SWE-InfraBench: Evaluating Language Models on Cloud Infrastructure Code

Anonymous Author(s)
Affiliation
Address
email

Abstract

Infrastructure-as-code (IaC) is critical for cloud reliability and scalability, yet LLM capabilities in this domain remain underexplored. Existing benchmarks focus on declarative tools like Terraform and full-code generation. We introduce SWE-InfraBench, a dataset of realistic incremental edits to AWS CDK repositories from real-world codebases. Each task requires modifying existing IaC based on natural language instructions, with correctness verified by passed tests. Results show current LLMs struggle: the best model (Sonnet 3.7) solves 34% of tasks, while reasoning models like DeepSeek R1 reach only 24%.

1 Introduction

Infrastructure as Code (IaC) enables scalable cloud management by treating infrastructure as software [18; 15; 14]. Declarative tools like Terraform specify end states, while imperative ones like AWS CDK use general-purpose languages to support logic and incremental change [2; 3]. Because enterprise workflows are iterative and closely tied to application code, we focus on CDK.

Existing LLM benchmarks emphasize code completion or debugging [10; 6; 16], leaving IaC editing largely unexplored. Imperative IaC tasks are especially challenging due to stateful codebases and the need for domain expertise.

We present SWE-InfraBench, a benchmark of 100 tasks from 34 real AWS CDK projects. Each task includes a repository, a natural-language modification instruction, and unit tests. Unlike prior IaC datasets [12; 20], SWE-InfraBench targets imperative CDK and emphasizes incremental modifications over full synthesis.

Our contributions are:

1. A dataset of 100 IaC editing tasks with instructions and unit tests.
2. A pipeline for generating and validating new tasks from arbitrary CDK repositories.
3. Baseline evaluations of diverse LLMs, including multi-turn agents with error feedback and RAG.

2 Related Work

LLMs for Infrastructure-as-Code Early studies applied LLMs to IaC generation (e.g., Ansible YAML [11; 17]) and recent benchmarks such as CloudEval-YAML [20] and IaC-Eval [12]. Results show LLMs perform far worse on IaC than on general programming (e.g., GPT-4 at 19.4% on IaC-Eval vs. 86.6% on EvalPlus). These datasets, however, emphasize declarative frameworks and full code generation. Our SWE-InfraBench instead targets incremental edits in imperative AWS CDK projects. LIG-MM [13] explores symbolic reasoning but not codebase-level modification.

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

Code Generation and Editing Benchmarks General code benchmarks have evolved from single functions (HumanEval [5]) to multi-file reasoning (CrossCodeEval [7]) and incremental tasks (SWE-bench, SWE-PolyBench [10; 19]). Others address efficiency (Mercury [8]) or instructional editing (CodeEditorBench [9], CanItEdit [4]). Our work is the first to focus specifically on IaC editing in enterprise-scale AWS CDK repositories.

3 SWE-InfraBench

SWE-InfraBench is a collection of AWS CDK codebases paired with natural-language modification instructions and unit tests. The task for an LLM is to update the codebase so the new functionality passes all tests. On average, each task includes 10 context files (30k characters). Further dataset statistics are in Appendix A and B. CDK compiles to CloudFormation templates [1], enabling automated validation without deployment, making it a practical basis for benchmarking IaC.

3.1 Benchmark Construction

We built the dataset via a three-stage pipeline (see Appendix D). **Stage I: Repository Selection.** We curated IaC repositories (open-source and custom; licensing in Appendix C). An LLM generated candidate tasks (prompt, canonical solution, tests), which were then reviewed. **Stage II: Critique and Refinement.** Candidates were validated on their canonical solution and refined with critic LLMs that assessed alignment, generality, and test quality. Feedback guided automatic or manual improvements. **Stage III: Expert Review.** Human experts finalized tasks, ensuring fairness, coverage, and challenge level.

The final dataset stores each task as JSON with context files (masked), prompt, canonical solution, tests, and CDK version. Task format and example prompts appear in Appendix E.2, E.1.

3.2 Dataset Collection Challenges

Most open-source IaC projects lacked tests, which we had to design manually. LLM assistance accelerated generation, but expert oversight was crucial. Engineers averaged five validated tasks/day. By releasing both the benchmark and pipeline, we enable scalable extension with human effort concentrated on validation.

3.3 Problem Definition

A model receives the full CDK codebase and a natural-language modification instruction (e.g., add an event-driven pipeline). Its output is integrated into the codebase and evaluated by the provided tests. We report pass@k [5], task correctness (equivalent to pass@1 for single-trial), generation success (valid, integrable outputs), and the proportion of passed tests across all tasks.

4 Experimental Results

We evaluate 20 state-of-the-art models on 100 SWE-InfraBench tasks. Each model gets a single attempt per task; solutions are integrated into the repository and validated by tests on the resulting CloudFormation output.

SWE-InfraBench is challenging: even top models solve under 35% of tasks, reflecting the domain knowledge, syntax precision, and context reasoning required.

Table 1 reports results for the metrics in Section 3.3. The Claude Sonnet family performs best. Reasoning models (e.g., Gemini 2.5 Pro, DeepSeek R1, OpenAI o3, o4-mini) generally outperform non-reasoning models (e.g., GPT-4.1, Claude 3 Haiku, Gemini 2.0 Flash). Generation success is high ($>$94%), but full correctness varies widely, indicating challenges in implementing functionality rather than following format. Tasks from open-source donors are not easier than custom-developed ones (Appendix I).

Table 1: Performance metrics for proprietary and open-source LLMs on IaC generation tasks. Models are grouped by source type and provider, and sorted from the most recent to the oldest variant within each group. † indicates LLMs executed with reasoning capabilities. ✝ Claude 3.7 was executed without extended reasoning. Top results are in **bold**.

Source	Company	Model	Correctness	Generation Success	Passed Tests Share
Proprietary	Anthropic	Claude 3.7 Sonnet ✝	**34%**	**100%**	**53.1%**
		Claude 3.5 Sonnet V2	32%	**100%**	47%
		Claude 3.5 Sonnet	29%	**100%**	47.9%
		Claude 3 Haiku	8%	88%	10.7%
	Google	Gemini 2.5 Pro (03-25 Preview) †	29%	97%	42.5%
		Gemini 2.5 Pro (05-06 I/O Edition Preview) †	29%	95%	41.1%
		Gemini 2.0 Flash Lite	5%	79%	11.5%
		Gemini 2.0 Flash	4%	43%	6.9%
	OpenAI	OpenAI o3 †	23%	**100%**	30.8%
		OpenAI o4-mini †	23%	99%	32.1%
		GPT-4.1	18%	**100%**	26.4%
		GPT-4o Mini	4%	84%	7%
Open-Source	DeepSeek	DeepSeek R1 †	24%	**100%**	34.9%
	Mistral	Mistral Large	14%	89%	20.3%
		Codestral	9%	96%	15%
	Meta	LLaMA 3.1 405B Instruct	9%	97%	13%
		LLaMA 3.1 70B Instruct	3%	97%	7.7%
		LLaMA 4 Maverick 17B Instruct	8%	94%	13%
		LLaMA 4 Scout 17B Instruct	2%	76%	2.4%
	Alibaba	Qwen2.5 72B Instruct Turbo	0%	94%	2.7%

Table 2: Model performance with multiple independent trials. Average Correctness = average success across all trials (all tests passed). pass@k = probability of solving the task in at least one of k attempts.

Model	Average Correctness	pass@1	pass@2	pass@5
Claude 3.7 Sonnet	**28.8%**	**34%**	36%	41%
Gemini 2.5 Pro (03-25 Preview)	26.4%	28%	**38%**	**47%**
DeepSeek R1	24.6%	24%	33%	46%
OpenAI o3	22.8%	23%	30%	45%
LLaMA 3.1 405B Instruct	6.8%	8%	11%	13%

4.1 Consistency Analysis

We run five independent trials on a subset of models to assess reliability. Higher pass@k increases the chance of solving a task at least once, but average correctness across attempts remains below 30%, indicating limited consistency (Table 2; Appendix J).

4.2 Error Type Distribution

Across 20 LLMs, common failures include incorrect CDK construct properties and other syntax issues, sometimes due to outdated knowledge of newer CDK versions (Figure 1; Appendix K). These patterns suggest that iterative, agent-like workflows with error feedback and documentation access can help.

4.3 Multi-Turn Agent Performance

We implement a two-turn agent: after an initial failure, models receive error messages and test results; we compare low-verbosity (V_L) and high-verbosity (V_H) feedback, and a RAG variant using retrieved AWS docs (see Table 6 for specification). Results (Table 3) show substantial gains: Claude 3.7 Sonnet reaches 64% with V_H (+30 points over one-turn). RAG can further help (Claude 3.5 Sonnet V2 up to 65%), though effects vary by model (e.g., GPT-4.1 at 26% with RAG vs. 48% standard two-turn). Improvements with V_H are consistent; model capacity modulates gains (e.g., modest improvements for smaller models like Mistral Large).

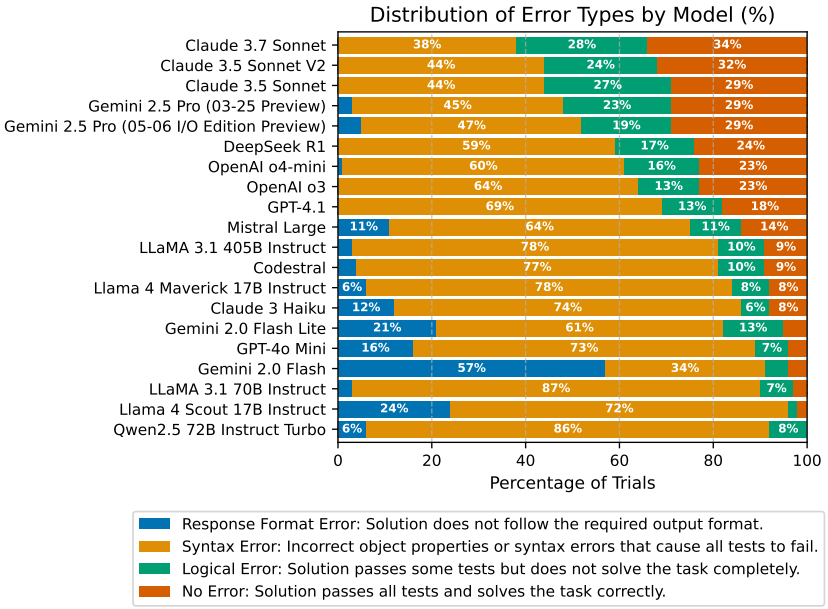

Figure 1: Error type distribution across trials. *Syntax errors* dominate (40–85%), often from incorrect CDK property usage. *Logical errors* (5–30%) occur when code is valid but fails requirements or doesn't capture context correctly. *Response format errors* are rare, with Claude and OpenAI models showing strong format adherence.

Table 3: Comparison of One-Turn vs. Two-Turn LLM Performance (with and without RAG).

Model	**Correctness** (↑)						**Passed Tests Share** (↑)					
	One-Turn	Two-Turn V_L	Two-Turn V_H	Two-Turn + RAG V_L	Two-Turn + RAG V_H	Best	One-Turn	Two-Turn V_L	Two-Turn V_H	Two-Turn + RAG V_L	Two-Turn + RAG V_H	Best
Claude 3.7 Sonnet	34.0%	44.0%	64.0%	51.0%	60.0%	**64.0%**	53.1%	49.5%	71.6%	60.6%	66.1%	**71.6%**
Claude 3.5 Sonnet V2	32.0%	52.0%	56.0%	52.0%	65.0%	**65.0%**	47.0%	60.9%	66.1%	65.5%	74.2%	**74.2%**
Gemini 2.5 Pro (03-25 Preview)	29.0%	41.0%	40.0%	41.0%	33.0%	**41.0%**	42.5%	41.5%	41.4%	41.3%	33.0%	**42.5%**
DeepSeek R1	24.0%	40.0%	43.0%	45.0%	39.0%	**45.0%**	34.9%	49.0%	51.7%	51.9%	49.5%	**51.9%**
GPT-4.1 (2025-04-14)	18.0%	40.0%	48.0%	46.0%	26.0%	**48.0%**	26.4%	48.0%	55.9%	56.7%	30.9%	**55.9%**
Mistral Large	14.0%	21.0%	23.0%	14.0%	17.0%	**23.0%**	20.3%	24.2%	28.6%	17.9%	23.7%	**28.6%**
LLaMA 4 Maverick 17B Instruct	8.0%	15.0%	18.0%	21.0%	21.0%	**21.0%**	13.0%	20.5%	23.7%	25.6%	27.3%	**27.3%**

Correctness: Percentage of completions where the solution passed all test cases.
Passed Tests Share: Average percentage of unit tests passed, including partial completions.
Two-Turn + RAG: Two-step prompting approach with retrieval-augmented generation support.
V_L/V_H: Low/High verbosity configurations.

Caveats: detailed test tracebacks may leak property values, making V_H results an optimistic upper bound, since such detailed tests are mostly unavailable in practice. DeepSeek R1 also consumes many tokens for reasoning, requiring higher output limits, and multi-turn or RAG approaches increase both token usage and latency. Overall, detailed feedback improves refinement, though RAG effectiveness remains model-dependent.

5 Conclusion

We introduced SWE-InfraBench, the first benchmark for evaluating LLMs on imperative IaC modification in AWS CDK. Tasks require natural-language interpretation, cloud resource reasoning, and code changes validated by tests. Current models perform poorly (best single-attempt 34% with Claude 3.7), though multi-turn approaches improve results. Future work includes extending beyond Python CDK and incorporating richer evaluation methods such as static analysis, cost modeling, or LLM-as-a-Judge to capture efficiency and security aspects.

References

[1] Amazon Web Services: What is aws cloudformation? - aws cloudformation (2025), https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html, accessed: 2025-05-05

[2] Amazon Web Services: What is the aws cdk? - aws cloud development kit (aws cdk) v2 (2025), https://docs.aws.amazon.com/cdk/v2/guide/home.html, accessed: 2025-05-05

[3] Ava: Terraform vs AWS CloudFormation vs AWS CDK. https://kudulab.io/posts/2023-2-terraform-vs-cdk-vs-cloudformation/ (2023), accessed: 2025-05-09

[4] Cassano, F., Li, L., Sethi, A., Shinn, N., Brennan-Jones, A., Ginesin, J., Berman, E., Chakhnashvili, G., Lozhkov, A., Anderson, C.J., et al.: Can it edit? evaluating the ability of large language models to follow code editing instructions. arXiv preprint arXiv:2312.12450 (2023)

[5] Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H.P., et. al, J.K.: Evaluating large language models trained on code (2021)

[6] Chowdhury, N., Aung, J., Shern, C.J., Jaffe, O., Sherburn, D., Starace, G., Mays, E., Dias, R., Aljubeh, M., Glaese, M., Jimenez, C.E., Yang, J., Ho, L., Patwardhan, T., Liu, K., Madry, A.: Introducing swe-bench verified (2024), https://openai.com/index/introducing-swe-bench-verified/, accessed: 2025-03-02

[7] Ding, Y., Wang, Z., Ahmad, W., Ding, H., Tan, M., Jain, N., Ramanathan, M.K., Nallapati, R., Bhatia, P., Roth, D., et al.: Crosscodeeval: A diverse and multilingual benchmark for cross-file code completion. Advances in Neural Information Processing Systems **36**, 46701–46723 (2023)

[8] Du, M., Luu, A.T., Ji, B., Liu, Q., Ng, S.K.: Mercury: A code efficiency benchmark for code large language models. arXiv preprint arXiv:2402.07844 (2024)

[9] Guo, J., Li, Z., Liu, X., Ma, K., Zheng, T., Yu, Z., Pan, D., Li, Y., Liu, R., Wang, Y., et al.: Codeeditorbench: Evaluating code editing capability of large language models. arXiv preprint arXiv:2404.03543 (2024)

[10] Jimenez, C.E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., Narasimhan, K.: Swe-bench: Can language models resolve real-world github issues? arXiv preprint arXiv:2310.06770 (2023)

[11] Kawaguchi, M., Mizutani, K., Iguchi, N.: An implementation of misconfiguration prevention system using language model for a network automation tool. IEICE Proceedings Series **72**(S5-8) (2022)

[12] Kon, P.T., Liu, J., Qiu, Y., Fan, W., He, T., Lin, L., Zhang, H., Park, O.M., Elengikal, G.S., Kang, Y., et al.: Iac-eval: A code generation benchmark for cloud infrastructure-as-code programs. Advances in Neural Information Processing Systems **37**, 134488–134506 (2024)

[13] Liu, C., Wu, X., Feng, Y., Cao, Q., Yan, J.: Towards general loop invariant generation: A benchmark of programs with memory manipulation. Advances in Neural Information Processing Systems **37**, 129120–129145 (2024)

[14] Morris, K.: Infrastructure as Code: Managing Servers in the Cloud (2nd Edition). O'Reilly Media (2020)

[15] Pahl, C., Gunduz, N.G., Sezen, O.C., Ghamgosar, A., El Ioini, N.: Infrastructure as code: Technology review and research challenges. In: Proc. of the 15th Int. Conf. on Cloud Computing and Services Science (CLOSER) (2025)

[16] Prasad, A., Stengel-Eskin, E., Chen, J.C.Y., Khan, Z., Bansal, M.: Learning to generate unit tests for automated debugging. arXiv preprint arXiv:2502.01619 (2025)

[17] Pujar, S., Buratti, L., Guo, X., Dupuis, N., Lewis, B., Suneja, S., Sood, A., Nalawade, G., Jones, M., Morari, A., et al.: Automated code generation for information technology tasks in yaml through large language models. In: 2023 60th ACM/IEEE Design Automation Conference (DAC). pp. 1–4. IEEE (2023)

[18] Quattrocchi, G., Tamburri, D.A.: Infrastructure as code. IEEE Software **40**(1), 37–40 (2023). https://doi.org/10.1109/MS.2022.3212034

[19] Rashid, M.S., Bock, C., Zhuang, Y., Buccholz, A., Esler, T., Valentin, S., Franceschi, L., Wistuba, M., Sivaprasad, P.T., Kim, W.J., et al.: Swe-polybench: A multi-language benchmark for repository level evaluation of coding agents. arXiv preprint arXiv:2504.08703 (2025)

[20] Xu, Y., Chen, Y., Zhang, X., Lin, X., Hu, P., Ma, Y., Lu, S., Du, W., Mao, Z.M., Cai, D., et al.: Cloudeval-yaml: A practical benchmark for cloud configuration generation. Proceedings of Machine Learning and Systems **6**, 173–195 (2024)

Appendix

A SWE-InfraBench Characteristics

SWE-InfraBench tasks contain examples for multiple CDK versions ranging from 2.0.0 to more recent version like 2.189.1, with the majority of tasks suitable for 2.178.2 and newer (see Figure 3). The tasks also vary in number and length of context files, prompt length and canonical solution length. The key dataset statistics are illustarted in Figure 2.

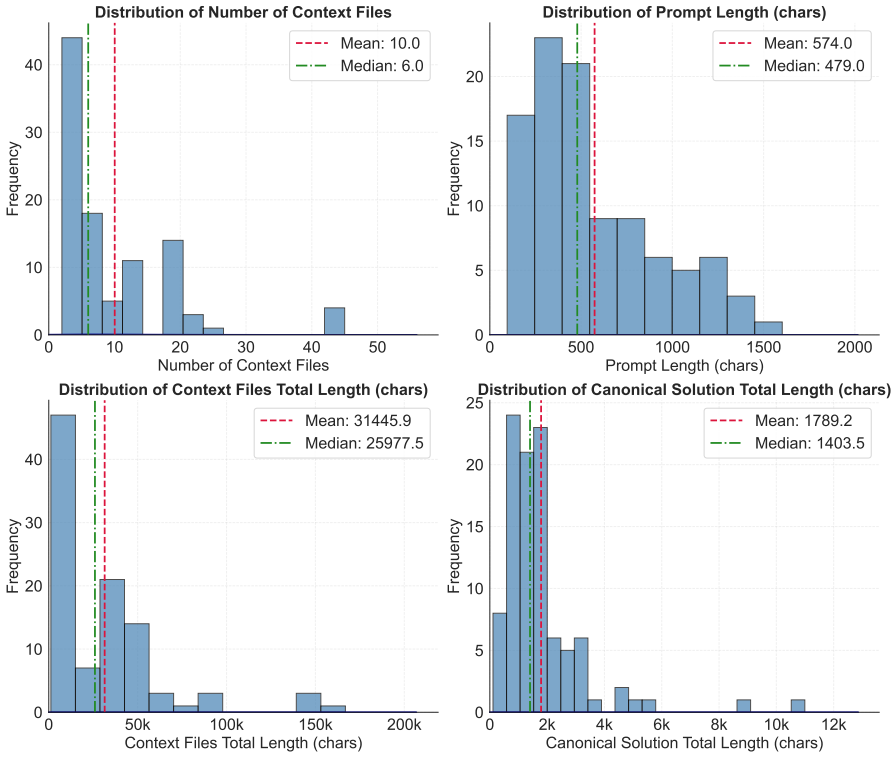

Figure 2: SWE-InfraBench Statistics on Context Size and Solution Length

B CDK Versions Distribution

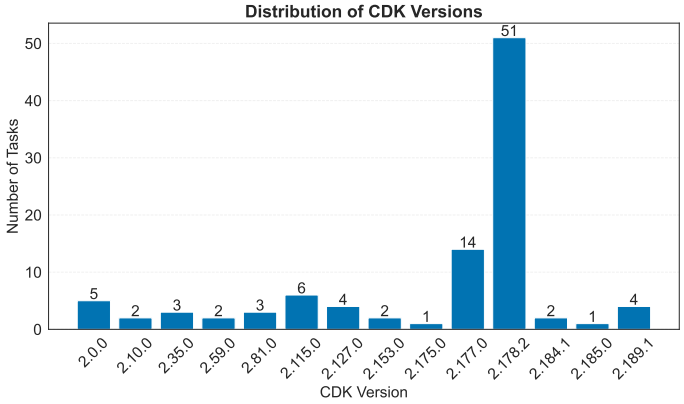

Figure 3: Distribution of AWS CDK Versions Across SWE-InfraBench Tasks

Figure 3 shows the distribution of AWS CDK versions across the SWE-InfraBench tasks. The benchmark predominantly utilizes version 2.178.2, highlighting a concentration on recent library versions, while also including several earlier versions to ensure diversity and represent real-world variability in CDK projects.

C Source Repositories Licences

SWE-InfraBench tasks are derived from open-source repositories and custom-developed sources. Note that original code snippets were substantially modified to create the tasks. See Table 4 for details on open-source repositories.

Table 4: Open-source repositories used as sources for SWE-InfraBench tasks

Repository Name	Source	License
aws-cdk-examples	aws-samples/aws-cdk-examples	Apache-2.0
generative-ai-cdk-constructs-samples	aws-samples/generative-ai-cdk-constructs-samples	Apache-2.0
generative-ai-ml-latam-samples	aws-samples/generative-ai-ml-latam-samples	MIT-0
aws-cdk-lambda-import-export-redshift-ddl	aws-samples/aws-cdk-lambda-import-export-redshift-ddl	MIT-0
amazon-elasticache-demo-using-aws-cdk	aws-samples/amazon-elasticache-demo-using-aws-cdk	MIT-0
deploy-langfuse-on-ecs-with-fargate	aws-samples/deploy-langfuse-on-ecs-with-fargate	MIT-0

All repository URLs are prefixed with `https://github.com/`

D Dataset Construction Pipeline

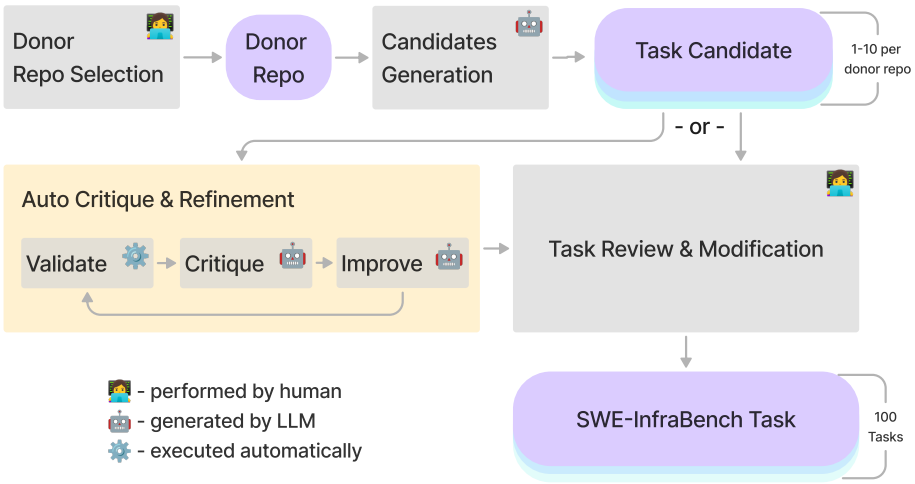

Figure 4: SWE-InfraBench task instances are created from open-source and custom developed IaC repositories in a semi-automated manner with a rigorous human engineers oversight.

The three-stage pipeline used to construct SWE-InfraBench is shown in Figure 4.

E Tasks Preview

E.1 Prompt Examples

SWE-InfraBench includes a diverse set of prompts that test various aspects of infrastructure-as-code generation. See examples of such prompts below:

API Gateway Integration

Generate code to create an API Gateway SampleAPI-EventBridge-Multi-Consumer that integrates with the event producer with proxy integration. Do not add a catch-all route in api routing (use proxy=False). Add a resource named 'items' to the root of the API and create a POST method for this resource.

Bedrock Agent Setup

Create a Bedrock agent using CDK with the following requirements in the CfnAgent construct.
Set the agent name, description, model and instruction from the class attributes.
Also, set the agent resource role ARN from the previously created role and add idle session time of 600 seconds.
We want the draft version to be always in sync via auto preparation and add test alias tags.
Finally include action groups and collaboration if provided and ensure the removal policy destroys the agent when the stack is deleted

WhatsApp IAM Policy

Create CDK code to add IAM policies to a Lambda function, granting permissions for social-messaging, transcribe, and bedrock services.
The policies should have separate policies for
1) allow sending whatsapp messages and getting media from them for all cell numbers in all regions and accounts
2) access to any transcribe action on all resources
3) and access to invoke* all agents, inference profiles and models in oregon

E.2 Task Example

Figure 5 illustrates the structure of a task from SWE-InfraBench. Each task is stored as a JSON file containing the prompt, context files, canonical solution, and tests.

Example Task: API Gateway Integration with EventBridge


```
{
  "task_id": "67ad6ef1-fb3e-45e6-b5e1-83ae385528b5",
  "entry_point": "api-eventbridge-lambda+api_gateway_integration",
  "prompt": "Generate code to create an API Gateway
  ↪  SampleAPI-EventBridge-Multi-Consumer
          that integrates with the event producer with proxy
          ↪  integration.
          Do not add a catch-all route in api routing (use
          ↪  proxy=False).
          Add a resource named 'items' to the root of the API and
          ↪  create a
          POST method for this resource.",
  "cdk_version": "2.178.2",
  "context": {
    "app.py": "#!/usr/bin/env python3\n\nfrom aws_cdk import
    ↪  App\n\nfrom api_eventbridge_lambda.api_eventbridge_lambda
    ↪  import ApiEventBridgeLambdaStack...",
    "api_eventbridge_lambda/api_eventbridge_lambda.py": "from
    ↪  constructs import Construct\nfrom aws_cdk import...",
    "lambda/event_consumer_lambda.py": "import json\nimport
    ↪  logging\n\nlogger = logging.getLogger()...",
    "lambda/event_producer_lambda.py": "import json\nimport
    ↪  boto3\nimport datetime..."
  },
  "canonical_solution": {
    "api_eventbridge_lambda/api_eventbridge_lambda.py": [
      "--- without_solution\n\n+++ with_solution\n\n@@ -102,0 +103,7
      ↪  @@\n\n+        # defines an API Gateway REST API resource
      ↪  backed by our \"atm_producer_lambda\" function.\n\n+
      ↪  api = api_gw.LambdaRestApi(self,
      ↪  'SampleAPI-EventBridge-Multi-Consumer',\n\n+
      ↪  handler=event_producer_lambda,\n\n+
      ↪  proxy=False\n\n+                                )\n\n+
      ↪  items = api.root.add_resource(\"items\")\n\n+
      ↪  items.add_method(\"POST\")  # POST /items\n"
    ]
  },
  "tests": {
    "test_api_gateway_integration.py": "import aws_cdk as cdk\nfrom
    ↪  aws_cdk.assertions import Template, Match..."
  }
}
```


Figure 5: Example task from InfraBench showing the JSON structure. Context and test files are truncated for brevity.

F Prompt Templates

F.1 Critic Prompt Template

The following is the template used for the critic in the InfraBench dataset construction process:

Critic Prompt Template

As an AI assistant specialized in Infrastructure as Code, your task is to critique a dataset item created for an LLM code generation challenge.
You will be given:
1. A repository description
2. The full repository code
3. A pre-suggestion with a specific code section to mask
4. The actual code that was masked
5. The generated human language prompt
6. The generated test file
Your job is to critically evaluate:
1. Whether the prompt accurately describes what needs to be implemented
2. Whether the tests effectively validate all requirements stated in the prompt
3. Whether the prompt and tests are general enough to allow various valid solutions that preserve the functionality of the masked code while remaining compatible with the overall repository structure
REPOSITORY DESCRIPTION

```
[Repository description is provided here]
```

REPOSITORY CONTENT

```
[Repository files are provided here]
```

PRE-SUGGESTION

```
Item Name: [Item name]
File Path: [File path]
Start Line: [Start line]
End Line: [End line]
Complexity: [Complexity]
Rationale: [Rationale]
```

ACTUAL CODE TO BE MASKED

```
[Masked code is provided here]
```

GENERATED PROMPT

```
[Generated prompt is provided here]
```

GENERATED TEST FILE
Path: [Test file path]

```
[Test file content is provided here]
```

GENERATOR NOTES

```
[Generator notes are provided here]
```

CRITIQUE GUIDELINES
Prompt vs Functional Requirements Evaluation
1. Does the prompt clearly describe ALL the necessary functional aspects needed for the masked code?
2. Is it sufficiently detailed for someone to implement the solution correctly without seeing the masked code?
3. Are there any ambiguities or missing requirements that would prevent a correct implementation?
4. Does it avoid revealing the actual implementation details while still being complete?
5. If something can be inferred unequivocally from the repository code, it does not need to be specified in the prompt.

Generality of Tests Evaluation
1. Tests should be general enough so that they pass if a developer or LLM follows the prompt accurately, regardless of the specific implementation details.
2. The prompt should give only the minimum necessary instructions needed to explain the functional requirements, while considering how the tests are built.
3. Tests should verify functionality rather than specific implementation approaches - they should allow for multiple valid solution patterns that fulfill the prompt.
4. Evaluate if tests are overly restrictive by enforcing a particular implementation approach when other valid approaches could fulfill the same requirements.
5. Tests won't be accepted if they don't pass with the original masked code (tested elsewhere) but consider as well if other reasonable implementations would pass.
Test Evaluation
For each test in the test file:
1. What is this test specifically checking for?
2. Is this test testing for something that's explicitly stated in the prompt?
3. A test is valid if it tests integration with functionality that exists elsewhere in the code base (not just the masked section).
4. Is the test appropriately written to verify the requirement?
5. If the test might fail with some valid implementations (including the original masked code), should the prompt be more explicit or should the test be less restrictive?
Test Completeness Evaluation
1. Do the tests collectively verify ALL requirements mentioned in the prompt?
2. Are there any requirements in the prompt that aren't tested?
3. Are there any tests for requirements not mentioned in the prompt?
4. Are all edge cases and error conditions properly tested?
Return the response in this JSON format:

```
{
    "prompt_vs_functional": {
        "explanation": "Detailed explanation of whether the prompt
    accurately describes all necessary functional aspects of the
    masked code",
        "corrections": "Specific corrections with concrete
    implementation suggestions - provide exact wording changes or
    additions to the prompt", // Only include if there are issues
        "example_improvements": "Suggested rewrites of problematic
    sections with specific language", // Only include if there are
    issues
        "accept": true|false  // Conclusion based on the
    explanation
    },
    "generality": {
        "explanation": "Analysis of whether the tests allow for
    multiple valid implementations that satisfy the requirements in
    the prompt",
        "issues": "Identification of any tests that could fail
    with valid implementations that follow the prompt", // Only
    include if there are issues
        "suggested_improvements": "Concrete suggestions for making
    tests more general while still validating functionality", //
    Only include if needed
        "accept": true|false  // Whether the tests are
    sufficiently general
    },
    "tests": {
        "test_name_1": {
            "purpose": "What this test is checking for",
            "coverage": "Explanation of how this relates to prompt
    requirements",
            "suggested_improvements": {
```

```
                "explanation": "Why the test needs improvement",
    // Only include if needed
                "code_snippet": "Complete improved version of the
    test with code fixes", // Provide actual code implementation
                "rationale": "Explanation of why this
    implementation is better"
            }, // Only include if needed
            "accept": true|false  // Conclusion based on the
    critic's analysis of the test quality
        },
        "test_name_2": {
            "purpose": "What this test is checking for",
            "coverage": "Explanation of how this relates to prompt
    requirements",
            "suggested_improvements": {
                "explanation": "Why the test needs improvement",
    // Only include if needed
                "code_snippet": "Complete improved version of the
    test with code fixes", // Provide actual code implementation
                "rationale": "Explanation of why this
    implementation is better"
            }, // Only include if needed
            "accept": true|false  // Conclusion based on the
    critic's analysis of the test quality
        }
        // Add entries for each test in the test file
    },
    "tests_completeness": {
        "explanation": "Analysis of whether the tests completely
    cover all prompt requirements",
        "missing_tests": ["list", "of", "requirements", "that",
    "should", "be", "tested", "but", "aren't"],
        "corrections": "Specific additional tests needed if the
    evaluation fails", // Only include if there are issues
        "accept": true|false  // Conclusion based on the analysis
    },
    "feedback": "Detailed feedback explaining all issues and
    providing clear guidance for improvements"
}
```

Be rigorous in your evaluation. The goal is to ensure high-quality dataset items that will effectively test LLM code generation capabilities. Only provide the JSON response, no additional explanation. The response will be parsed with json.loads(response) so be sure json format is correct. Instead of triple-quotes use characters.
IMPORTANT CUSTOM INSTRUCTIONS

```
[Custom instructions from human reviewers are provided here when
    available. These instructions provide special guidance in the
    evaluation process.]
```

F.2 Generator Prompt Template

The following is the template used for the generator in the InfraBench dataset construction process:

Generator Prompt Template

As an AI assistant specialized in Infrastructure as Code, your task is to create a high-quality dataset item for an LLM code generation challenge.
You will be given:
1. A repository description

2. A pre-suggestion with a specific code section to mask
3. The full repository content
Based on this information, you need to:
1. Create a clear, detailed human language prompt describing what code needs to be generated
2. Develop comprehensive pytest tests that validate the generated code meets all requirements
REPOSITORY DESCRIPTION

```
[Repository description is provided here]
```

PRE-SUGGESTION

```
Item Name: [Item name]
File Path: [File path]
Start Line: [Start line]
End Line: [End line]
Complexity: [Complexity]
Rationale: [Rationale]
```

PREVIOUS FEEDBACK
When available, this section may include:
PREVIOUS TASK SUGGESTION

```
{
  "item_name": "example_item",
  "file_path": "path/to/file",
  "insert_points": {
    "start_line": 10,
    "end_line": 20
  },
  "prompt": "previous prompt text",
  "test_file": {
    "path": "tests/test_example.py",
    "content": "previous test file content"
  },
  "generator_notes": "previous notes"
}
```

IF THERE ARE VALIDATION ERRORS
The tests failed when run against the original masked code. Here are the errors:

```
Error details from validation step
```

Make sure the original code will pass the tests, and write them correctly according to what you see in these error logs.
CRITIQUE FEEDBACK

```
[Detailed feedback from the critique step explaining issues with
    the previous suggestion.]
```

GUIDELINES FOR CREATING A QUALITY DATASET ITEM
For the human language prompt:
1. COMPLETENESS: Describe ALL functional aspects needed for the code.
2. GENERALITY: Allow for multiple possible valid solutions that fulfill the requirements.
3. CLARITY: Be specific about requirements but avoid dictating implementation specifics.
4. PRECISION: Include all requirements that would allow someone to implement the solution correctly.
5. CONTEXT: Describe the functionality, purpose, and integration with other components.
6. ESSENTIAL DETAILS ONLY: Specify necessary parameters and behaviors, but avoid over-constraining the solution.
7. NO SPOILERS: DO NOT include the actual implementation details or code snippets.
8. IMPLEMENTATION FREEDOM: Focus on "what" needs to be achieved, not "how" it must be done.

9. NO HINTS: If something can be inferred unequivocally from the repository code, it does not need to be specified in the prompt.
For the test file: 1. COMPREHENSIVE COVERAGE: Tests must verify ALL aspects mentioned in the prompt
2. EXPLICIT PURPOSE: Each test should clearly indicate what requirement it's checking
3. APPROPRIATE VERIFICATION: Tests must use assertions that correctly validate the implementation
4. PRACTICALITY: Tests MUST pass when run against the original code that will be masked
5. ROBUSTNESS: Tests should fail if important requirements are not met
6. COMPLETENESS: No requirement from the prompt should be left untested
7. STRUCTURE: Use appropriate fixtures and mocks, follow pytest best practices
REPOSITORY CONTENT File to be masked: [File path]

```
[File content is provided here]
```

```
[All other relevant context files are provided here]
```

Return the response in this JSON format:

```
{
    "item_name": "SAME_AS_PRE_SUGGESTION",
    "file_path": "path/to/file",
    "insert_points": {
        "start_line": number,
        "end_line": number
    },
    "prompt": "detailed description of what needs to be generated",
    "test_file": {
        "path": "tests/test_something.py",
        "content": "complete content of the test file including
    imports, fixtures, and test cases"
    },
    "generator_notes": "You don't need to accept all the
    suggestions from the feedback, but give an explanation of your
    approach, learnings regarding test syntax for the cannonical
    solution to pass, detailed design decisions, rationale for
    implementation choices, and how you've ensured generality in
    the prompt and tests. There will be a new critic, so explain
    your decisions without assuming the critic understands the
    current feedback. IMPORTANT: give an analysis of possible flaws
    in the generated prompt and tests focusing on its generality
    (if various valid solutions would be accepted), coverage and
    alignment between prompt and tests, etc."
}
```

Ensure that:
1. The insert_points are within the pre-suggestion range
2. The prompt is comprehensive and covers ALL requirements
3. The test_file content is complete and will validate ALL requirements
4. The tests MUST pass when run against the original masked code
Only provide the JSON response, no additional explanation.
IMPORTANT CUSTOM INSTRUCTIONS

```
[Custom instructions from human reviewers are provided here when
    available. These instructions provide special guidance in the
    generation process.]
```

F.3 Zero-shot Direct Solver

Anthropic template

You are tasked with implementing a solution based on the following prompt:

```
<example_prompt>
{example_prompt}
</example_prompt>
```

Ensure the solution is compatible with AWS Python CDK version aws-cdk-lib = `{cdk_version}`.
You have access to the following context files:

```
<context_files>
{context_files}
</context_files>
```

Your task is to provide git-style unified diffs that show the changes needed to implement the solution. For each file that needs changes, provide a unified diff. Only add content, do not remove any lines
Provide your response as a JSON object where: - Keys are the file paths - Values are arrays containing the unified diffs for each change section
The diff format should be:

```
<output_format>
--- without_solution
+++ with_solution
@@ -line,count +line,count @@
    context lines
+added lines
    context lines
</output_format>
```

Example response format:

```
<example_response>
{
    "path/to/file1.py": [
        "--- without_solution\\n+++ with_solution\\n
        @@ -10,3 +10,5
        @@\\n     existing_line\+
        new_line1\\n+    new_line2\\n
        existing_line"
    ]
}
</example_response>
```

Only provide the JSON response, no additional explanation. The response will be parsed with `json.loads(response_text)` so make sure the string is correct JSON. Do NOT include ```` ```json ````
Ensure the diffs include proper line numbers and context.

Default template

You are tasked with implementing a solution based on the following prompt:

```
{example_prompt}
```

Ensure the solution is compatible with AWS Python CDK version aws-cdk-lib = `{cdk_version}`.
You have access to the following context files:

```
{context_files}
```

Your task is to provide git-style unified diffs that show the changes needed to implement the solution. For each file that needs changes, provide a unified diff. Only add content, do not remove any lines
Provide your response as a JSON object where: - Keys are the file paths - Values are arrays containing the unified diffs for each change section
The diff format should be:

```
--- without_solution
+++ with_solution
@@ -line,count +line,count @@
    context lines
+added lines
    context lines
```

Example response format:

```
{
    "path/to/file1.py": [
        "--- without_solution\\n+++ with_solution\\n
        @@ -10,3 +10,5
        @@\\n     existing_line\+    new_line1
        \\n+    new_line2\\n
        existing_line"
    ]
}
```

Only provide the JSON response, no additional explanation. The response will be parsed with `json.loads(response_text)` so make sure the string is correct JSON. Do NOT include ```` ```json ````
Ensure the diffs include proper line numbers and context.

F.4 Zero-shot Two-Turn Solver

First Turn Prompt

```
# Task overview
You are tasked with implementing a solution
based on the following prompt:
{example_prompt}

# Steps
- Have a look at the version of aws-cdk-lib
    - Current version is {cdk_version}
- Read each all of the context files ("# Context files" section)
- Try to understand what should be done
- Read task overview
("# Task overview" section)
- If provided, read related AWS documentation
("# Related documentation" section)
- Provide solution with a specified format
("# Response format" section)

# Response format
## General guidelines
Provide your solution as a JSON object with:
1. File paths as keys
2. Arrays of unified diffs as values

## Diff format specification
```

````
Each diff must follow this structure:
```
--- without_solution
+++ with_solution
@@ -line,count +line,count @@
 context lines
+added lines
 context lines
```

## Important rules:
- Only ADD content, do not remove any lines
- Include proper line numbers and context
- Ensure diffs are properly formatted

## Example JSON response:
```json
{{
    "path/to/file1.py": [
        "--- without_solution\n+++ with_solution\n
        @@ -10,3 +10,5 @@\n
        existing_line\n+    new_line1\n+    new_line2\n
        existing_line"
    ]
}}
```

## Additional guidelines
- Provide ONLY the JSON response
- No additional explanation
- Response must be valid JSON (will be parsed with json.loads())
- If you want to think before returning response,
be short and concise

# Context files
Here are the context files to analyze:
{context_files}
````

Second Turn Prompt

```
# Task overview
You are tasked with fixing errors in an LLM-generated solution
based on the following initial prompt:
{example_prompt}

# Steps
- Have a look at the version of aws-cdk-lib
    - Current version is {cdk_version}
- Read each all of the context files ("# Context files" section)
- Try to understand what should be done
- Read task overview ("# Task overview" section)
    - This is an initial formulation of the task
    that LLM have used to generate its solution
- Read previous attempt solution code
("# Previous attempt" section)
    - This is the solution generated by LLM which fails checks
- Read error traceback ("# Error message" section)
```

````
    - Interpret error message
    - Identify specific error type (syntax, type, logic, etc.)
- If provided, read related AWS documentation
("# Related documentation" section)
- Provide error analysis
    - Put it in "error_analysis"
- Provide a working solution with a specified format
    - Put it in "regenerated_solution"

# Response format
## General guidelines
Provide your solution as a JSON object with:
1. "error_analysis": Your complete error analysis
2. "regenerated_solution": Object containing file paths and
their diffs

## Diff format specification
Each diff must follow this structure:
```
--- without_solution
+++ with_solution
@@ -line,count +line,count @@
 context lines
+added lines
 context lines
```

## Important rules:
- Only ADD content, do not remove any lines
- Include proper line numbers and context
- Ensure diffs are properly formatted

## Example JSON response:
```json
{{
    "error_analysis": "The error occurred because...",
    "regenerated_solution": {{
        "path/to/file1.py": [
            "--- without_solution\n+++ with_solution\n
            @@ -10,3 +10,5
            @@\n     existing_line\n+    new_line1\n
            +    new_line2\n
            existing_line"
        ]
    }}
}}
```

## Additional guidelines
- Provide ONLY the JSON response
- No additional explanation
- Response must be valid JSON (will be parsed with json.loads())
- If you want to think before returning response,
be short and concise

# Related documentation
Here are supporting documentation pieces:
```markdown
````
````

````
{documentation}
```

# Context files
Here are the context files to analyze:
{context_files}

# Previous attempt
Previous solution, which needs fixing:
{solution_code}

# Error message
Error message of the previous solution:
{error_message}
````

F.5 Zero-shot Two-Turn Solver (RAG)

For this configuration, same prompt template from previous configuration is used for second turn. First turn template is different from the default two-turn solver configuration and allows in the same time to generate solution and keywords to search for in the documentation:

First Turn Prompt

```
# Task overview
You are tasked with implementing a solution
based on the following prompt:
{example_prompt}

# Context files
Here are the context files to analyze:
{context_files}

# Steps
- Have a look at the version of aws-cdk-lib
    - Current version is {cdk_version}
- Read task overview ("# Task overview" section)
- Read each all of the context files ("# Context files" section)
- Try to understand what should be done
- If provided, read related AWS documentation
("# Related documentation" section)
- Provide solution with a specified format
("# Response format" section)
- Provide documentation support keywords

# Documentation Support
To confirm your implementation you should provide keywords in
"search"
in key:
- Include a "search" key in your JSON response
- Provide up to 5 specific keywords
related to the AWS services
and features you need help with
- Example: "search": "data access policies opensearch"
    - Avoid using underscores or other similar
    special characters, query should be human-readable
    - Keywords should reflect the resources used in the stack
```
```

````
    or errors seen during execution,
    i.e. 'opensearch' or 'appsync'
    - Avoid too generic keywords like 'aws' or 'cloudformation'
- Keywords you provide will be used in the next interaction
in case solution does not pass the tests

# Response format
## General guidelines
Provide your solution as a JSON object with:
1. File paths as keys
2. Arrays of unified diffs as values

## Diff format specification
Each diff must follow this structure:
```
--- without_solution
+++ with_solution
@@ -line,count +line,count @@
 context lines
+added lines
 context lines
```

## Important rules:
- Only ADD content, do not remove any lines
- Include proper line numbers and context
- Ensure diffs are properly formatted

## Example JSON response:
```json

{{
    "path/to/file1.py": [
        "--- without_solution\n+++ with_solution\n
        @@ -10,3 +10,5
        @@\n     existing_line\n+    new_line1\n
        +    new_line2\n
        existing_line"
    ],
    "search": "..."
}}
```


## Additional guidelines
- Provide ONLY the JSON response
- No additional explanation
- Response must be valid JSON (will be parsed with json.loads())

------

Your JSON response (start with ```json):
````

G Model Configurations

Table 5 overviews the models used in our experiments and provides their invocation parameters. Some reasoning models (DeepSeek R1, OpenAI o3, OpenAI o4-mini) have **4 times bigger token budget**, Gemini 2.5 models require **5 times more**, reaching more than 20K maximum tokens per task. This was done to allow for longer outputs for reasoning models to avoid result truncation.

Table 5: Models Invocation Parameters

Model	Model ID	Invocation Parameters
Claude 3 Haiku	anthropic.claude-3-haiku-20240307-v1:0	max_tokens=4096; temperature=0.25; top_p=0.9
Claude 3.5 Sonnet	anthropic.claude-3-5-sonnet-20240620-v1:0	max_tokens=4096; temperature=0.25; top_p=0.9
Claude 3.5 Sonnet V2	anthropic.claude-3-5-sonnet-20241022-v2:0	max_tokens=4096; temperature=0.25; top_p=0.9
Claude 3.7 Sonnet	claude-3-7-sonnet-latest	max_tokens=4096; temperature=0.25
Codestral	codestral-latest	max_tokens=4096; temperature=0.25
DeepSeek R1	deepseek-ai/DeepSeek-R1	max_tokens=16384; temperature=0.25
GPT-4.1	gpt-4.1	max_output_tokens=4096; temperature=0.25
GPT-4o Mini	gpt-4o-mini	max_output_tokens=4096; temperature=0.25
Gemini 2.0 Flash	gemini-2.0-flash	max_output_tokens=4096; temperature=0.25
Gemini 2.0 Flash Lite	gemini-2.0-flash-lite	max_output_tokens=4096; temperature=0.25
Gemini 2.5 Pro (03-25 Preview)	gemini-2.5-pro-preview-03-25	max_output_tokens=20480; temperature=0.25
Gemini 2.5 Pro (05-06 I/O Edition Preview)	gemini-2.5-pro-preview-05-06	max_output_tokens=20480; temperature=0.25
LLaMA 3.1 405B Instruct	meta.llama3-1-405b-instruct-v1:0	max_tokens=4096; temperature=0.25; top_p=0.9
LLaMA 3.1 70B Instruct	us.meta.llama3-3-70b-instruct-v1:0	max_tokens=4096; temperature=0.25; top_p=0.9
LLaMA 4 Maverick 17B Instruct	meta.llama4-maverick-17b-instruct-v1:0	max_tokens=4096; temperature=0.25
LLaMA 4 Scout 17B Instruct	meta.llama4-scout-17b-instruct-v1:0	max_tokens=4096; temperature=0.25
Mistral Large	mistral-large-latest	max_tokens=4096; temperature=0.25
OpenAI o3	o3	max_output_tokens=16384
OpenAI o4-mini	o4-mini	max_output_tokens=16384
Qwen2.5 72B Instruct Turbo	Qwen/Qwen2.5-72B-Instruct-Turbo	max_tokens=4096; temperature=0.25

H Verbosity Configurations for Two-Turn Solvers

Our experimental framework uses pytest as the testing library, with two distinct verbosity configurations designed to evaluate how different levels of feedback detail affect model performance. Here are the details on each of the configurations:

Table 6: Verbosity Parameters

	Low-verbosity (V_L)	**High-verbosity (V_H)**
Configuration		
Flags	`-q -tb=no -no-summary`	$\emptyset$
Output Features		
Pass/fail counter	✓	✓
Exception	✓	✓
Traceback	×	✓
Test files names	×	✓
Test functions names	×	✓

I Models Performance by Donor Repository Type

We analyzed the performance of models across different repository sources to investigate potential biases in our benchmark. The SWE-InfraBench dataset comprises 100 tasks derived from 34 distinct base repositories, with 66 examples originating from open-source repositories and 34 from custom-developed sources. While each task underwent substantial engineering modifications regardless of its origin, we sought to determine whether tasks derived from open-source repositories might be

less difficult compared to custom-developed ones due to models being potentially trained on the open-source data.

As shown in Table 7 for the majority of models the tasks derived from open-source repositories are equally or even more challenging. One important consideration is that randomness in the generation process and limited group sizes (34 tasks in the smallest group) suggest caution in the results interpretation.

Table 7: Correctness of proprietary and open-source LLMs on the whole benchmark and separately for tasks created from open-source and from custom created repositories. Models are grouped by source type and provider, and sorted from most recent to oldest variant within each group. † indicates LLMs executed with reasoning capabilities. ‡ Claude 3.7 was executed without extended reasoning. Top results are in **bold**.

Source	Company	Model	Correctness		
			All Tasks	Custom Derived Tasks	Open-Source Derived Tasks
Proprietary	Anthropic	Claude 3.7 Sonnet ‡	**34%**	35%	**33%**
		Claude 3.5 Sonnet V2	32%	32%	32%
		Claude 3.5 Sonnet	29%	32%	27%
		Claude 3 Haiku	8%	12%	6%
	Google	Gemini 2.5 Pro (03-25 Preview) †	29%	**44%**	21%
		Gemini 2.5 Pro (05-06 I/O Edition Preview) †	29%	38%	24%
		Gemini 2.0 Flash Lite	5%	6%	5%
		Gemini 2.0 Flash	4%	0%	6%
	OpenAI	OpenAI o3 †	23%	29%	20%
		OpenAI o4-mini †	23%	26%	21%
		GPT-4.1	18%	18%	18%
		GPT-4o Mini	4%	9%	2%
Open-Source	DeepSeek	DeepSeek R1 †	24%	26%	23%
	Mistral	Mistral Large	14%	12%	15%
		Codestral	9%	15%	6%
	Meta	LLaMA 3.1 405B Instruct	9%	12%	8%
		LLaMA 3.1 70B Instruct	3%	6%	2%
		LLaMA 4 Maverick 17B Instruct	8%	9%	8%
		LLaMA 4 Scout 17B Instruct	2%	3%	2%
	Alibaba	Qwen2.5 72B Instruct Turbo	0%	0%	0%

J Consistency Analysis

We quantified consistency using a "Success Consistency" metric. For each model and example, the "Task Success Consistency Gap" was calculated as the difference between the best and worst attempt correctness:

$$\text{Task Success Consistency Gap} = \max_{i \in \text{attempts}} (\text{correctness}_i) - \min_{i \in \text{attempts}} (\text{correctness}_i) \tag{1}$$

Success Consistency is then defined as:

$$\text{Success Consistency} = 1 - \text{Task Success Consistency Gap} \tag{2}$$

This value is averaged over all examples for each model. A value of 1 means perfectly consistent performance across attempts (either always failing or always succeeding). A value of 0 indicates maximum inconsistency (some attempts succeed while others fail).

As indicated in Section 4 all LLMs demonstrate substantial improvement when given multiple opportunities to solve a task. Gemini 2.5 Pro achieves the highest pass@5 rate, indicating that for 47% of tasks, at least one of five attempts produces a fully correct solution. This represents a considerable improvement over its single-attempt performance. DeepSeek R1 and OpenAI o3 show similar patterns, with pass@5 rates consistently higher than pass@1. Claude 3.7 Sonnet, however, shows higher average correctness of the results. It explains how Claude 3.7 Sonnet with its slightly lower pass@5 of 41% achieves the top position in one-trial benchmarking.

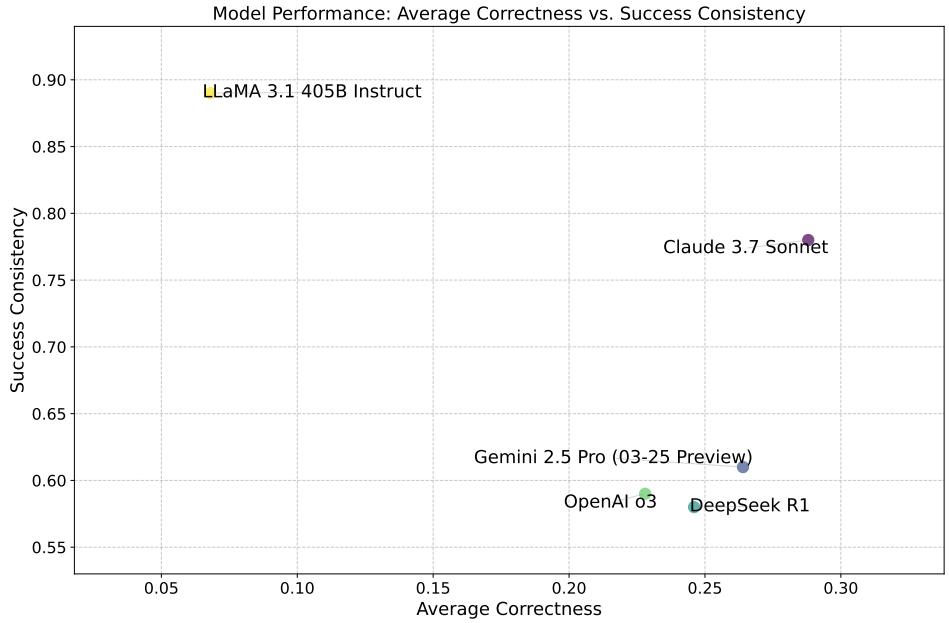

Figure 6: Average Correctness vs Consistency. Models positioned higher on average correctness, notably Claude 3.7 Sonnet, tend to demonstrate greater consistency, providing similar outcomes across repeated attempts. In contrast, models such as Gemini 2.5 Pro, DeepSeek R1, and OpenAI o3, despite achieving significant improvements when allowed multiple attempts, exhibit less consistency, indicating more variability in their success across trials.

Figure 6 illustrates the relationship between average correctness across trials and success consistency for tested models. We observed that higher-performing on average Claude 3.7 Sonnet demonstrated better consistency, while Gemini 2.5 Pro, DeepSeek R1 and OpenAI o3 are not as consistent in providing correct results on this dataset.

K Error Type Distribution

Figure 7 illustrates how error distributions change when models are given a second attempt with feedback (high verbosity configuration) from their first attempt. All models benefit from the second attempt, improving both on syntax and logical errors. However, a subset of LLMs without reasoning, namely GPT-4.1, Mistral Large and Llama 4 Maverick 17B Instruct, retain the same percentage of logical errors, mostly correcting the syntax ones. Such a behavior indicates that even high verbosity level is insufficient to address these more complex, implementation-specific failures. On the other hand, Claude Sonnet models, as well as Gemini 2.5 and DeepSeek R1, that further leverage their reasoning capabilities, showcase the considerable improvement on this matter as a prove of better analysis of underlying infrastructure dependencies and relationships.

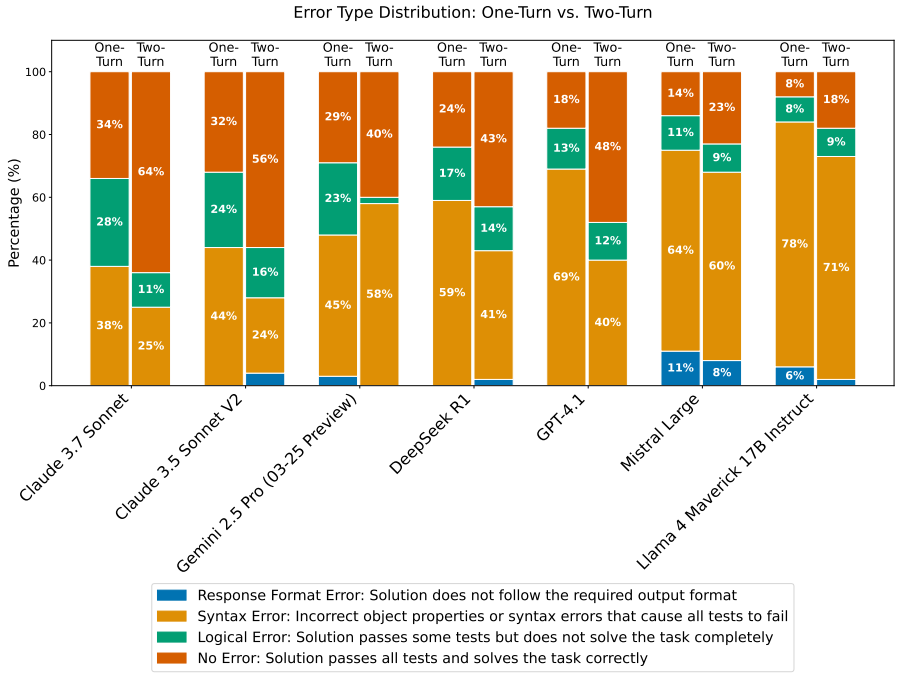

Figure 7: Error type distribution comparison between one-turn and two-turn (without RAG, high verbosity) approaches. All models show improved performance in the two-turn approach. The Claude family models demonstrate balanced improvement by reducing both syntax and logical errors. Gemini model primarily addresses logical errors in its second attempt, while GPT-4.1 and DeepSeek R1 shows substantial reduction in syntax errors. While most examples benefit from the two-turn approach, a small percentage show regression due to the inherent randomness in the generation process.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] The claims in the abstract and introduction are accurately aligned with the actual scope of the paper, which introduces a novel infrastructure-as-code benchmark.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors? [Yes] The Conclusion section discusses some limitations and possible future extensions of the introduced benchmark and empirical experiments conducted in the paper.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof? [NA] The paper does not include any theoretical results.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)? [Yes] The paper provides a description of the benchmark construction as well as empirical experiments conducted on it. We publish the complete dataset, disclose the links and licenses of the donor repositories, and the code implementing the benchmark construction pipeline and the evaluation experiments for different models and approaches tested in the paper. This allows reproducing our results, or extending the analysis by evaluating techniques not considered in this paper.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material? [Yes] The paper provides access to the data and code, with instructions to reproduce the main experimental results and analyze the dataset entries.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results? [Yes] The paper provides the prompt templates, LLM inference parameters, and evaluation metrics definitions for the provided evaluation experiments.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments? [No] We do not perform statistical significant tests of the overall model performance due to high costs of repeated LLM calls. At the same time, we provide additional analysis, including error distribution analysis and investigate performance of selected LLMs in a setting with multiple independent generations, presented in the main paper and in the Appendix. This provides insights on the consistency of the observed results.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments? [No] The paper focuses on benchmarking LLMs available via API calls by different model providers, which does not create any non-standard computer resources to reproduce the experiments.

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`? [Yes] Justification: The research conforms with the NeurIPS Code of Ethics.

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed? [NA] The benchmark targets low-level infrastructure code modifications without direct interaction with end users or social systems, and thus does not raise immediate concerns around fairness, privacy, or misuse.

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)? [NA] The paper poses no substantial risks of misuse.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected? [Yes] The paper provides a list of licenses for the donor repositories that were used to construct the benchmark.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets? [Yes] The paper provides documentation of the introduced asset – a new benchmark dataset for infrastructure-as-code generation.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)? [NA] The paper does not involve crowdsourcing nor research with human subjects.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained? [NA] The paper does not involve crowdsourcing nor research with human subjects.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required. [Yes] As described in the paper, we use a semi-automated pipeline for generating dataset examples using LLMs. The process is documented in detail in Section 2, and prompt examples are provided in the Appendix.