on the reconciliation objective.

3.3 Continual Learning

Our next design is on continual learning. Cloud infrastructure is long-lived and reconciliation tasks repeated arise. The lack of memory would force the agent to repeatedly rediscover mappings between API traces and IaC resources, relearn project-specific conventions, and re-synthesize fixes for error cases it has already encountered. Such redundancy not only wastes compute resources but also limits robustness, since reconciliation of large IaC projects often involves recurring drift patterns that demand consistency across runs. To overcome this limitation, we introduce a lightweight, domain-specific knowledge base (KB) that enables continual learning [44, 46] across reconciliation sessions. The KB captures information about prior reconciliations, such as effective patching strategies, frequently observed drift patterns, and project-specific semantics (e.g., naming conventions, module structures, or dependency layouts). This KB is continuously updated after a reconciliation task completes; during the next reconciliation, the agent retrieves relevant knowledge on-demand to guide patch generation, effectively grounding its reasoning in accumulated experience rather than starting from scratch.

We design the KB using the following principles. (1) Updates are *selective*: only reconciliations that succeed contribute new entries, while failed sessions are discarded to avoid propagating errors. (2) The KB is *scoped per project* rather than shared globally, since much of the knowledge (e.g., naming conventions, module structure) is project-specific; this prevents cross-project contamination while allowing repeated reconciliations on the same project to benefit from accumulated insights. The learning process is *agent-driven*: the agent explicitly invokes the tools `knowledge_update` and `knowledge_retrieval` when it judges that the current context is complete and accurate enough to warrant persistence for reuse, as opposed to always logging every intermediate action.

Constructing a useful KB entry requires the complete and up-to-date context of a reconciliation attempt; otherwise, the agent risks recording incomplete or misleading knowledge. We employ a ReAct-style [55] prompting strategy that enforces an *Observation–Thought–Action* loop on every iteration, instructing the agent to ground its reasoning in the latest observations before deciding whether to update or retrieve knowledge. For example, if the agent repeatedly fails to apply a patch because an S3 bucket update requires adjusting its versioning setting, once it discovers the correct fix it can invoke `knowledge_update` to record this resolution for future runs. Over time, this mechanism enables a form of experience-guided reconciliation, where the agent progressively adapts to each IaC project.

The KB is a light-weight, project-specific text file for our prototype. It can be easily extended with vector database [26] support for larger or more complex knowledge. The `knowledge_retrieval` tool queries the KB to access previously recorded reconciliation insights, while `knowledge_update` writes new successful reconciliation experiences to the KB. This lightweight implementation ensures that knowledge persistence and retrieval remain simple yet effective, while maintaining the per-project scoping principle. The agent has the ability to edit or discard outdated KB entries when previously successful strategies fail under evolving external systems, ensuring that knowledge remains accurate over time.

4 Evaluation

We conduct a comprehensive evaluation of NSync on real-world Terraform projects and realistic drifts. We design our experiments to answer five key research questions:

- **RQ1 (Effectiveness) + RQ2 (Efficiency)**: How effective is NSync at reconciling infrastructure drifts, as measured by pass@k accuracy, compared to a baseline agent? How well does NSync reduce computational overhead while maintaining or improving reconciliation accuracy?
- **RQ3 (Intent Identification)**: How well does API trace consolidation help identify the intent, thus improving performance, and what are the trade-offs in accuracy and efficiency?
- **RQ4 (Patch Generation)**: How do specialized IaC tools improve reconciliation performance, and what does their usage frequency and timing reveal about the agentic problem-solving process?
- **RQ5 (Continual Learning)**: How reliably does the learning algorithm improve outcomes?

In the rest of this section, we first describe our methodology in curating realistic drifts and evaluating reconciliation effectiveness, discuss the experimental setup, and then answer these questions.

4.1 Evaluation Methodology: Generating and Evaluating Realistic Drifts

Drift reconciliation is a novel task, which requires a new methodology for evaluation. Our evaluation pipeline addresses two challenges: (1) generating realistic drifts scenarios and (2) evaluating them against ground truth. This pipeline and dataset are another contribution of our paper.

Realistic drift scenarios. To evaluate reconciliation performance, we need realistic drift scenarios that represent how DevOps engineers modify cloud infrastructure in practice. We source these scenarios primarily from AWS Systems Manager (SSM) Automation documents [8], which are AWS-provided operational runbooks containing both natural language descriptions and executable API scripts for common infrastructure management tasks. For example, one such scenario is "enabling CloudWatch monitoring on EC2 instances." By using these authoritative AWS documents, we ensure our injected drifts reflect real-world infrastructure changes. We further supplement these AWS-sourced scenarios with manually curated drifts specific to each IaC project. The complete list of SSM-sourced scenarios and project-specific scenarios can be found in Appendix A.1 and Appendix A.2 respectively.

Generating assessable drifts. The next challenge arises in generating drift in a manner that allows reconciliation to be evaluated. One naïve strategy is to induce drift by simply executing

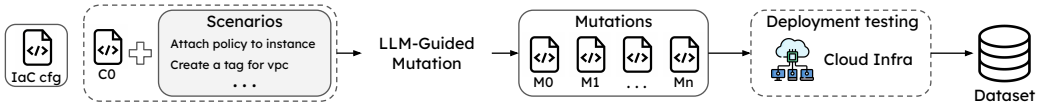

Fig. 3. Dataset generation pipeline. Mutated configurations are generated from a base configuration and natural-language scenario, validated through a deployment testing, and retained if successful.

the API scripts of the drift scenarios in the IaC-managed infrastructure. However, this approach is difficult to evaluate: there is no *ground truth* against which we can compare the reconciled IaC configuration, short of manually inspecting the new IaC configuration and the cloud state, which would not permit scalable evaluation. We design a novel dataset generation pipeline (Fig. 3). which analyzes a drift scenario using LLMs, but generates a validated *IaC configuration* capturing the mutations; it further validates the mutation by attempting to deploy it. During the generation process, we reset the infrastructure back to its original state after each mutation—matching the base configuration `C0`—manually repairing the infrastructure if necessary. We inject drift by deploying mutated IaC configurations rather than directly executing API scripts. This approach enhances reproducibility and enables automated validation of patch correctness. First, for reproducibility, API scripts typically depend on existing resource identifiers and runtime parameters from the original infrastructure, making them difficult to reproduce since these identifiers cannot be known beforehand and must be resolved during execution. In contrast, IaC configurations declaratively specify the desired state, making drift injection reproducible across different environments. Second, for validation, having the mutated IaC configuration provides a ground truth against which we can automatically evaluate patch correctness - we can compare the reconciled patch against the known desired infrastructure state derived from the ground truth configuration.

Evaluation pipeline. The inputs of the evaluation pipeline are a base workspace containing the IaC configuration and a mutated configuration. The configuration in the base workspace is deployed, generating a local Terraform state. Then, in a separate "drift workspace", we induce drift by applying the mutated configuration to the same underlying cloud infrastructure. Since the two workspaces are isolated, the base workspace is unaware of the out-of-band changes. We use CloudTrail to capture API events, and provide this trace together with the base workspace to NSync for drift reconciliation. To validate patch correctness, we compare the patched configuration against the ground truth infrastructure state using `terraform plan`. The ground truth is derived from the mutated configuration and represented as a local state file. A patch is considered correct when it satisfies two conditions: (1) all resources that previously existed outside of Terraform are now defined in the configuration and are ready to be imported to the local state file via "import" actions; and (2) all Terraform-managed resources match their desired state—meaning the configuration and state are consistent, and any deleted resources have been removed from the configuration. If `terraform plan` reports no changes other than the required "import" actions, the patch is considered correct. Any additions, updates, or deletions indicate that the patch failed to fully reconcile the infrastructure with the ground-truth state. In evaluation, when an agent produces the correct resource under a different name, we detect the match and insert `moved` blocks to avoid false negatives. A `moved` block informs Terraform of differently aliases of the same resource, preventing spurious delete–create actions.

Dataset contribution. Using this pipeline, we have curated the first dataset of realistic IaC drift scenarios for systematic evaluation of reconciliation agents. Our five base projects are public Terraform respositories that vary widely in size and domain, from tens to thousands of resources (Table 2). They include **lab12** [43] (multi-VPC networking), **ssm** [24] (zero-downtime patching), **livescore** [15] (event-driven service), **flask** [37] (REST API with DynamoDB), and **megamesh** [36]

Table 2. Benchmark statistics: configuration size, mutation space, and event counts.

Benchmark	#Resources	LoC	#Scenarios	#API Events		#Mutating Events	
				Avg	(Min, Max)	Avg	(Min, Max)
lab12 [43]	47	289	96	183.2	(90, 441)	9.5	(1, 47)
flask [37]	74	799	94	280.8	(158, 671)	10.0	(1, 56)
ssm3 [24]	66	377	103	268.0	(149, 737)	8.0	(1, 48)
live-score [15]	193	1582	61	612.3	(373, 1163)	8.5	(1, 42)
mega-mesh [36]	1930	7087	18	3735.9	(3692, 3920)	7.8	(1, 31)

(multi-region VPC mesh). Our dataset contains 372 validated drift cases, generated by mutating and deploying these real-world Terraform projects.

4.2 Experimental Setup

We implemented NSync in 11k lines of code in Python, and it targets Terraform-based IaC for the AWS cloud; it uses Boto3 to interface with CloudTrail for obtaining API traces.

Table 3. Tools available to all agents.

Agent Tools	Description
`file_read`	Read a file
`file_write`	Overwrite a file
`editor`	Fine-grained edits
`shell`	Run shell commands

Baselines. We evaluate three agents, all built using the Strand agent framework [42] and deployed via the AWS Bedrock service with Claude 3.7 Sonnet as the backbone LLM. The Baseline solution is a LLM agent equipped with pre-built file editing tools and shell access, but without domain specialization. The second method, NSync-NL, is our system without continual learning, where each reconciliation run is treated independently. Our full system, NSync, builds on the same framework and backbone, but with all domain-specific mechanisms detailed in Section 3—including intent identification, patch generation (e.g., `drift_report` and `self_critique`), as well as a project-level knowledge base (KB) that enables continual learning across runs. The annotation process in both variants relies on the same Claude 3.7 Sonnet model. All three systems operate on identical inputs (mutating API traces and the IaC project directory) and share the same core editing tools shown in Table 3. All experiments are conducted against realistic deployments on AWS to ensure results reflect practical operating conditions. For each system, we perform three independent runs, with drift scenarios presented in arbitrary order to reduce any bias from experiment ordering.

Metrics. We focus on correctness and efficiency, using the pass@k metric. A reconciliation attempt is considered correct if the agent's patched configuration, when evaluated with `terraform plan` against the *ground truth state* and the *live infrastructure* created by the mutation, produces no differences—for example, the out-of-band creation of a new EC2 instance has been captured by the patch, i.e., it includes the corresponding resource block (e.g., `aws_instance`). Efficiency is measured in terms of two complementary metrics: tokens processed (computational cost), agent steps. Since multiple candidate patches may be generated for each scenario, we report results using the pass@k metric, which measures whether at least one of the top-k patches is correct while accounting for the associated efficiency cost. We report both pass@1 and pass@3 results to evaluate the single-attempt accuracy of the system and its robustness under limited retries.

4.3 RQ1 & 2: Effectiveness and Efficiency

We start by answering RQ1 and RQ2, and Table 4 summarizes the detailed accuracy and efficiency results of NSync, NSync-NL, and the baseline. NSync proves highly effective, significantly outperforming the baseline agent. Across 372 diverse real-world projects and mutations, NSync achieves 0.97 accuracy, far exceeding the baseline. At the same time, it is more efficient, consuming on average 0.47M tokens per reconciliation, a 1.47× improvement in token efficiency. Furthermore,

Table 4. Accuracy and Efficiency Comparison between Baseline, NSync-NL, and NSync. M: Millions.

Benchmark	# Exp.	Pass@3 Accuracy			Avg Tokens (M)			Avg Steps		
		Baseline	NSync-NL	NSync	Baseline	NSync-NL	NSync	Baseline	NSync-NL	NSync
lab12	96	0.89	0.98	**0.99**	0.45	**0.26**	0.36	21.7	**14.5**	16.2
flask	94	0.64	0.95	**0.97**	0.73	**0.57**	0.66	26.1	**18.2**	19.4
ssm3	103	0.74	0.92	**0.97**	0.52	**0.32**	0.33	21.0	**15.3**	15.5
live-score	61	0.57	0.93	**0.98**	0.70	**0.39**	0.51	22.4	**17.3**	21.6
mega-mesh	18	0.72	**0.94**	**0.94**	1.05	**0.44**	0.50	18.9	18.1	**15.6**
Overall	**372**	0.71	0.95	**0.97**	0.69	**0.40**	0.47	22.0	**16.7**	17.7

both NSync-NL and NSync significantly outperform the baseline across all benchmarks, requiring fewer reasoning cycles and less computation overall. On simpler projects (e.g., `lab12`), which has a flat structure without modules, all methods perform well. The performance gap widens on projects with custom or prebuilt modules, where the code structure is more complex. And the performance of NSync-NL and NSync scale more gracefully to project complexity.

A closer inspection shows that the baseline consumes more tokens and fails more often, because it relies heavily on raw `terraform plan` outputs. Its context becomes cluttered with verbose and often irrelevant information produced by this Terraform-native tool. The context window overflows, triggering summarization that discards useful details. This forces the agent into repeated reasoning and revisiting Terraform files, further increasing token usage. Even worse, the baseline hallucinates: due to the bloated context dominated by `plan` noise, it incorrectly reasons that applying the current configuration will fix the drift (i.e., undoing the out-of-band change), rather than updating the configuration to reflect the changes. In contrast, NSync's specialized `drift_report` tool feeds only the relevant drifted resources into the model's context. This focused signal not only improves efficiency but also yields higher accuracy by avoiding context dilution.

NSync has close to 100% success rates, and our analysis of the failure cases shows that the remaining failures typically arise in `import` handling. In Terraform, `import` is used to bring externally created resources under IaC management by linking them with corresponding configuration blocks. In one case from the `flask` project, when importing a resource, the agent mistakenly attempted "`RootAccUsage:/aws/cloudtrail/flask-micros`", which incorrectly reversed the name and identifier, and consequently failed to discover the correct format. The correct format is "`/aws/cloudtrail/flask-micros:RootAccUsage.`" Such cases could be addressed with retrieval-augmented generation (RAG), where the precise import syntax is supplied directly to the agent. In summary, these results demonstrate that NSync yields substantial accuracy and efficiency gains.

Table 5. Ablation study of intent identification (`IID`) on the `live-score` benchmark.

Method	Pass@3	Tokens (M)	Steps
baseline	0.57	0.70	22.4
NSync-NL	**0.93**	**0.39**	**17.3**
NSync-NL w/o `IID`	0.92	0.51	22.5
NSync	**0.98**	**0.51**	**21.6**
NSync w/o `IID`	0.98	0.66	28.1

4.4 RQ3: Intent Identification

Next (RQ3), we show that intent identification improves efficiency while maintaining accuracy. The small annotation overhead from LLM is amortized by the much larger savings during reconciliation. Table 5 compares NSync with and without intent identification (`IID`) on the `live-score` benchmark. Both NSync and NSync-NL maintain high accuracy, while `IID` increases their efficiency by 23% in token usage and steps. Accuracy remains unchanged for NSync (0.98) and even slightly improves for NSync-NL (0.93 vs. 0.92).

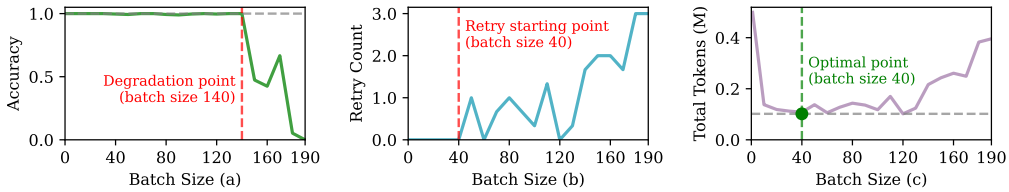

Fig. 4. LLM annotation of a synthetic API trace with 190 mutating events.

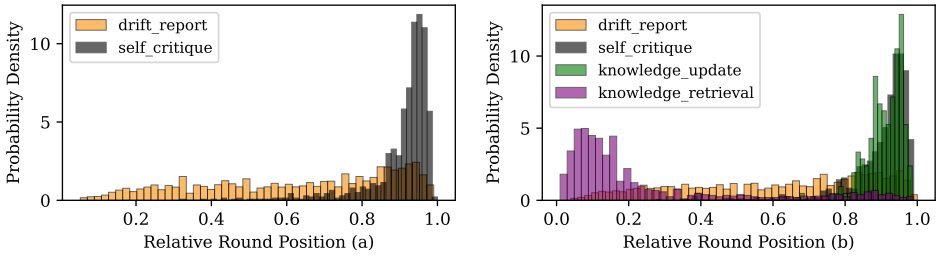

Fig. 5. Distribution of tool usage across reconciliation progress. The x-axis denotes the normalized round position (`tooluse_round` / `total_round`) for each run. Figure 5a In NSync-NL, `drift_report` is used uniformly throughout the process, while `self_critique` is concentrated near the end. Figure 5b In NSync, similar patterns are observed, but knowledge tools exhibit distinct timing: `knowledge_retrieval` is typically invoked early to supply prior context, while `knowledge_update` appears near the end to record new insights.

Intent identification proceeds in two steps: annotation and consolidation. We next examine the scalability of its annotation stage—specifically, whether LLMs can annotate large batches of API events effectively. We find that LLM annotation consumes on average 5.6K tokens per drift—only about 1% of the overall reconciliation cost, which is on the order of millions of tokens. To stress-test the system, we construct a single trace containing 190 mutating API calls. Figure 4 shows the effect of varying batch size on this trace. Accuracy drops sharply beyond 140, marking a clear degradation point. The number of retries increases once the batch size exceeds 40, indicating that the model struggles to process overly large batches in a single pass. This effect is also visible in token usage: it is minimized around a batch size of 40, which we identify as the optimal operating point. Beyond this point, additional retries required to handle larger batches lead to higher overall token consumption. These results show that event consolidation is essential, and that batching around 40 mutating API calls is sufficient in practice, providing a good balance of accuracy, efficiency, and robustness. We therefore adopt a batch size of 40 for all experiments.

4.5 RQ4: Patch Generation

Table 6 shows that specialized tools substantially improve reconciliation performance, with each removal lowering accuracy and the absence of `drift_report` causing the steepest drop. We perform an ablation study on the `live-score` benchmark. Removing any tool or knowledge-base (KB) operation reduces reconciliation accuracy. Without learning—i.e., in NSync-NL, where the KB operations `knowledge_update` and `knowledge_retrieval` are disabled—accuracy drops from 0.98 to 0.93. Excluding `self_critique` reduces it further to 0.81, while excluding

Table 6. Performance ablation study on the `live-score` benchmark.

Method	Pass@3
NSync	**0.98**
NSync-NL	0.93
NSync w/o `drift_report`	0.60
NSync w/o `self_critique`	0.81

Table 7. Performance across runs under the pass@1 metric

Benchmark	Baseline		NSync-NL		NSync	
	Pass@1	range ± std	Pass@1	range ± std	Pass@1	range ± std
lab12	0.63	(0.52, 0.78) ± 0.11	0.88	(0.88, 0.89) ± 0.01	**0.90**	(0.86, 0.95) ± 0.04
flask	0.40	(0.31, 0.47) ± 0.07	**0.70**	(0.69, 0.72) ± 0.01	0.65	(0.55, 0.74) ± 0.08
ssm3	0.51	(0.49, 0.52) ± 0.01	0.73	(0.65, 0.80) ± 0.06	**0.80**	(0.76, 0.84) ± 0.03
live-score	0.38	(0.34, 0.39) ± 0.03	0.70	(0.66, 0.75) ± 0.04	**0.84**	(0.80, 0.93) ± 0.06
mega-mesh	0.53	(0.47, 0.59) ± 0.05	0.76	(0.59, 0.88) ± 0.13	**0.78**	(0.71, 0.82) ± 0.06
Overall	0.49	(0.43, 0.55) ± 0.05	0.76	(0.69, 0.81) ± 0.05	**0.80**	(0.73, 0.86) ± 0.05

`drift_report` has the most severe impact, lowering accuracy to 0.60. This shows that each tool plays an important role, with `drift_report` and `self_critique` being especially critical for achieving high performance.

We next analyze the timing of tool use during reconciliation. As shown in Figure 5, `drift_report` is invoked by the agent throughout the process, `self_critique` is concentrated near the end, and in NSync, knowledge tools follow a clear temporal structure (`retrieval` at earlier stages, `update` at later stages). This confirms that the agent is able to use our tools effectively, in a structured and phase-specific manner.

Taking a closer look to tool usage frequency across the three methods (Figure 6). While the baseline relies heavily on generic `file_read` and shell operations, NSync-NL shifts its usage toward domain-specific tools, `drift_report` and `self_critique`. The frequency of editor and `file_write` calls remains similar across agents, since these tools are still required to generate patches. By contrast, `shell` usage decreases substantially, as the baseline often used `shell` to locate files and invoke native Terraform commands—tasks that are largely replaced by domain-specific tools in NSync. `file_read` usage also decreases, since the baseline repeatedly re-scans the codebase to rediscover context, whereas NSync makes more efficient use of retained context and KB. NSync further incorporates KB operations, though at relatively low frequency compared to other tools; this is also reflected in Figure 5 as KB operation only happens at the beginning and the end.

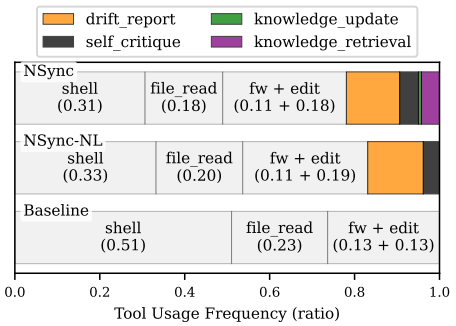

Fig. 6. Tool usage frequency breakdown.

4.6 RQ5: Continual Learning

To answer RQ5, we find that NSync remain the most robust under stricter pass@1 evaluation when using our methods, as shown in Table 7; it also maintains reliable performance across experiment runs where drifts are presented in arbitrary order. Figure 8 presents a snippet of the evolving reconciliation knowledge base learned from experience. The knowledge spans diverse categories, including how to inspect resource state with Terraform commands, map API parameters to Terraform attributes, resolve formatting issues, and apply the custom tool `drift_report` to pinpoint attribute-level drifts more effectively.

Table 7 evaluates robustness under the pass@1 metric, where no retries are allowed and results are averaged across three runs. The baseline performs poorly in this setting, with an overall accuracy of only 0.49, highlighting its unreliability when a single attempt must succeed. NSync-NL improves substantially to 0.76, and NSync further increases robustness to 0.80 by reusing knowledge from

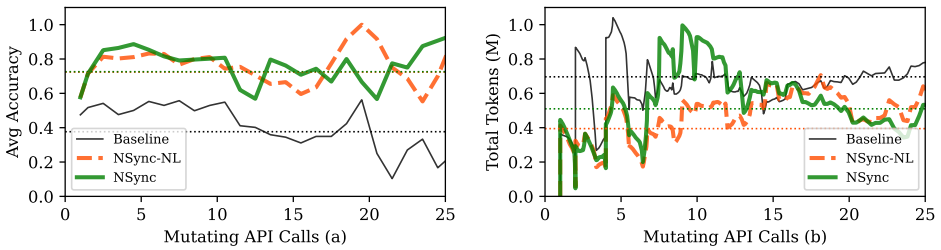

Fig. 7. Performance comparison across mutating API calls. Figure 7a reports average accuracy across all drift scenarios from five projects, reorganized by the number of mutating API calls. Figure 7b presents the corresponding token cost for reconciliation as drift complexity increases.

A snippet of the evolving reconciliation knowledge base for `lab12`.

- When working with VPC flow logs, remember to escape the `log_format` string with double dollar signs (`$$`) to avoid Terraform interpolation errors.
- Use `terraform state show <resource_name>` to examine the current state of a specific resource.
- Use `moved` blocks in `move.tf` to handle resource renaming and prevent unnecessary resource recreation.
- For EventBridge rules, use `is_enabled = false` to disable rules instead of `state = "DISABLED"` which is the API parameter but not the Terraform attribute name
- For CloudTrail resources, ensure the S3 bucket policy includes the correct permissions for CloudTrail to write logs.
- When importing KMS keys, remove `deletion_window_in_days` if it causes drift.
- When importing SSM documents, ensure the content format (quotes, indentation, etc.) exactly matches the actual state.
- When importing S3 buckets created outside of Terraform, make sure to also import any associated bucket policies separately.
- Use `drift_report` to identify specific attributes that need to be changed in resources.

Fig. 8. An example of the evolving reconciliation knowledge base. All entries for the `lab12` project represent accumulated knowledge, continually updated by the agent from its prior reconciliation experience

prior reconciliations. Importantly, the drift scenarios are presented in arbitrary order across runs, ensuring that improvements are not due to any favorable ordering of the data. This confirms that the learning method is agnostic to ordering and that there is no hidden bias or structure in our benchmarks that guarantees improvement. While all methods see a drop compared to pass@3, the degradation is smaller for NSync-NL and especially for NSync, which remains consistently strong across runs. These results demonstrate that learning not only improves average accuracy but also reduces variance, yielding more reliable reconciliation outcomes.

Figure 7 shows that, as drift complexity increases (more mutating API calls), the baseline shows a clear downward trend in accuracy, while NSync-NL remain stable. In terms of efficiency, the baseline's token usage steadily grows with complexity, whereas NSync-NL consistently requires fewer tokens, and NSync adds only a modest overhead due to knowledge retrieval. These results highlight that our methods are more resilient under increasing drift complexity, maintaining accuracy and efficiency where the baseline degrades. when numbers of mutating API increases the baseline token usage eventually suppress NSync.

5 Discussion

IaC lifting. A class of IaC tools such as `Terraform-cfg-gen` [21], Terraformer [19], aws2tf [6], Terracognita [2], aztfexport [1] attempt a related but different goal—to "lift" a non-IaC managed

infrastructure to an IaC configuration from scratch. This need arises because some companies want to port their legacy infrastructure (e.g., created via API scripts) to be managed by IaC. This requires reconstructing configurations by issuing these APIs calls to retrieve the fragmented cloud state, and then inferring IaC-level configurations using heuristic mappings. Lifting attempts at a different goal from reconciliation, and is inherently more difficult: these tools add support for each new resource or service manually, and continuously track updates from both the cloud provider side (new APIs, changing semantics) and the IaC side (schema and language evolution). In practice, these tools produce brittle code with limited coverage [34], embedding raw runtime values (e.g., identifiers, subnet IDs) instead of reusable references. The result is flat, non-modular configurations with generic names that are neither maintainable nor reusable. And importantly, they aim at lifting from an infrastructure to new configurations from scratch, rather than producing targeted patches to an existing IaC project. For larger infrastructures, these lifting tools degrade further, often producing syntax errors or unusable configurations [34]. Simply put, these tools attempt at program synthesis from scratch, whereas ours aims at program repair. By reducing the complexity of the task, and by leveraging LLMs, NSync sidesteps the key challenges that lead to brittleness in lifting tools.

Beyond Terraform and AWS. Our evaluation is on the leading IaC platform—Terraform—and the most popular cloud—AWS. We believe that NSync's architecture is generalizable to other IaC frameworks (e.g., Pulumi [35], OpenTofu [32]) and cloud providers (e.g., Azure, GCP). Key changes may include different API monitoring tools across clouds and resource management practices. For instance, Azure and GCP's event logs may use different event schemas compared to AWS CloudTrail. Each IaC framework also has its own local state scheme and planning operations. Hence, an interesting avenue of future work is to generalize NSync to other IaC frameworks and clouds.

6 Related Work

AI Agent for Cloud Management. Recent advances have explored the use of AI agents to automate diverse aspects of cloud operations, or "AIOps." However, much of this work only focuses on incident detection, root cause diagnosis, and remediation [11, 12, 33, 47, 51]—e.g., via log mining, causal inference, or LLM-powered analysis to recommend or execute corrective actions. At the same time, several vision papers highlight the potential of AI agents to expand beyond diagnostic tasks to handle provisioning, continuous monitoring, and workload optimization [13, 14, 54]. NSync is an AIOps design but is specialize to detecting and reconciling drift in IaC frameworks.

AI Agent for Program Repair. Automated Program Repair (APR) has been extensively studied, with recent surveys showing that large language models (LLMs) have transformed the field by enabling repair through fine-tuning, prompting, procedural workflows, and agentic frameworks [52, 57]. Fine-tuned repair systems like VulMaster [59], RepairLLaMA [40], and RepairCAT [25] leverage domain-specific bug corpora for high accuracy, while prompting approaches such as AlphaRepair [50] and TracePrompt [20] achieve lightweight deployment through zero/few-shot queries. More advanced pipelines integrate retrieval or analysis feedback to guide iterative repair (e.g., Repilot [48], Agentless [49]), and fully agentic systems such as SWE-Agent [53], RepairAgent [9], OpenHands [45], and AutoCodeRover [58] orchestrate external tools, testing, and multi-step reasoning under LLM control. These works primarily target source code bugs in conventional software projects, with correctness validated against unit tests or benchmarks. NSync builds on the program repair perspective but in a novel domain: Infrastructure-as-Code. Unlike traditional APR, where explicit failing tests provide repair oracles, IaC reconciliation operates without runnable test cases: its "specification" is implicitly encoded in cloud API traces. This shifts

the repair challenge from patching faulty logic based on test cases to synthesizing declarative updates that represent API traces, ensuring that the IaC codebase reflects the true deployed state.

7 Conclusion

Infrastructure-as-Code (IaC) frameworks are gaining popularity, and they encode the infrastructure in a declarative configuration as the source of truth. However, when IaC frameworks are used together with other management interfaces (e.g., cloud consoles, CLI, SDK), the IaC configuration no longer captures the de facto state, leading to infrastructure drift. We presented NSync, the first agentic system for automated IaC reconciliation. NSync relies on cloud APIs to identify and reconcile drift, and it works by cleaning and consolidating noisy API traces, generating patches to update configurations, and continuously learning from experience. To support systematic evaluation, we introduced a pipeline for generating realistic and assessable drift scenarios, and curated the first IaC reconciliation dataset containing 372 validated cases across five real-world Terraform projects. Experiments show that NSync achieves 0.97 accuracy; a 26% improvement over the baseline, while reducing token usage by 1.5×. These results highlight that IaC reconciliation is both feasible and practical with modern agentic systems.

References

[1] 2025. aztfexport. https://github.com/Azure/aztfexport

[2] 2025. terracognita. https://www.cycloid.io/open-source/terracognita

[3] Alibaba Cloud. 2025. Alibaba Cloud ActionTrail. https://www.alibabacloud.com/help/en/actiontrail/ ActionTrail monitors and records your Alibaba Cloud account activities; latest documentation updated February 12, 2025.

[4] Amazon Web Services. 2024. AWS CloudTrail Documentation. https://docs.aws.amazon.com/awscloudtrail.

[5] Amazon Web Services. 2024. AWS Systems Manager Automation. Amazon Web Services. https://docs.aws.amazon.com/systems-manager/latest/userguide/systems-manager-automation.html AWS Systems Manager Documentation.

[6] Amazon Web Services. 2024. aws2tf. AWS Samples. https://github.com/aws-samples/aws2tf GitHub Repository.

[7] Amazon Web Services. 2024. Understanding CloudTrail events. https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-events.html AWS Documentation.

[8] AWS. 2024. AWS Systems Manager Automation runbook reference. https://docs.aws.amazon.com/systems-manager-automation-runbooks/latest/userguide/automation-runbook-reference.html.

[9] Islem Bouzenia, Premkumar Devanbu, and Michael Pradel. 2024. Repairagent: An autonomous, llm-based agent for program repair. arXiv preprint arXiv:2403.17134 (2024).

[10] Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. 2022. CodeT: Code Generation with Generated Tests. https://arxiv.org/abs/2207.10397

[11] Yinfang Chen, Manish Shetty, Gagan Somashekar, Minghua Ma, Yogesh Simmhan, Jonathan Mace, Chetan Bansal, Rujia Wang, and Saravan Rajmohan. 2025. AIOpsLab: A Holistic Framework to Evaluate AI Agents for Enabling Autonomous Clouds. https://arxiv.org/abs/2501.06706

[12] Yinfang Chen, Huaibing Xie, Minghua Ma, Yu Kang, Xin Gao, Liu Shi, Yunjie Cao, Xuedong Gao, Hao Fan, Ming Wen, Jun Zeng, Supriyo Ghosh, Xuchao Zhang, Chaoyun Zhang, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Tianyin Xu. 2024. Automatic Root Cause Analysis via Large Language Models for Cloud Incidents (EuroSys '24).

[13] Qian Cheng, Doyen Sahoo, Amrita Saha, Wenzhuo Yang, Chenghao Liu, Gerald Woo, Manpreet Singh, Silvio Saverese, and Steven C. H. Hoi. 2023. AI for IT Operations (AIOps) on Cloud Platforms: Reviews, Opportunities and Challenges.

[14] Yingnong Dang, Qingwei Lin, and Peng Huang. 2019. AIOps: Real-World Challenges and Research Innovations. In 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion).

[15] Chris Dobson. 2023. live-scores. https://github.com/ChrisDobby/live-scores

[16] env0. 2025. Meet Cloud Compass: AI-Assisted IaC Coverage, Audit and Risk Mitigation. https://www.env0.com/blog/meet-cloud-compass-ai-assisted-iac-coverage-audit-and-risk-mitigation. https://www.env0.com/blog/meet-cloud-compass-ai-assisted-iac-coverage-audit-and-risk-mitigation Accessed: 2025-10-06.

[17] Firefly. 2025. State of Infrastructure as Code 2025. Firefly.ai. https://www.firefly.ai/state-of-iac-2025

[18] Google Cloud. 2024. Cloud Audit Logs Overview. https://cloud.google.com/logging/docs/audit.

[19] Google Cloud Platform. 2024. Terraformer. Google Cloud Platform. https://github.com/GoogleCloudPlatform/terraformer GitHub Repository.

[20] Mirazul Haque, Petr Babkin, Farima Farmahinifarahani, and Manuela Veloso. 2025. Towards Effectively Leveraging Execution Traces for Program Repair with Code LLMs. arXiv preprint arXiv:2505.04441 (2025).

[21] HashiCorp. 2024. Generating Configuration from Import. https://developer.hashicorp.com/terraform/language/import/generating-configuration Terraform Documentation.

[22] Hashicorp. 2025. Manage resource drift. https://developer.hashicorp.com/terraform/tutorials/state/resource-drift#introduce-drift Terraform Documentation.

[23] HashiCorp. 2025. Terraform. https://developer.hashicorp.com/terraform

[24] Joseph Heyburn. 2023. terraform-examples: AWS SSM Automation Examples. https://github.com/jdheyburn/terraform-examples/tree/main/aws-ssm-automation-3

[25] Nan Jiang and Yi Wu. 2024. RepairCAT: applying large language model to fix bugs in AI-generated programs. In Proceedings of the 5th ACM/IEEE international workshop on automated program Repair. 58–60.

[26] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-scale similarity search with GPUs. arXiv:1702.08734 [cs.CV] https://arxiv.org/abs/1702.08734

[27] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the Middle: How Language Models Use Long Contexts. Transactions of the Association for Computational Linguistics (2024).

[28] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-Refine: Iterative Refinement with Self-Feedback. arXiv:2303.17651 [cs.CL] https://arxiv.org/abs/2303.17651

[29] Microsoft. 2024. Automation in Microsoft Sentinel. Microsoft. https://learn.microsoft.com/en-us/azure/sentinel/automation/automation Microsoft Azure Documentation.

[30] Microsoft Azure. 2022. Configuring event ordering policies for Azure Stream Analytics. https://learn.microsoft.com/en-us/azure/stream-analytics/event-ordering Azure Documentation.

[31] Microsoft Azure. 2024. Azure Activity Logs and Resource Manager Logs. https://learn.microsoft.com/en-us/azure/azure-monitor/platform/activity-log.

[32] OpenTofu Project. 2024. OpenTofu: The Open Source Infrastructure as Code Tool. https://opentofu.org/ Accessed: October 24, 2025.

[33] Changhua Pei, Zexin Wang, Fengrui Liu, Zeyan Li, Yang Liu, Xiao He, Rong Kang, Tieying Zhang, Jianjun Chen, Jianhui Li, Gaogang Xie, and Dan Pei. 2025. Flow-of-Action: SOP Enhanced LLM-Based Multi-Agent System for Root Cause Analysis (WWW '25). Association for Computing Machinery.

[34] Jingjia Peng, Yiming Qiu, Patrick Tser Jern Kon, Pinhan Zhao, Yibo Huang, Zheng Guo, Xinyu Wang, and Ang Chen. 2025. Automated Lifting for Cloud Infrastructure-as-Code Programs. In 2025 IEEE/ACM International Workshop on Cloud Intelligence & AIOps (AIOps).

[35] Pulumi Corporation. 2024. Pulumi: Modern Infrastructure as Code. https://www.pulumi.com/ Accessed: October 24, 2025.

[36] Jude Quintana. 2023. terraform-aws-mega-mesh. https://github.com/JudeQuintana/terraform-aws-mega-mesh

[37] AWS Samples. 2023. deploy-python-flask-microservices-to-aws-using-open-source-tools. https://github.com/aws-samples/deploy-python-flask-microservices-to-aws-using-open-source-tools

[38] SentinelOne. 2023. What Is Infrastructure as Code (IaC) Security? SentinelOne. https://www.sentinelone.com/cybersecurity-101/cloud-security/what-is-infrastructure-as-code-iac-security/

[39] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language Agents with Verbal Reinforcement Learning. arXiv:2303.11366 [cs.AI] https://arxiv.org/abs/2303.11366

[40] André Silva, Sen Fang, and Martin Monperrus. [n. d.]. RepairLLaMA: Efficient Representations and Fine-Tuned Adapters for Program Repair. Technical Report 2312.15698. arXiv. http://arxiv.org/abs/2312.15698

[41] Spacelift. 2025. Drift Detection - Spacelift Documentation. https://docs.spacelift.io/concepts/stack/drift-detection. https://docs.spacelift.io/concepts/stack/drift-detection Accessed: 2025-10-06.

[42] Strands Agents. 2024. Latest Updates. Strands Agents. https://strandsagents.com/latest/ Website content may no longer be available.

[43] Its-All-About the Journey. 2024. AWS Advanced Networking. https://github.com/Its-All-About-the-Journey/AWS-Advanced-Networking.

[44] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An Open-Ended Embodied Agent with Large Language Models. https://arxiv.org/abs/2305.16291

[45] Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. 2024. Openhands: An open platform for ai software developers as generalist agents. arXiv preprint arXiv:2407.16741 (2024).

[46] Zhiruo Wang, Daniel Fried, and Graham Neubig. 2024. TroVE: Inducing Verifiable and Efficient Toolboxes for Solving Programmatic Tasks. https://arxiv.org/abs/2401.12869

[47] Zefan Wang, Zichuan Liu, Yingying Zhang, Aoxiao Zhong, Jihong Wang, Fengbin Yin, Lunting Fan, Lingfei Wu, and Qingsong Wen. 2024. RCAgent: Cloud Root Cause Analysis by Autonomous Agents with Tool-Augmented Large Language Models (CIKM '24).

[48] Yuxiang Wei, Chunqiu Steven Xia, and Lingming Zhang. 2023. Copiloting the copilots: Fusing large language models with completion engines for automated program repair. In Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 172–184.

[49] Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. 2024. Agentless: Demystifying llm-based software engineering agents. arXiv preprint arXiv:2407.01489 (2024).

[50] Chunqiu Steven Xia and Lingming Zhang. 2022. Less training, more repairing please: revisiting automated program repair via zero-shot learning (ESEC/FSE 2022). Association for Computing Machinery. https://doi.org/10.1145/3540250.3549101

[51] Yiming Xiang, Zhenning Yang, Jingjia Peng, Hermann Bauer, Patrick Tser Jern Kon, Yiming Qiu, and Ang Chen. 2025. Automated Bug Discovery in Cloud Infrastructure-as-Code Updates with LLM Agents. In 2025 IEEE/ACM International Workshop on Cloud Intelligence & AIOps (AIOps).

[52] Boyang Yang, Zijian Cai, Fengling Liu, Bach Le, Lingming Zhang, Tegawendé F Bissyandé, Yang Liu, and Haoye Tian. 2025. A Survey of LLM-based Automated Program Repair: Taxonomies, Design Paradigms, and Applications. arXiv preprint arXiv:2506.23749 (2025).

[53] John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. Swe-agent: Agent-computer interfaces enable automated software engineering. Advances in Neural Information Processing Systems 37 (2024), 50528–50652.

[54] Zhenning Yang, Archit Bhatnagar, Yiming Qiu, Tongyuan Miao, Patrick Tser Jern Kon, Yunming Xiao, Yibo Huang, Martin Casado, and Ang Chen. 2025. Cloud Infrastructure Management in the Age of AI Agents. SIGOPS Operating Systems Review (2025). doi:10.1145/3759441.3759443

[55] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In Proceedings of the International Conference on Learning Representations (ICLR). https://openreview.net/forum?id=WE_vluYUL-X

[56] Siyu Yuan, Zehui Chen, Zhiheng Xi, Junjie Ye, Zhengyin Du, and Jiecao Chen. 2025. Agent-R: Training Language Model Agents to Reflect via Iterative Self-Training. arXiv:2501.11425 [cs.AI] https://arxiv.org/abs/2501.11425

[57] Quanjun Zhang, Chunrong Fang, Yang Xie, Yuxiang Ma, Weisong Sun, Yun Yang, and Zhenyu Chen. 2024. A Systematic Literature Review on Large Language Models for Automated Program Repair. arXiv preprint arXiv:2405.01466 (2024).

[58] Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. 2024. Autocoderover: Autonomous program improvement. In Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis. 1592–1604.

[59] Xin Zhou, Kisub Kim, Bowen Xu, DongGyun Han, and David Lo. 2024. Out of sight, out of mind: Better automatic vulnerability repair by broadening input ranges and sources. In Proceedings of the IEEE/ACM 46th international conference on software engineering. 1–13.

A Benchmark Curation

This section details the benchmark curation process in Section 4.1. Our benchmark includes five Terraform projects and their associated mutations, designed to evaluate IaC reconciliation approaches. The five Terrafrom projects are from GitHub and listed in Table 2.

We create the benchmark to enable the following evaluation workflow. First, we deploy the original IaC configuration (one of the five Terraform projects) to provision the initial cloud infrastructure. Then, we inject infrastructure drift by applying one of our curated mutations to the same infrastructure. During this process, we collect two key components: (1) the API call traces that capture the drift-inducing operations, which serve as input to the reconciliation approach, and (2) the Terraform state from the mutated configuration, which serves as ground truth representing the up-to-date infrastructure state. To evaluate the correctness of a reconciliation approach's output, we execute `Terraform Plan` using the reconciled IaC configuration against this ground truth state. A successful reconciliation should result in no planned changes, indicating that the updated configuration accurately captures the infrastructure's current state.

For a IaC project, its mutations are generated via a LLM agent based on drift scenarios specified in natural language. We derived drift scenarios from two complementary sources: (1) **AWS Systems Manager (SSM) Automation runbooks** [8] and (2) **manually crafted drift scenarios**. The primary source, SSM Automation runbooks, are AWS-provided operational documents that contain both natural language descriptions and executable API scripts for common infrastructure management tasks. These runbooks represent real-world infrastructure modifications. We use their descriptions as drift scenarios. For each IaC project, we prompted an LLM with the runbook's drift scenario descriptions to generate corresponding mutated IaC configurations. We then validated each generated mutation through deployment testing - verifying that the mutation could be successfully deployed and reverted back to the original configuration. We supplement the drift scenarios with a few manually-curated ones tailored to each IaC project.

To complement our evaluation with persistent drifts, we also incorporated false-positive cases. These cases were created by first deploying a mutated configuration and then reverting it to the original state, simulating situations where infrastructure changes are initiated but subsequently rolled back. Although these cases generate API call traces that indicate infrastructure modifications occurred, they result in no persistent drift in the final infrastructure state. This approach allows us to test how well reconciliation tools handle traces that introduce non-persistent drifts. We sampled these false-positive cases from our overall mutation pool across different projects.

In total, we collected **372 mutation cases**, of which **127 are false-positive cases**. The following subsections describe the AWS runbook–based mutations and the manually crafted ones in detail.

A.1 Sampling from AWS Automation Runbooks

We began with a collection of 763 publicly available AWS Systems Manager (SSM) runbooks, which are pre-built operational scripts. Through a systematic filtering process, we narrowed these down to 90 unique scenarios (shown in Table 8). The filtering process excluded runbooks that do not perform infrastructure modifications, meaning they don't issue mutating API calls. We also removed runbooks that require pre-existing resources not included in our IaC project configurations. Additionally, we excluded runbooks that create irreversible or non-importable infrastructure changes, such as those that scale up EBS volumes (which cannot be scaled down) or create EBS volume images (which cannot be imported back into IaC).

After filtering down to 90 unique runbook scenarios, we selected applicable scenarios for each IaC project based on their specific cloud resources. The selection is done using a LLM-based classifier. The classifier takes as input the runbook description and the resources provisioned by a

Terraform project, and output whether the runbook can be meaningfully applied to mutate the AWS infrastructure. The prompt to the LLM is shown in Listing 1. The runbooks per project are listed in Tables 9, 10, 11, 12, and 13

For each applicable scenario, we employed an LLM to generate infrastructure mutations by analyzing both the IaC configuration and the scenario description, as illustrated in Figure 3. To ensure the quality and reliability of these mutations, we implemented a two-step validation process: first verifying that each mutation could be successfully deployed, and then confirming that the infrastructure could be safely reverted to its original configuration after deployment. Only mutations that passed both validation steps were included in evaluation.

Table 8. AWS Runbook Scenarios for Infrastructure Mutations

No.	AWS Runbook	Description
1	AWS-EnableNeptuneDbBackupRetentionPeriod	This runbook enables automated backups for an Amazon Neptune DB cluster by setting a backup retention period between 7 and 35 days.
2	AWS-EnableVPCFlowLogs	This runbook creates VPC flow logs to capture IP traffic information for specified VPCs, publishing the logs to either CloudWatch or S3 for network traffic monitoring and analysis.
3	AWSConfigRemediation-DropInvalidHeadersForALB	This runbook configures an Application Load Balancer to remove HTTP headers with invalid fields by enabling the drop invalid headers attribute.
4	AWSSupport-ConfigureDNSQueryLogging	This runbook automates the configuration of DNS query logging for AWS Route 53 public hosted zones or VPCs, capturing DNS queries and publishing them to CloudWatch Logs, S3, or Kinesis to help with troubleshooting connectivity issues.
5	AWSDocs-S3StaticWebsiteCustomDomain	This runbook automates the configuration of a static website hosted on Amazon S3 using a custom domain registered with Route 53, creating necessary S3 buckets, configuring website hosting, and setting up domain routing.
6	AWSConfigRemediation-EnableSecurityHub	This runbook enables AWS Security Hub for an AWS account in the current region and verifies its enabled status.
7	AWSConfigRemediation-EnablePITRForDynamoDbTable	This runbook enables Point-In-Time Recovery (PITR) for a specified Amazon DynamoDB table and verifies the feature was successfully activated.
8	AWSFleetManager-AddUsersToGroups	This runbook adds a list of specified users to multiple groups across Windows and Linux systems, verifying user and group existence before attempting the additions.
9	AWSEC2Launch-RunMigration	This runbook automates the migration process from EC2Config and EC2Launch v1 to EC2Launch v2 on Windows instances, with an option to perform a dry run before actual migration.
10	AWS-RestrictIncomingTraffic	This runbook restricts incoming TCP traffic to EC2 security groups by removing ingress rules that allow unrestricted access (0.0.0.0/0 and ::/0) on specified ports.
11	AWS-RemediateSSMAgentVPCEndpoints	This runbook creates VPC endpoints required by SSM Agent if they don't exist, and associates them with one subnet in each availability zone to ensure proper SSM connectivity.
12	AWS-CreateManagedLinuxInstance	This runbook automates the creation of a Linux managed instance in AWS with Systems Manager (SSM) integration, configuring necessary components like security groups, IAM roles, and the SSM agent for remote management.
13	AWS-SetRequiredTags	This runbook adds specified tags to one or more AWS resources across various services, with the ability to track both successful and failed tagging operations.
14	AWSConfigRemediation-CreateGuardDutyDetector	This automation enables Amazon GuardDuty by creating and activating a detector in the current AWS region.
15	AWS-EnableStepFunctionsStateMachineLogging	This runbook enables or updates logging configuration on an AWS Step Functions State Machine, specifying logging levels, CloudWatch log group destination, execution data inclusion options, and X-Ray tracing.
16	AWS-EnableCloudTrail	This runbook creates and enables a new CloudTrail trail to log AWS API activity, directing the log files to a specified S3 bucket.
17	AWS-EnableCWAlarm	This runbook creates CloudWatch alarms for AWS resources (EC2 instances, EBS volumes, S3 buckets, and RDS clusters) that don't already have them, configuring metric-based monitoring with customizable thresholds and comparison operators.
18	AWS-ArchiveS3BucketToIntelligentTiering	This runbook creates or modifies an S3 Intelligent-Tiering configuration to automatically optimize storage costs by moving data to more cost-effective tiers based on access patterns.
19	AWSConfigRemediation-CreateCloudTrailMultiRegionTrail	This runbook creates a multi-region CloudTrail trail that logs activities across multiple AWS regions and delivers the log files to a specified S3 bucket with log file validation enabled.
20	AWS-EnableS3BucketEncryption	This runbook enables default server-side encryption on an Amazon S3 bucket using the specified encryption algorithm (defaulting to AES256).
21	AWS-CreateSnapshot	The runbook creates a snapshot of a specified EBS volume and waits for the snapshot creation process to complete.

No.	AWS Runbook	Description
22	AWSConfigRemediation-DetachIAMPolicy	This runbook detaches a specified IAM policy from all entities (groups, users, and roles) it is attached to, verifying complete removal after execution.
23	AWSConfigRemediation-EnableLoggingForALBAndCLB	This runbook enables and verifies logging for AWS Application Load Balancers (ALB) or Classic Load Balancers (CLB) by configuring them to send access logs to a specified S3 bucket.
24	AWSConfigRemediation-UpdateXRayKMSKey	This runbook enables encryption for AWS X-Ray data using a customer-managed AWS KMS key to meet security best practices.
25	AWSConformancePacks-SecurityBestPracticesforAWSWAF	This runbook deploys AWS Config rules to enforce security best practices for AWS WAF, ensuring WAF is enabled on Application Load Balancers and API Gateway stages, and verifying that WAF Regional rules, rule groups, and web ACLs are properly configured and not empty.
26	AWSSupport-ConfigureEC2Metadata	This runbook helps configure EC2 Instance Metadata Service (IMDS) options, allowing users to enforce IMDSv2, set the HTTP hop limit, or disable metadata access entirely. Note: Changes to IMDS configuration should be made cautiously as they may break applications that rely on specific metadata service versions or access.
27	AWSConfigRemediation-EnableVPCFlowLogsToS3Bucket	This runbook replaces an existing VPC Flow Log that publishes to CloudWatch Logs with one that publishes to an Amazon S3 bucket instead.
28	AWS-EnableCloudTrailCloudWatchLogs	This runbook configures CloudTrail trails to deliver their events to a specified CloudWatch log group, enabling log monitoring and analysis capabilities.
29	AWSConformancePacks-SecurityBestPracticesforCloudTrail	This runbook implements security best practices for CloudTrail by configuring AWS Config rules that monitor and enforce CloudTrail settings like encryption, log validation, multi-region logging, CloudWatch Logs integration, and S3 data event tracking.
30	AWSConfigRemediation-ConfigureS3PublicAccessBlock	This runbook creates or modifies the S3 PublicAccessBlock configuration for an AWS account to restrict public access to S3 buckets.
31	AWS-AttachIAMToInstance	This runbook attaches an IAM role to an EC2 instance, with the ability to create a new instance profile if needed and optionally replace an existing IAM profile.
32	AWS-DisableEventBridgeRule	This runbook disables a specified rule in Amazon EventBridge, allowing users to temporarily prevent a rule from triggering events.
33	AWSConfigRemediation-EnableAutoScalingGroupELBHealthCheck	This runbook enables Elastic Load Balancer (ELB) health checks for an Amazon EC2 Auto Scaling group and sets an appropriate grace period before those health checks begin.
34	AWSSupport-ConfigureTrafficMirroring	This runbook configures traffic mirroring to help troubleshoot connectivity issues between load balancers and EC2 instances by copying inbound and outbound network traffic from network interfaces.
35	AWS-DisableS3BucketPublicReadWrite	This runbook disables public read and write access to an S3 bucket by applying public access block settings.
36	AWSSupport-SetupIPMonitoringFromVPC	This runbook sets up continuous network monitoring for specified IPv4/IPv6 addresses by creating an EC2 instance that runs ping, MTR, traceroute, and traceTCP tests at regular intervals, with results sent to CloudWatch Logs and visualized on a dashboard for troubleshooting network issues.
37	AWSConfigRemediation-ConfigureS3BucketPublicAccessBlock	This runbook creates or modifies the PublicAccessBlock configuration for an S3 bucket to control public access settings, helping to enhance security by restricting public access to bucket contents.
38	AWS-UpdateALBDesyncMitigationMode	This runbook updates the desync mitigation mode on an Application Load Balancer (ALB) to help manage how the load balancer handles requests that might pose security risks to applications.
39	AWSConfigRemediation-DeleteDefaultVPCRoutes	This runbook deletes default routes (0.0.0.0/0 and ::/0) from a specified Amazon EC2 VPC route table to enhance network security by removing unrestricted internet access.
40	AWSSupport-RemediateLambdaS3Event	This runbook troubleshoots and remedies issues with S3 event triggers for Lambda functions by checking event configurations, adding necessary permissions, and fixing resource policy problems.
41	AWSConfigRemediation-DeleteUnusedSecurityGroup	This runbook deletes non-default security groups that are not being utilized by elastic network interfaces, while preserving any security group named "default".

No.	AWS Runbook	Description
42	AWS-RemediateSSMAgentHTTPSAccess	This runbook configures EC2 security groups to enable HTTPS communication between SSM Agent on EC2 instances and Systems Manager through VPC endpoints.
43	AWSConfigRemediation-EnableAccountAccessAnalyzer	This runbook creates an IAM Access Analyzer at the AWS account level to help identify resources that are shared with external entities.
44	AWSConfigRemediation-SetIAMPasswordPolicy	This runbook sets and verifies the IAM password policy for an AWS account, configuring parameters such as password length, complexity requirements, expiration, and reuse prevention.
45	AWS-CreateS3PolicyToExpireMultipartUploads	Creates and applies an S3 bucket lifecycle policy to automatically expire incomplete multipart uploads after a specified number of days, while preserving any existing lifecycle rules.
46	AWS-RemediateSSMAgentVPCAttributes	This runbook remediates VPC attribute issues by enabling DNS support and DNS hostnames in VPCs to fix unmanaged EC2 instance connectivity problems.
47	AWS-EnableS3BucketKeys	This runbook enables S3 Bucket Keys on a specified S3 bucket to create data keys for new objects, supporting either server-side encryption with S3 managed keys (SSE-S3) or AWS KMS keys (SSE-KMS).
48	AWS-ConfigureDocker	This runbook configures Windows instances to either install or uninstall Docker and container functionality.
49	AWS-SetupManagedRoleOnEc2Instance	This runbook configures an EC2 instance with a managed IAM role that has the AmazonSSMManagedInstanceCore policy, enabling Systems Manager management capabilities for the instance.
50	AWS-DisablePublicAccessForSecurityGroup	This runbook disables SSH (port 22) and RDP (port 3389) access in a specified security group, either from all IP addresses or from a specific IPv4/IPv6 address.
51	AWS-AttachExcludeConditionToS3DenyPolicies	This runbook excludes a specified principal from all Deny statements in an S3 bucket's resource-based policies by adding appropriate ArnNotEquals conditions.
52	AWSConfigRemediation-EnforceEC2InstanceIMDSv2	This runbook enforces IMDSv2 on an EC2 instance by modifying its metadata options to require token-based authentication, enhancing security against potential SSRF vulnerabilities.
53	AmazonCloudWatch-MigrateCloudWatchAgent	This runbook automates the migration from the SSM CloudWatch Plugin to the Amazon CloudWatch Agent on Windows systems by checking agent compatibility, disabling the old agent, installing the new agent, migrating existing configurations, and reconfiguring the new agent.
54	AWS-CreateDSManagementInstance	This runbook creates and configures a Windows management instance for AWS Directory Service, including domain joining the EC2 instance to your AWS-managed Active Directory and installing Active Directory administration tools.
55	AWSConfigRemediation-EnableCloudFrontOriginFailover	This runbook configures origin failover functionality for an Amazon CloudFront distribution by setting up an origin group that automatically redirects traffic to a secondary origin when the primary origin returns specified error codes.
56	AWS-JoinDirectoryServiceDomain	This runbook joins EC2 instances to a specified AWS Directory Service domain, allowing for centralized management of Windows instances through Active Directory.
57	AWSConfigRemediation-RemovePrincipalStarFromS3BucketPolicy	This runbook removes policy statements with wildcard principals (Principal: * or Principal: "AWS": *) for Allow actions from an S3 bucket policy to enhance security, deleting the entire bucket policy if it contains only such wildcard statements.
58	AWSConfigRemediation-EnableCloudFrontAccessLogs	This runbook enables and configures access logging for a specified Amazon CloudFront distribution, storing the logs in a designated S3 bucket with options for log file prefixing and cookie inclusion.
59	AWSConfigRemediation-EnableSystemsManagerSessionManager-AuditLogsToS3	This runbook enables AWS Systems Manager Session Manager audit logging to an Amazon S3 bucket by creating or updating the SSM-SessionManagerRunShell document with the specified S3 bucket configuration.
60	AWSConfigRemediation-EnableNLBCrossZoneLoadBalancing	This runbook enables Cross Zone Load Balancing on an AWS Network Load Balancer and verifies that the setting was successfully applied.
61	AWSSupport-ContainS3Resource	This runbook isolates an S3 bucket or object in response to a security incident by restricting access policies, blocking public access, and enforcing bucket ownership controls, while preserving the resource for investigation.

No.	AWS Runbook	Description
62	AWSSupport-AnalyzeEBSResourceUsage	This runbook checks for unused EBS volumes, snapshots without source volumes, and unused AMIs, then generates CSV reports and stores them in a user-specified S3 bucket to help identify potential cost savings.
63	AWSConfigRemediation-DisableSubnetAutoAssignPublicIP	This runbook disables the automatic assignment of public IP addresses on a specified subnet by setting the MapPublicIpOnLaunch attribute to false.
64	AWS-ModifyDynamoDBProvisionedCapacity	This runbook modifies the read and write provisioned capacity of a DynamoDB table while handling table state transitions and considering auto-scaling configurations.
65	AWS-ConfigureS3BucketVersioning	This runbook enables or suspends versioning for a specified Amazon S3 bucket, configuring whether objects added to the bucket receive unique version IDs or null version IDs.
66	AWSConfigRemediation-EncryptSNSTopic	This runbook enables encryption on an Amazon SNS topic using a customer-managed KMS key to improve security compliance.
67	AWS-ConfigureCloudWatchOnEC2Instance	This runbook enables or disables CloudWatch monitoring on an EC2 instance.
68	AWSConformancePacks-OperationalBestPractices-forAmazonS3withRemediation	This runbook implements operational best practices for Amazon S3 by automatically detecting and remediating security issues including public access, encryption, and logging configurations for S3 buckets.
69	AWSConfigRemediation-EnableKeyRotation	This runbook enables automatic key rotation for an AWS KMS symmetric customer master key (CMK) and verifies that the rotation has been successfully enabled.
70	AWSDocs-S3StaticWebsite	This runbook automates the setup of a static website hosted on Amazon S3 by creating a publicly accessible S3 bucket with website hosting enabled and uploading index and error HTML documents.
71	AWS-EnableS3BucketEventNotifications	This runbook creates or updates Amazon S3 Event Notifications for a specified bucket to send alerts when selected events occur, helping detect accidental or intentional modifications that could lead to unauthorized data access.
72	AWSConfigRemediation-RestrictBucketSSLRequestsOnly	This runbook creates an S3 bucket policy that denies HTTP requests to a specified bucket, enforcing SSL/TLS encryption for all data transfers to enhance security.
73	AWS-ReleaseElasticIP	This runbook releases a specified Elastic IP address from an AWS account using its allocation ID.
74	AWS-ConfigureCloudTrailLogging	This runbook enables or disables logging for a specified CloudTrail based on the input parameter, first checking the current logging status to avoid redundant operations.
75	AWSSupport-EnableVPCFlowLogs	This runbook automates the creation of VPC flow logs for various AWS resources (subnets, elastic network interfaces, VPCs, transit gateways, and transit gateway attachments) with options to publish the logs to CloudWatch Logs or Amazon S3.
76	AWS-UpdateCLBDesyncMitigationMode	This runbook updates the desync mitigation mode on a Classic Load Balancer (CLB) to help protect applications from security risks posed by HTTP desynchronization attacks.
77	AWSSupport-SetupConfig	This runbook automates the setup of AWS Config, creating necessary components like service linked roles, recorder, S3 bucket, delivery channel, and optional aggregator authorizations if they don't already exist, otherwise leveraging existing resources.
78	AWSFleetManager-CreateUser	This runbook creates a local user account on both Windows and Linux systems, with options to specify a description and create a home directory on Linux.
79	AWS-DisableIncomingSSHOnPort22	This runbook disables unrestricted incoming SSH traffic on port 22 for specified EC2 security groups by removing ingress rules that allow public access (from '0.0.0.0/0' and '::/0').
80	AWSConfigRemediation-EnableCWLoggingForSessionManager	This runbook enables AWS Systems Manager Session Manager to store session output logs in a specified CloudWatch log group by creating or updating the necessary configuration document.
81	AWS-CreateEKSClusterWithFargateProfile	Creates an Amazon EKS cluster with a Fargate profile, providing a serverless Kubernetes environment where compute capacity is provisioned by AWS Fargate instead of EC2 instances.

No.	AWS Runbook	Description
82	AWS-SetupInventory	This runbook creates and manages an association between specified EC2 instances and a software inventory document, enabling automated collection of system inventory data on a scheduled basis.
83	AWSDocs-ScaleLoadBalanced	This runbook automates the setup of a scaled and load-balanced application by creating a launch template, auto scaling group, and target group, then associating the auto scaling group with a load balancer.
84	AWSConfigRemediation-EnableCloudTrailEncryptionWithKMS	This runbook encrypts an AWS CloudTrail trail using a specified AWS KMS customer master key to enhance security according to recommended best practices.
85	AWS-CloseSecurityGroup	This runbook closes a security group by removing all ingress and egress rules, effectively blocking all traffic to and from resources associated with the security group.
86	AWS-ConfigureS3BucketLogging	This runbook configures server access logging for an Amazon S3 bucket, directing log files to a target bucket with options for prefix configuration and permission settings.
87	AWSConfigRemediation-EncryptLambdaEnvironmentVariablesWithCMK	This runbook encrypts Lambda function environment variables at rest using a specified AWS KMS customer managed key to enhance security.
88	AWSConfigRemediation-RemoveUnrestrictedSourceIngressRules	This runbook removes all ingress rules from a specified security group that allow traffic from unrestricted source addresses (0.0.0.0/0 for IPv4 and ::/0 for IPv6) to improve security posture.
89	AWSDocs-LambdaWithS3SSMDocument	This runbook demonstrates an automated tutorial that sets up a Lambda function triggered by S3 events to create thumbnail images when new images are uploaded to a source bucket.
90	AWS-ChangeDDBRWCapacityMode	This runbook changes the read/write capacity mode for one or more DynamoDB tables, switching between on-demand mode and provisioned mode with appropriate capacity settings.

Table 9. AWS Runbooks applicable to `ssm`

No.	Idx	Scenario
1	69	AWSConfigRemediation-EnableKeyRotation
2	10	AWS-RestrictIncomingTraffic
3	55	AWSConfigRemediation-EnableCloudFrontOriginFailover
4	71	AWS-EnableS3BucketEventNotifications
5	34	AWSSupport-ConfigureTrafficMirroring
6	26	AWSSupport-ConfigureEC2Metadata
7	68	AWSConformancePacks-OperationalBestPracticesforAmazonS3withRemediation
8	35	AWS-DisableS3BucketPublicReadWrite
9	3	AWSConfigRemediation-DropInvalidHeadersForALB
10	57	AWSConfigRemediation-RemovePrincipalStarFromS3BucketPolicy
11	72	AWSConfigRemediation-RestrictBucketSSLRequestsOnly
12	59	AWSConfigRemediation-EnableSystemsManagerSessionManagerAuditLogsToS3
13	18	AWS-ArchiveS3BucketToIntelligentTiering
14	86	AWS-ConfigureS3BucketLogging
15	24	AWSConfigRemediation-UpdateXRayKMSKey
16	5	AWSDocs-S3StaticWebsiteCustomDomain
17	25	AWSConformancePacks-SecurityBestPracticesforAWSWAF
18	56	AWS-JoinDirectoryServiceDomain
19	6	AWSConfigRemediation-EnableSecurityHub
20	42	AWS-RemediateSSMAgentHTTPSAccess
21	38	AWS-UpdateALBDesyncMitigationMode
22	67	AWS-ConfigureCloudWatchOnEC2Instance
23	89	AWSDocs-LambdaWithS3SSMDocument
24	88	AWSConfigRemediation-RemoveUnrestrictedSourceIngressRules
25	76	AWS-UpdateCLBDesyncMitigationMode
26	23	AWSConfigRemediation-EnableLoggingForALBAndCLB
27	45	AWS-CreateS3PolicyToExpireMultipartUploads
28	28	AWS-EnableCloudTrailCloudWatchLogs
29	50	AWS-DisablePublicAccessForSecurityGroup
30	48	AWS-ConfigureDocker
31	19	AWSConfigRemediation-CreateCloudTrailMultiRegionTrail
32	61	AWSSupport-ContainS3Resource
33	44	AWSConfigRemediation-SetIAMPasswordPolicy
34	30	AWSConfigRemediation-ConfigureS3PublicAccessBlock
35	51	AWS-AttachExcludeConditionToS3DenyPolicies
36	74	AWS-ConfigureCloudTrailLogging
37	43	AWSConfigRemediation-EnableAccountAccessAnalyzer
38	14	AWSConfigRemediation-CreateGuardDutyDetector
39	62	AWSSupport-AnalyzeEBSResourceUsage
40	82	AWS-SetupInventory
41	13	AWS-SetRequiredTags
42	9	AWSEC2Launch-RunMigration
43	22	AWSConfigRemediation-DetachIAMPolicy
44	77	AWSSupport-SetupConfig
45	66	AWSConfigRemediation-EncryptSNSTopic
46	17	AWS-EnableCWAlarm
47	52	AWSConfigRemediation-EnforceEC2InstanceIMDSv2
48	4	AWSSupport-ConfigureDNSQueryLogging
49	83	AWSDocs-ScaleLoadBalanced
50	60	AWSConfigRemediation-EnableNLBCrossZoneLoadBalancing
51	16	AWS-EnableCloudTrail
52	36	AWSSupport-SetupIPMonitoringFromVPC
53	37	AWSConfigRemediation-ConfigureS3BucketPublicAccessBlock
54	70	AWSDocs-S3StaticWebsite
55	15	AWS-EnableStepFunctionsStateMachineLogging
56	80	AWSConfigRemediation-EnableCWLoggingForSessionManager
57	65	AWS-ConfigureS3BucketVersioning
58	11	AWS-RemediateSSMAgentVPCEndpoints
59	47	AWS-EnableS3BucketKeys
60	29	AWSConformancePacks-SecurityBestPracticesforCloudTrail

Table 10. AWS Runbooks applicable to lab12

No.	Idx	Scenario
1	10	AWS-RestrictIncomingTraffic
2	31	AWS-AttachIAMToInstance
3	68	AWSConformancePacks-OperationalBestPracticesforAmazonS3withRemediation
4	81	AWS-CreateEKSClusterWithFargateProfile
5	59	AWSConfigRemediation-EnableSystemsManagerSessionManagerAuditLogsToS3
6	8	AWSFleetManager-AddUsersToGroups
7	6	AWSConfigRemediation-EnableSecurityHub
8	21	AWS-CreateSnapshot
9	42	AWS-RemediateSSMAgentHTTPSAccess
10	27	AWSConfigRemediation-EnableVPCFlowLogsToS3Bucket
11	88	AWSConfigRemediation-RemoveUnrestrictedSourceIngressRules
12	28	AWS-EnableCloudTrailCloudWatchLogs
13	50	AWS-DisablePublicAccessForSecurityGroup
14	19	AWSConfigRemediation-CreateCloudTrailMultiRegionTrail
15	44	AWSConfigRemediation-SetIAMPasswordPolicy
16	30	AWSConfigRemediation-ConfigureS3PublicAccessBlock
17	74	AWS-ConfigureCloudTrailLogging
18	43	AWSConfigRemediation-EnableAccountAccessAnalyzer
19	79	AWS-DisableIncomingSSHOnPort22
20	14	AWSConfigRemediation-CreateGuardDutyDetector
21	53	AmazonCloudWatch-MigrateCloudWatchAgent
22	62	AWSSupport-AnalyzeEBSResourceUsage
23	82	AWS-SetupInventory
24	39	AWSConfigRemediation-DeleteDefaultVPCRoutes
25	13	AWS-SetRequiredTags
26	49	AWS-SetupManagedRoleOnEc2Instance
27	77	AWSSupport-SetupConfig
28	17	AWS-EnableCWAlarm
29	52	AWSConfigRemediation-EnforceEC2InstanceIMDSv2
30	4	AWSSupport-ConfigureDNSQueryLogging
31	85	AWS-CloseSecurityGroup
32	73	AWS-ReleaseElasticIP
33	16	AWS-EnableCloudTrail
34	36	AWSSupport-SetupIPMonitoringFromVPC
35	70	AWSDocs-S3StaticWebsite
36	63	AWSConfigRemediation-DisableSubnetAutoAssignPublicIP
37	80	AWSConfigRemediation-EnableCWLoggingForSessionManager
38	11	AWS-RemediateSSMAgentVPCEndpoints
39	29	AWSConformancePacks-SecurityBestPracticesforCloudTrail

Table 11. AWS Runbooks applicable to `flask`

No.	Idx	Scenario
1	10	AWS-RestrictIncomingTraffic
2	58	AWSConfigRemediation-EnableCloudFrontAccessLogs
3	71	AWS-EnableS3BucketEventNotifications
4	26	AWSSupport-ConfigureEC2Metadata
5	3	AWSConfigRemediation-DropInvalidHeadersForALB
6	57	AWSConfigRemediation-RemovePrincipalStarFromS3BucketPolicy
7	72	AWSConfigRemediation-RestrictBucketSSLRequestsOnly
8	59	AWSConfigRemediation-EnableSystemsManagerSessionManagerAuditLogsToS3
9	12	AWS-CreateManagedLinuxInstance
10	18	AWS-ArchiveS3BucketToIntelligentTiering
11	6	AWSConfigRemediation-EnableSecurityHub
12	42	AWS-RemediateSSMAgentHTTPSAccess
13	67	AWS-ConfigureCloudWatchOnEC2Instance
14	33	AWSConfigRemediation-EnableAutoScalingGroupELBHealthCheck
15	88	AWSConfigRemediation-RemoveUnrestrictedSourceIngressRules
16	45	AWS-CreateS3PolicyToExpireMultipartUploads
17	28	AWS-EnableCloudTrailCloudWatchLogs
18	50	AWS-DisablePublicAccessForSecurityGroup
19	44	AWSConfigRemediation-SetIAMPasswordPolicy
20	30	AWSConfigRemediation-ConfigureS3PublicAccessBlock
21	78	AWSFleetManager-CreateUser
22	74	AWS-ConfigureCloudTrailLogging
23	43	AWSConfigRemediation-EnableAccountAccessAnalyzer
24	90	AWS-ChangeDDBRWCapacityMode
25	62	AWSSupport-AnalyzeEBSResourceUsage
26	82	AWS-SetupInventory
27	54	AWS-CreateDSManagementInstance
28	13	AWS-SetRequiredTags
29	64	AWS-ModifyDynamoDBProvisionedCapacity
30	22	AWSConfigRemediation-DetachIAMPolicy
31	77	AWSSupport-SetupConfig
32	17	AWS-EnableCWAlarm
33	20	AWS-EnableS3BucketEncryption
34	52	AWSConfigRemediation-EnforceEC2InstanceIMDSv2
35	85	AWS-CloseSecurityGroup
36	36	AWSSupport-SetupIPMonitoringFromVPC
37	7	AWSConfigRemediation-EnablePITRForDynamoDbTable
38	37	AWSConfigRemediation-ConfigureS3BucketPublicAccessBlock
39	63	AWSConfigRemediation-DisableSubnetAutoAssignPublicIP
40	80	AWSConfigRemediation-EnableCWLoggingForSessionManager
41	11	AWS-RemediateSSMAgentVPCEndpoints
42	47	AWS-EnableS3BucketKeys

Table 12. AWS Runbooks applicable to `live-score`

No.	Idx	Scenario
1	6	AWSConfigRemediation-EnableSecurityHub
2	10	AWS-RestrictIncomingTraffic
3	74	AWS-ConfigureCloudTrailLogging
4	43	AWSConfigRemediation-EnableAccountAccessAnalyzer
5	66	AWSConfigRemediation-EncryptSNSTopic
6	20	AWS-EnableS3BucketEncryption
7	90	AWS-ChangeDDBRWCapacityMode
8	52	AWSConfigRemediation-EnforceEC2InstanceIMDSv2
9	26	AWSSupport-ConfigureEC2Metadata
10	79	AWS-DisableIncomingSSHOnPort22
11	32	AWS-DisableEventBridgeRule
12	2	AWS-EnableVPCFlowLogs
13	35	AWS-DisableS3BucketPublicReadWrite
14	88	AWSConfigRemediation-RemoveUnrestrictedSourceIngressRules
15	53	AmazonCloudWatch-MigrateCloudWatchAgent
16	87	AWSConfigRemediation-EncryptLambdaEnvironmentVariablesWithCMK
17	72	AWSConfigRemediation-RestrictBucketSSLRequestsOnly
18	75	AWSSupport-EnableVPCFlowLogs
19	28	AWS-EnableCloudTrailCloudWatchLogs
20	50	AWS-DisablePublicAccessForSecurityGroup
21	13	AWS-SetRequiredTags
22	41	AWSConfigRemediation-DeleteUnusedSecurityGroup
23	12	AWS-CreateManagedLinuxInstance
24	86	AWS-ConfigureS3BucketLogging
25	65	AWS-ConfigureS3BucketVersioning
26	19	AWSConfigRemediation-CreateCloudTrailMultiRegionTrail
27	44	AWSConfigRemediation-SetIAMPasswordPolicy
28	30	AWSConfigRemediation-ConfigureS3PublicAccessBlock

Table 13. AWS Runbooks applicable to `mega-mesh`

No.	Idx	Scenario
1	84	AWSConfigRemediation-EnableCloudTrailEncryptionWithKMS
2	13	AWS-SetRequiredTags
3	59	AWSConfigRemediation-EnableSystemsManagerSessionManagerAuditLogsToS3
4	78	AWSFleetManager-CreateUser
5	18	AWS-ArchiveS3BucketToIntelligentTiering
6	40	AWSSupport-RemediateLambdaS3Event
7	51	AWS-AttachExcludeConditionToS3DenyPolicies
8	47	AWS-EnableS3BucketKeys
9	62	AWSSupport-AnalyzeEBSResourceUsage
10	24	AWSConfigRemediation-UpdateXRayKMSKey
11	65	AWS-ConfigureS3BucketVersioning
12	72	AWSConfigRemediation-RestrictBucketSSLRequestsOnly
13	37	AWSConfigRemediation-ConfigureS3BucketPublicAccessBlock
14	46	AWS-RemediateSSMAgentVPCAttributes
15	1	AWS-EnableNeptuneDbBackupRetentionPeriod
16	20	AWS-EnableS3BucketEncryption
17	56	AWS-JoinDirectoryServiceDomain
18	28	AWS-EnableCloudTrailCloudWatchLogs

Listing 1. Prompt for assessing whether a runbook scenario is applicable to an IaC project.

```
You are an expert in Terraform and AWS infrastructure management.
You need to determine if a Terraform scenario can be meaningfully applied
     to mutate the current AWS infrastructure.

Analyze both inputs carefully:
- <STATE>: Outputs from the command `terraform state list` describing the
     resources under the current AWS infrastructure
- <SCENARIO>: A Terraform scenario description that explains how to
    implement a specific configuration change or infrastructure mutation

Each scenario aims to mutate the terraform infrastructure in a meaningful
     way by:
- Adding new resources or configurations
- Modifying existing resource properties
- Removing or deleting resources
- Implementing security improvements
- Enhancing monitoring or logging capabilities
- Optimizing resource configurations

Determine if the scenario can meaningfully mutate this infrastructure by
    checking:
1. If the resources mentioned in the scenario exist in the current
    Terraform state or can be logically added/removed
2. If the scenario would result in a small change to the infrastructure,
    typically involves only few resources
3. If the scenario's requirements are compatible with the existing
    infrastructure architecture
4. If the mutation can be safely implemented without creating security
    risks or breaking existing functionality
5. If the scenario adds value by improving security, performance,
    monitoring, operational capabilities, or cost optimization through
    resource removal

A scenario is applicable if it can meaningfully transform or enhance the
    existing infrastructure state through addition, modification, or
    removal of resources.

Output Format:
<REASON>A concise explanation of why the scenario is or is not
    meaningfully applicable to mutate this infrastructure, focusing on
    the value and feasibility of the proposed changes</REASON>
<LABEL>Yes</LABEL> if the scenario can meaningfully mutate the
    infrastructure, <LABEL>No</LABEL> if not applicable
```

A.2 Manually Crafted Mutations

For each project, we hand-crafted a small set of mutations to complement the automatically generated ones. The project-specific cases are summarized for `lab12` in Table 14, `flask` in Table 15, `ssm3` in Table 16, and `live-score` in Table 17. Each mutation was tested to ensure it was both deployable and revertible. The `mega-mesh` project was excluded from large-scale mutation testing due to high deployment costs, and only a small number of mutations were evaluated.

Table 14. Manually Crafted Mutations for the 1ab12 project.

No.	Manually Crafted Mutations for 1ab12	Description
1	_delete_route_table_associations	The mutation removes route table associations between subnets and route tables in an egress VPC network configuration, eliminating the connections that direct traffic flow between network components.
2	_delete_huge_networks	The mutation removes route tables, route associations, and Transit Gateway route table associations, eliminating the networking infrastructure that connects the VPCs.
3	_update_tgroute2_recreate	Changes the destination CIDR block for a transit gateway route from 10.0.0.0/8 to 11.0.0.0/8 and removes lifecycle create_before_destroy settings from security groups.
4	_update_sg_name_recreate	This mutation updates the name of a security group from "VPC_B_SG" to "VPC_B_SG-updated" and cause resource to be recreated.
5	_scale_up_vm_vpcs	This diff modifies the infrastructure to scale up from single instances to multiple instances in each VPC by adding count variables, removing hard-coded IP addresses, and updating resource references to support multiple instances.
6	_delete_eip_NGW_2	The mutation removes a second Network Address Translation (NAT) gateway along with its associated elastic IP, route table, and route table association, eliminating redundancy in the network architecture.
7	_delete_update_vpc_ec2_getw	The mutation consolidates networking infrastructure by removing a NAT gateway, limiting security group access to a specific IP range, removing lifecycle rules, and deleting an EC2 instance connect endpoint.
8	_delete_Bastion_Endpoint_VPC_B	The change removes the EC2 instance connect endpoint for VPC B along with its security group lifecycle configurations.
9	_rediect_vpca_and_b_for_ec2	Swaps the network interfaces between two instances and removes lifecycle management blocks from security groups.

Table 15. Manually Crafted Mutations for the `flask` project.

No.	Manually Crafted Mutations for `flask`	Description
1	_delete_autoscaling_alarms_table	The mutation removes the DynamoDB table used for storing autoscaling alarms configuration while preserving the core application infrastructure including networking, IAM roles, ECS cluster, and CloudTrail logging capabilities.
2	_delete_music_table	The mutation removes the DynamoDB music table resource while keeping the Flask application that references it, potentially causing application errors as the table no longer exists for storing and retrieving music data.
3	_update_role_name_recreations	The mutation involves relocating a Flask microservices application with its associated infrastructure from a nested directory structure to the root directory, updating the IAM role name from "ecs_service_role" to "ecs_service_role_updated".
4	_update_scaling_ecs_count	The mutation updates the desired capacity for an ECS (Elastic Container Service) cluster from 1 to 5 instances, increasing the application's scalability to handle more traffic.
5	_delete_aws_route_table_associations	The mutation removed AWS route table associations while setting up a complete Flask microservice infrastructure with CloudTrail logging, VPC networking, ECS clusters, load balancing, and DynamoDB tables.
6	_delete_polcy2	This mutation creates a Flask microservice application deployed on AWS with CloudTrail logging capabilities, removing duplicate endpoint functions in the app.py file.
7	_delete_aws_network_acl_rules	The mutation removes Network Access Control List (NACL) rules from a network configuration, effectively removing the firewall rules that control traffic to and from subnets in a VPC infrastructure.
8	_delete_scaling_policys	This mutation removes auto-scaling policies from an AWS infrastructure while setting up a Flask microservice application with necessary infrastructure components like networking, IAM roles, DynamoDB database, and CloudTrail monitoring.
9	_delete_aws_route_pub	The mutation recreates an application infrastructure by removing a public route from a network configuration, which could improve security by limiting external access to resources.
10	_delete_aws_network_acl	The mutation moves a complete Flask microservice infrastructure from multiple AWS-specific folders to a single root directory, consolidating the implementation of a web application with DynamoDB backend, network infrastructure, and monitoring components.
11	_delete_aws_route_private_1	The mutation removes an AWS route for a private subnet (private_1), relocating all infrastructure files from various AWS configuration patterns into a simplified Flask microservice environment.
12	_update_cloudwatch_logs_enabled	The mutation sets up a Flask microservices application with CloudTrail logging enabled and configures CloudWatch Logs integration for monitoring and auditing purposes.

Table 16. Manually Crafted Mutations for the ssm3 project.

No.	Manually Crafted Mutations for ssm3	Description
1	_add_ec2_detailed_monitoring	This mutation adds detailed CloudWatch monitoring to EC2 instances, improving operational visibility with 1-minute granularity metrics for better performance analysis and faster issue detection.
2	_update_s3_policy_readonly	This mutation updates an S3 bucket policy to remove write permissions, establishing read-only access for improved security testing.
3	_delete_kms_key_s3_encryption	This mutation deletes a customer-managed KMS key and downgrades S3 bucket encryption from KMS to standard AES256 to test security configuration drift scenarios.
4	_delete_iam_instance_profile	This mutation removed the IAM instance profile from EC2 instances, resulting in the loss of Systems Manager management capabilities and automated maintenance functions.
5	_delete_iam_mw_policy	This mutation successfully removed an IAM policy and its role attachment, reducing permissions for maintenance window operations.
6	_delete_ssh_security_rule	The mutation deletes a security group rule that previously allowed SSH access from a specific IP address, blocking remote SSH access to EC2 instances.
7	_update_s3_lifecycle_retention	The mutation shortened the retention period for SSM output logs in an S3 bucket from 90 days to 30 days, reducing storage costs but potentially affecting compliance and debugging capabilities.
8	_update_s3_bucket_versioning	This mutation adds versioning capabilities to a storage bucket to enhance data protection and improve recovery options against accidental deletions or modifications.
9	_update_s3_encryption_aes256	The report summarizes the simplification of S3 bucket encryption from customer-managed KMS to AWS-managed AES256 to reduce operational complexity while maintaining data security.
10	_delete_s3_lifecycle_config	This mutation deleted the automatic log cleanup configuration, which will cause SSM output logs to accumulate indefinitely instead of being deleted after 90 days.
11	_update_maintenance_window_schedule	The mutation report describes a change in system maintenance frequency from daily to weekly execution on Sundays.
12	_update_ec2_instance_type	This mutation updates the instance type of EC2 virtual machines from a smaller to a larger size, increasing the compute capacity of the infrastructure.
13	_update_alb_listener_port_8080	The mutation updated an application load balancer's listening port from 80 to 8080 for standardization purposes, affecting both the load balancer configuration and associated security group rules.
14	_delete_ssm_graceful_reboot	This mutation removed a Systems Manager document used for graceful server rebooting and updated related automation tasks to reference a different document instead.
15	_update_ec2_instance_count	This modification reduced the number of EC2 instances from 3 to 2 by changing the instance count parameter, resulting in the removal of one instance and its associated load balancer target group attachment.
16	_update_iam_user_tags	This mutation updates tags on an IAM user to test tag change detection and reconciliation processes in infrastructure management.
17	_update_alb_health_check	This mutation updated an application load balancer's health check configuration to use more conservative thresholds and longer intervals, which could result in slower detection of health status changes.
18	_update_security_group_cidr_corporate	The update expanded network access permissions from a single IP address to a broader corporate network range (10.0.0.0/16) for both SSH and ALB ingress traffic, improving operational flexibility while maintaining security boundaries.

Table 17. Manually Crafted Mutations for the `live-score` project.

No.	Manually Crafted Mutations for `live-score`	Description
1	_delete_ssh_security_group	This mutation removed a custom security group allowing unrestricted SSH access and replaced it with a more restrictive default security group to improve the system's security posture.
2	_delete_lambda_event_source_mapping	This mutation removed the connection between a DynamoDB stream and Lambda function, disabling the automatic triggering of processor creation when database changes occur.
3	_update_sns_topic	The mutation enhanced a notification topic by adding encryption and a descriptive display name to improve security and usability of the scorecard update system.
4	_update_cloudwatch_schedule	This modification changes the execution schedule of the get-scorecard-urls Lambda function from a seasonal weekend-based pattern to a consistent daily weekday morning schedule that runs year-round.
5	_update_sqs_queue	The configuration changes extended the message retention period and visibility timeout for a queue to improve reliability and processing time for web notification messages.
6	_delete_api_authorizer_lambda	This mutation removed authentication from the notifications API Gateway by eliminating the API authorizer Lambda function and its associated resources, making all notification endpoints publicly accessible without authentication.
7	_delete_update_sanity_module	The mutation removed the external Sanity CMS integration functionality from a live scores system while preserving all core scoring functionality.
8	_update_lambda_runtime_nodejs16x	This mutation downgraded the Node.js runtime version from 18.x to 16.x for all 16 Lambda functions in a live scores system while maintaining operational continuity.
9	_update_dynamodb_connections_encryption	Added server-side encryption to the live-score-connections database table to protect stored connection data through encryption at rest.
10	_delete_cloudwatch_event_rule	The mutation removed the scheduled automatic triggering mechanism for the get-scorecard-urls Lambda function while maintaining the function itself, eliminating automated cricket scorecard URL fetching during the season.
11	_delete_sqs_scorecard_html	This mutation removed an unused SQS queue resource and updated all dependent components to handle its absence gracefully through conditional logic.
12	_delete_api_gateway_authorizer	This modification removed authentication requirements from the API Gateway notifications endpoints, enabling unauthenticated access to test security configuration drift scenarios.
13	_delete_api_route	This mutation removed a WebSocket connection route from the API gateway to simulate and test how the system would respond when a critical connection endpoint becomes unavailable.
14	_update_dynamodb_subscriptions_ttl_gsi	This update adds time-based record expiration and a date-based lookup index to a database table for improved query performance, automatic data cleanup, and cost optimization.
15	_delete_dynamodb_connections	The report summarizes the removal of a DynamoDB table used for WebSocket connection tracking, which eliminated the ability to track and manage connection states while preserving core system functionality.
16	_update_cloudwatch_log_retention	The report summarizes a successful update to CloudWatch log group retention policies across multiple Lambda functions, extending the retention period from 14 to 30 days to improve log management and compliance capabilities.
17	_delete_s3_scorecards_bucket	This mutation removed the S3 scorecards bucket and related configurations from a live scores project, eliminating scorecard storage functionality while maintaining system stability through conditional logic.
18	_update_security_group	The mutation restricts SSH access from the public internet to private network ranges only, improving security by reducing the attack surface.
19	_update_sqs_encryption	The mutation adds encryption at rest to a web notification queue system using AWS KMS, enhancing message security without disrupting service.