

Cloud-Gym: A Benchmark for LLM-Based Infrastructure-as-Code Repair

Author Name
Affiliation
email@example.com

Abstract

Infrastructure-as-Code (IaC) configurations are critical to modern cloud deployments, yet LLM-based repair of broken IaC remains underexplored due to the lack of standardized benchmarks. We present **Cloud-Gym**, a benchmark and training data generation framework for evaluating LLM repair of Terraform and CloudFormation configurations. Cloud-Gym features a 37-fault taxonomy spanning 8 categories (syntactic, reference, semantic, dependency, provider, security, cross-resource, and intrinsic function faults), a programmatic inversion engine that generates validated broken configurations from gold instances, and an evaluation harness using the $\text{pass}@k$ metric. We collect gold configurations from GitHub, the Terraform Registry, and AWS sample repositories, generate thousands of training pairs via both rule-based and LLM-assisted fault injection, and establish baseline repair rates using open-weight models. Our results show that while models handle syntactic faults reasonably well, semantic and cross-resource faults remain challenging, motivating further research in IaC-specific code repair.

1 Introduction

Infrastructure-as-Code (IaC) has become the standard for managing cloud resources, with tools like Terraform and CloudFormation enabling declarative specification of complex infrastructure. As IaC configurations grow in complexity, misconfigurations become a significant source of deployment failures, security vulnerabilities, and operational incidents.

Recent advances in large language models (LLMs) have shown promise for automated code repair across general-purpose programming languages. However, IaC repair presents unique challenges:

- **Domain-specific semantics:** IaC faults span syntactic errors, cross-resource dependency issues, and cloud provider-specific constraints that differ fundamentally from traditional programming bugs.
- **Validation complexity:** Correctness requires interaction with cloud provider APIs and validation tools (e.g., `terraform validate`, `cfn-lint`), not just syntactic well-formedness.
- **Lack of benchmarks:** No standardized benchmark exists for evaluating LLM-based IaC repair, making it difficult to compare approaches or measure progress.

We address these gaps with **Cloud-Gym**, a framework that provides:

1. A **37-fault taxonomy** across 8 categories, covering both Terraform/HCL and CloudFormation/YAML fault types.

2. A **programmatic inversion engine** that generates validated broken configurations from gold (valid) instances through targeted fault injection.
3. A **training data pipeline** producing thousands of (broken, gold, error) triples for supervised fine-tuning.
4. An **evaluation harness** with $\text{pass}@k$ metrics and stratified reporting by fault category, difficulty, and IaC format.

2 Related Work

IaC Analysis and Repair. Prior work on IaC analysis has focused on static analysis tools [AWS, 2023], security scanning [Bridgecrew, 2023], and policy enforcement. However, automated *repair* of IaC configurations using learned models remains largely unexplored. Existing tools can detect issues but require manual intervention for fixes.

LLM-Based Code Repair. The application of LLMs to automated program repair has gained significant attention. Benchmarks like SWE-bench [Jimenez et al., 2024] evaluate end-to-end issue resolution, while HumanEval [Chen et al., 2021] and MBPP [Austin et al., 2021] focus on code generation. CLI-Gym [cli, 2024] provides a reinforcement learning environment for command-line tasks. Cloud-Gym extends this line of work to the IaC domain.

Fault Injection and Mutation Testing. Mutation testing [Jia and Harman, 2011] systematically introduces faults to evaluate test suite adequacy. Cloud-Gym’s inversion engine draws on mutation testing principles but targets IaC-specific fault patterns validated by real infrastructure tools.

3 Method

3.1 Fault Taxonomy

We define a taxonomy of 37 fault types organized into 8 categories, designed to cover the spectrum of real-world IaC misconfigurations. Table ?? summarizes the taxonomy.

- **Syntactic** (7 faults): Parse-level errors including missing braces, type mismatches, invalid syntax, and missing required arguments.
- **Reference** (6 faults): Broken variable references, non-existent resource references, undefined parameters, and invalid module sources.
- **Semantic** (7 faults): Logically invalid values such as wrong resource types, malformed AMI IDs, invalid CIDR blocks, and non-existent regions.
- **Dependency** (4 faults): Circular dependencies and missing ordering constraints.
- **Provider** (2 faults): Missing provider configurations and impossible version constraints.
- **Security** (4 faults): Overly permissive network rules and missing encryption.
- **Cross-Resource** (3 faults): Mismatched VPC/subnet references and wrong availability zones.

- **Intrinsic** (4 faults, CF-only): Malformed `!Sub`, out-of-bounds `!Select`, undefined conditions, and invalid `!Join`.

3.2 Inversion Engine

The inversion engine transforms gold (valid) configurations into broken ones via targeted fault injection. Given a gold config c_g and fault type f , the engine produces a broken config c_b such that:

1. $c_b \neq c_g$ (the config was actually modified),
2. $\text{validate}(c_b) = \text{FAIL}$ (the modification breaks validation), and
3. The diff between c_g and c_b is minimal (single-fault injection).

Programmatic Injection. For Terraform/HCL, we use a parse-then-regex approach: `python-hcl2` parses the config for structural analysis, then targeted string manipulation applies the fault. For CloudFormation, we use YAML/JSON round-trip mutation. Each fault type has a dedicated injector function.

Agentic Injection. We also employ local LLMs (via Ollama) for diverse fault generation, with quality gates ensuring similarity (≥ 0.7) and minimal diff ($< 20\%$ lines changed).

3.3 Evaluation Protocol

We evaluate repair models using the $\text{pass}@k$ metric [Chen et al., 2021]:

$$\text{pass}@k = \mathbb{E} \left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}} \right]$$

where n is the number of attempts and c is the number of correct repairs. A repair is correct if the repaired config passes validation with zero errors.

4 Dataset

4.1 Gold Configuration Collection

We collect gold (valid) IaC configurations from three sources:

- **GitHub:** Public repositories containing Terraform and CloudFormation files, filtered by minimum resource count and validated.
- **Terraform Registry:** Official modules from the Terraform Registry.
- **AWS Samples:** CloudFormation templates from AWS’s official sample repositories.

All collected configurations are validated using `terraform validate` (for Terraform) or `cfn-lint` (for CloudFormation). Only configurations that pass validation are retained as gold instances.

4.2 Training Data Generation

From the gold configurations, we generate training pairs using the inversion engine (Section 3.2). For each gold config, we apply N programmatic fault injections via stratified sampling across categories, plus M agentic injections for diversity.

The resulting dataset is split by gold config hash to prevent leakage: all variants derived from the same gold config appear in the same split (train/val/test at 80/10/10).

4.3 Benchmark Curation

From the test split, we curate a balanced benchmark subset with the following criteria:

- Single-fault configurations only
- Minimum 10-line configs (non-trivial)
- Balanced across fault categories and difficulty levels
- Deduplicated per gold config

5 Experiments

5.1 Baseline Models

We evaluate two open-weight models in a zero-shot repair setting:

- **DeepSeek-R1 1.5B**: A small reasoning model, serving as a performance floor.
- **Qwen2.5-Coder 7B**: A code-specialized model, serving as the main baseline.

Both models are run locally via Ollama. The repair prompt provides the broken configuration and its validation errors, asking the model to return only the fixed configuration.

5.2 Results

5.3 Analysis

6 Conclusion

We presented Cloud-Gym, a benchmark and data generation framework for evaluating LLM-based repair of Infrastructure-as-Code configurations. Our 37-fault taxonomy covers the spectrum of real-world IaC misconfigurations across Terraform and CloudFormation. The programmatic inversion engine generates validated training pairs at scale, and the evaluation harness provides fine-grained pass@ k metrics.

Our baseline results indicate that current open-weight models show varying repair capability across fault categories, with syntactic faults being most amenable to repair and semantic/cross-resource faults posing the greatest challenge. This gap motivates future work in IaC-specific model training and more sophisticated repair strategies.

Future Work.

- Fine-tuning models on Cloud-Gym training data
- Extending the taxonomy to additional IaC tools (Pulumi, CDK, Ansible)
- Multi-fault repair scenarios
- Integration with CI/CD pipelines for real-time repair suggestions

References

Cli-gym: A framework for interactive cli agent evaluation. 2024. Preprint.

Jacob Austin, Augustus Odena, Maxwell Nye, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

AWS. cfn-lint: Cloudformation linter. <https://github.com/aws-cloudformation/cfn-lint>, 2023.

Bridgecrew. Checkov: Policy-as-code for infrastructure. <https://github.com/bridgecrewio/checkov>, 2023.

Mark Chen, Jerry Tworek, Heewoo Jun, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Yue Jia and Mark Harman. An analysis and survey of the development of mutation testing. *IEEE Transactions on Software Engineering*, 37(5):649–678, 2011.

Carlos E Jimenez, John Yang, Alexander Wettig, et al. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2024.