AMBIG-IAC: Multi-level Disambiguation for Interactive Cloud Infrastructure-as-Code Synthesis

Zhenning Yang, Kaden Gruizenga, Tongyuan Miao, Patrick Tser Jern Kon, Ang Chen
University of Michigan
{znyang,kgruiz,tymiao,patkon,chenang}@umich.edu

Hui Guan
University of Massachusetts Amherst
huiguan@umass.edu

Abstract

The scale and complexity of modern cloud infrastructure have made Infrastructure-as-Code (IaC) essential for managing deployments. While large Language models (LLMs) are increasingly being used to generate IaC configurations from natural language, user requests are often underspecified. Unlike traditional code generation, IaC configurations cannot be executed cheaply or iteratively repaired, forcing the LLMs into an almost one-shot regime. We observe that ambiguity in IaC exhibits a tractable compositional structure: configurations decompose into three hierarchical axes (resources, topology, attributes) where higher-level decisions constrain lower-level ones. We propose a training-free, disagreement-driven framework that generates diverse candidate specifications, identifies structural disagreements across these axes, ranks them by informativeness, and produces targeted clarification questions that progressively narrow the configuration space. We introduce AMBIG-IAC, a benchmark of 300 validated IaC tasks with ambiguous prompts, and an evaluation framework based on graph edit distance and embedding similarity. Our method outperforms the strongest baseline, achieving relative improvements of +18.4% and +25.4% on structure and attribute evaluations, respectively.

1 Introduction

Large Language models are increasingly adopted for synthesizing structured artifacts from natural-language specifications: code, database queries, and configuration programs (Ding et al., 2025; Zhou et al., 2025; Mu et al., 2024). However, user requests that drive this synthesis are often underspecified. User prompts state high-level goals, not implementation details, so many valid outputs can satisfy the same prompt. A model that silently "resolves" this ambiguity via its own priors may produce plausible but misaligned results. Cloud Infrastructure-as-Code (IaC) is a particularly demanding instance of this problem (Yang et al., 2025a). IaC frameworks such as Terraform (HashiCorp, 2025b) define cloud infrastructure as declarative configuration programs (Yang et al., 2025a), and recent work has begun generating them directly from natural language (Kon et al., 2025). A single user request typically specifies only high-level infrastructure goals, leaving lower-level decisions (e.g., compute abstractions, resource connectivity, and security or networking policies) underspecified. This ambiguity gives rise to a large space of plausible yet potentially misaligned configurations (Figure 1).

A natural response to ambiguity is iterative refinement. In conventional code generation, agents refine programs using execution feedback: running tests, observing failures, and repairing. Cloud configuration offers no such loop. Executing an IaC program requires provisioning real infrastructure, which is slow, costly, and often irreversible or subject to strict rate limits. Dry-run validators such as `terraform plan` catch syntactic and type errors

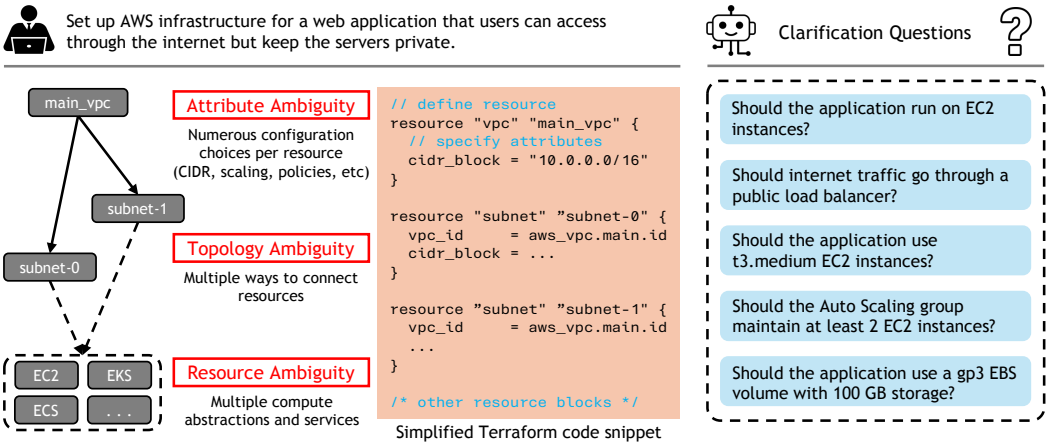

Figure 1: An underspecified user request corresponds to many plausible cloud infrastructures that can be represented as resource dependency graphs (left). Ambiguity arises in choosing resource abstractions and services for compute, networking, databases, etc., inter-resource topology, and per-resource configuration attributes when translating intent into Infrastructure-as-Code (right).

but cannot detect semantic misalignment with user intent. In one recent public incident, an AI-assisted Terraform workflow deleted a production database, requiring roughly 24 hours of recovery (Grigorev, 2026). Cloud agents, therefore, operate in an almost one-shot regime where incorrect decisions incur real cost and are difficult to undo. Hence, precise disambiguation is a first-class requirement in IaC generation.

Existing clarification methods were designed for settings where ambiguity is open-ended and driven by common-sense reasoning, and where feedback is cheap. Neither assumption holds for cloud IaC. Prior work on LM-based clarification either assumes ambiguity can be resolved in very few turns (Zhang et al. (2025) restricts to one-or-two-turn interactions, Andukuri et al. (2024) cap at K=3 turns, Mu et al. (2024) batches an average of 2.85 questions, and Min et al. (2020) avoids interaction entirely) or depends on supervised fine-tuning and preference optimization that assume stable task distributions and cheap supervision (Zhang et al., 2025; Andukuri et al., 2024; Chen et al., 2025a). Methods from code generation detect ambiguity by comparing multiple candidate programs and identifying behavioral divergence on synthesized or distinguishing tests (Mu et al., 2024; Lahiri et al., 2023), a strategy that cannot easily transfer to IaC, where execution means provisioning expensive cloud resources. More fundamentally, no existing approach provides a principled mechanism for deciding *where* to direct clarification effort for cloud IaC. Vijayvargiya et al. (2025) find empirically that LMs struggle to distinguish well-specified from underspecified instructions, and that unstructured prompting wastes the question budget on low-value clarifications. As Figure 1 illustrates, cloud IaC generation involves a hierarchically structured configuration space where resources, topology, and attributes are tightly coupled. This structure can and should help guide the clarification process.

Cloud configurations decompose into three specification axes (*resources*: which components to provision; *topology*: how they connect; *attributes*: what settings to apply) that form a hierarchy: resource decisions constrain which topologies are feasible, and topology decisions constrain which attributes are relevant. Because each axis admits multiple valid choices, we can probe the extent of ambiguity by generating diverse candidate specifications and comparing them structurally: points where candidates disagree reveal where ambiguity concentrates. We propose a disagreement-driven disambiguation framework that exploits this observation to guide clarification under a fixed interaction budget. The framework generates diverse candidate specifications, identifies structural disagreements across the three axes, ranks them by informativeness, and uses the most informative disagreements to generate targeted clarification questions. After each user's answer, incompatible candidates are pruned, and the process repeats, progressively narrowing the configuration space. The framework requires no training or fine-tuning and works with any sufficiently capable LLM.

This paper makes the following contributions:

- We present, to our knowledge, the first investigation of interactive cloud IaC generation under ambiguous user requests, where an agent explicitly reasons about the structure of uncertainty and resolves it through multi-turn clarification dialogue.
- We propose a training-free, disagreement-driven framework that decomposes IaC ambiguity into three hierarchical axes (resource, topology, and attribute), generates diverse candidate specifications, and uses structural disagreements—ranked by entropy with balanced cross-dimension selection—to produce targeted clarification questions that efficiently narrow the configuration space.
- We introduce AMBIG-IAC, a curated benchmark of 300 IaC tasks, with validated reference configurations and LLM-generated ambiguous prompts. We additionally propose an evaluation framework that measures configuration correctness along structure (via graph edit distance) and attributes (via embedding similarity).
- We performed a comprehensive evaluation and show that our method consistently outperforms all baselines at different interaction budgets (5, 10, and 15 rounds). At 15 rounds, it achieves 54.85% structure and 45.72% attribute correctness, improving over the strongest baseline by +8.53 and +9.25 points, respectively (+18.4% and +25.4% relative).

2 Motivation

This problem is especially important in practice because cloud infrastructure is now economically central and operationally complex. Modern computing increasingly relies on large-scale cloud deployments to support data-intensive applications, distributed services, and machine learning workloads. And that footprint continues to grow: Gartner forecasts worldwide IT spending of $6.15 trillion in 2026, including more than $650 billion in data center spending driven in part by hyperscale cloud demand (Gartner, 2026). As cloud estates expand, they also become harder to manage; Firefly's 2025 State of IaC report finds that 68% of respondents operate across multiple clouds, 65% report increased cloud complexity over the past two years, and only 6% report complete IaC coverage (Firefly, 2025). In such environments, ad hoc manual changes become difficult to audit, reproduce, and reconcile (Yang et al., 2025b), motivating Infrastructure-as-Code (IaC) frameworks such as Terraform that express infrastructure through declarative, version-controlled specifications rather than imperative step-by-step operations (Amazon Web Services, 2026; HashiCorp, 2025a).

At the same time, writing IaC remains difficult because it requires substantial knowledge of cloud services, configuration semantics, and provider-specific tooling (Srivatsa et al., 2024). This has fueled growing interest in using large language models to generate IaC from natural language, as reflected in recent surveys and benchmarks such as IaC-Eval and Multi-IaC-Eval that explicitly study cloud infrastructure generation and editing (Srivatsa et al., 2024; Kon et al., 2025; Davidson et al., 2025). However, generating correct IaC remains challenging because it requires mapping underspecified user requests to large, interdependent cloud configurations (Kon et al., 2025).

3 The Problem

We study the problem of generating IaC configurations from ambiguous natural-language requests through interactive clarification. Given an underspecified user intent I, there exists a large set $\mathcal{C}(I)$ of valid configurations that satisfy the explicitly stated intent requirements. The user possesses both an *explicit intent* (the initial request) and an *implicit intent* (unstated constraints and preferences), which together determine a target configuration c^*. The goal is for LLM to elicit the implicit intent through interaction.

Ambiguity in IaC synthesis arises along three axes: **resources**, **topology**, and **attributes**. Together, these axes define the combinatorial space of plausible infrastructure configurations.

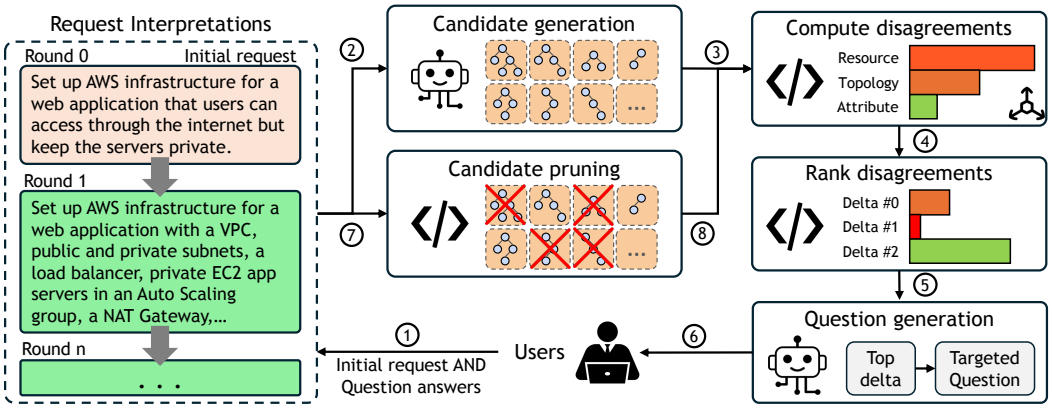

Figure 2: Overview of the iterative multi-level disambiguation process for interactive Infrastructure-as-Code synthesis. ① Users provide an initial IaC request. ② Initially, a pool of diverse structured specifications will be generated based on the current interpretations. ③ Symbolically compute disagreements among the current pool of candidates. ④ These disagreements are ranked to identify the most informative differences measured by entropy. ⑤ Based on the top-ranked disagreement, the system generates a targeted clarification question. ⑥ Interaction with the user. ⑦ If the candidate pool is non-empty, prune candidates inconsistent with the user's answer. ⑧ The refined candidate pool is fed back into the next iteration, repeating the process until the interaction budget is exhausted. Otherwise, pool regeneration will be triggered.

In practice, infrastructure-as-code (IaC) languages interleave these concerns within unified resource blocks for convenience (see Figure 1). In contrast, we explicitly disentangle these axes and represent each IaC specification as a structured triple $s = (R, T, A)$: **Resources** R (a multiset of component types), **Topology** T (directed dependency edges), and **Attributes** A (configuration key–value pairs per resource). This decomposition induces a structured interface that supports more efficient and systematic intent elicitation by LLM-based agents.

4 Method

Given an ambiguous natural-language request and given a fixed interaction budget K, the goal is to generate a series of clarification questions that help the agent narrow the set of valid configurations toward the target configuration c^*. From this initial request, the LLM may infer several possible interpretations of the user's implicit intent. Our method makes these alternatives explicit by maintaining a *candidate pool*—a set of structured specifications representing different plausible ways the request could be answered. Disagreements among these candidates reveal where ambiguity concentrates—in the resources, the topology, or the attributes—and are used to generate targeted clarification questions. The agent iteratively asks one question per round, prunes the pool based on the user's answer, and regenerates candidates when the pool is exhausted. After the budget is spent, the best surviving candidate is returned.

Candidate generation. The candidate pool serves as a concrete approximation of the space of plausible configurations $\mathcal{C}(I)$. Given the user's request (and, in subsequent rounds, the accumulated question–answer history), the agent generates N candidate specifications. To encourage diversity, we sample candidates from the LLM using temperature sampling, so that they explore different resource selections, topology wirings, and attribute choices. Candidates that share the same resource types and topology edges are deduplicated, as these higher-level axes determine the overall infrastructure design; attribute-level differences are still captured during disagreement detection.

Compute disagreements. Given the candidate pool, the agent identifies points of disagreement across the three specification axes. **Structural Diffing.** Because each candidate is

represented as a structured specification (R, T, A), disagreements can be computed symbolically along each axis. For **resources**, the agent extracts the set of resource types from each candidate and records any type that is present in some candidates but absent from others. For **topology**, it extracts the set of directed dependency edges (typed by source and target resource types) and similarly records edges on which candidates disagree. For **attributes**, it groups candidates by resource type, then for each shared attribute key, checks whether candidates assign different values. Each disagreement records which candidates fall on each side of the split.

Rank disagreements. Not all disagreements are equally informative. A disagreement where nearly all candidates agree carries little discriminative value, while an even split maximizes the information gained from a single question. We rank disagreements by how evenly they divide the candidate pool, measured by Shannon entropy over the split. Low-entropy disagreements are discarded. To ensure balanced coverage, the top disagreements are selected via a round-robin scheduling across the three dimensions, preventing any single axis from dominating the question budget.

Question generation. In each round, the agent selects the top-ranked disagreement and converts it into a clarification question. Each disagreement already partitions the candidate pool into two sides (e.g., candidates that include a NAT gateway vs. those that do not). The agent passes this disagreement to an LLM, whose role is to rephrase the structural difference as a natural-language clarification question that a user can answer without knowledge of the underlying candidates.

Candidate pruning. From the user's answer, the agent updates the candidate pool to retain only the candidates that are consistent with the answer, and discards the rest. Because the partition is already known from the diff, pruning is immediate and deterministic—one answer can potentially eliminate multiple candidates at once, efficiently narrowing the pool.

Regeneration will be triggered when the pool is exhausted. The regeneration procedure repeats candidate construction, conditions on both the original request and the full accumulated interaction history. This allows the model to produce candidates that are consistent with information gathered so far, while still exploring remaining uncertainty. Benefit from the structured specification, newly generated candidates can be easily de-duplicated and filtered against prior interactions to avoid revisiting resolved disagreements. The interaction terminates when the question budget K is exhausted or when the agent can no longer produce structurally distinct candidates, indicating that the model's uncertainty has been sufficiently resolved. Upon termination, the agent returns the best surviving candidate from the pool. If no candidate survives, the agent performs one final generation conditioned on the original request and the full interaction history.

5 Experimental Setup

Dataset. We construct AMBIG-IAC, a benchmark of 300 Terraform tasks sourced from IaC-Eval (Kon et al., 2025). We manually verified and fixed the reference programs using `terraform plan`, filtering out programs that could not be validated. The dataset covers 167 unique AWS resource types across 44 service families (e.g., VPC, S3, IAM, RDS, Lambda). To introduce realistic ambiguity, we use an LLM to rewrite each original task prompt into a high-level, goal-oriented request that preserves consistency with the reference configuration but admits multiple plausible architectures. For example, "Configure a weighted routing policy that splits users between two db_instances that are replicas of a main db_instance" becomes "Our database is getting slow for users in other regions—need to fix that and make reads faster globally."

User Proxy. We simulate user responses with an LLM-based oracle. Given a ground-truth Terraform configuration and a binary clarification question, the proxy answers based solely on whether the reference configuration implies yes or no. The proxy is explicitly instructed to answer strictly from the reference — no external knowledge or assumptions beyond what

is specified may be used. All methods interact with the same oracle to ensure consistent information exposure across comparisons.

Baselines. We compare against three interactive clarification baselines, all operating under the same question budget K:

- **Direct Question Generation**: LLM directly generates clarification questions.
- **Best-of-N**: At each round, the LLM generates N candidate questions conditioned on prior Q&A. A separate ranker LLM selects the question.
- **Self-Consistency**: At each round, N candidate questions are generated, embedded, and clustered. The question closest to the centroid of the largest cluster is selected, favoring questions that the model consistently considers important.

All baselines use the same final generation step: after K rounds of clarification Q&A, the accumulated interaction history is used to construct a clarified intent, from which the final specification is generated.

Metrics. We evaluate generated infrastructure specifications against ground-truth references along two axes: *structure* and *attributes*. Both the generated and reference specs are first normalized to a common label namespace, then compared via a two-stage pipeline that jointly evaluates resource composition and topology through graph matching, and attribute correctness through embedding similarity.

Structure Score. We compute the Graph Edit Distance (GED) between the reference and generated graphs using unit costs for node and edge insertions/deletions, zero cost for same-type node substitutions, and unit cost for cross-type substitutions. This jointly penalizes missing or extra resources *and* incorrect wiring (topology) in a single score. The raw GED is normalized by dividing by the total number of nodes and edges in both graphs, yielding a score in $[0, 1]$ where 1.0 indicates identical graph structure. **Attribute Score (Embedding Similarity).** For each aligned node pair, we serialize the resource type and attributes into a canonical string (keys sorted lexicographically), embed both with a sentence transformer (`all-MiniLM-L6-v2`), and compute cosine similarity. Unmatched reference nodes (deletions) and surplus generated nodes (insertions) receive a similarity of 0. The per-task attribute score is the arithmetic mean of similarities across all nodes, including matched pairs, deletions, and insertions. This avoids dependence on an external LLM judge while capturing semantic equivalence (e.g., different phrasings of the same configuration value). Full details on the GED cost model, embedding serialization, and edge cases are in Appendix A.

6 Results and Analysis

We organize our evaluation and analysis around five research questions:

- **RQ1:** Does structure-aware disambiguation outperform structure-agnostic baselines?
- **RQ2:** Does our method make better use of additional interaction rounds?
- **RQ3:** Does the method generalize across backbone LLMs?
- **RQ4:** How does the candidate pool evolve during interaction?
- **RQ5:** Does cross-dimension balancing matter?

Across all five, the same pattern emerges: our disagreement-driven method produces better final specifications on both structure and attributes, and the advantage grows as the interaction budget increases.

RQ1. We evaluated all methods at interaction budgets of K=5, 10, and 15 rounds. Our method consistently achieves the highest scores on both structure and attribute metrics across all budgets, shown in Table 1. We observed that popular test-time inference techniques like Best-of-N and Self-Consistency do not offer a performance leap over direct question generation of IaC. At

Method	Struct. (%)	Attr. (%)
Direct Q Generation	+5.27	+2.50
Best-of-N	+4.94	+7.02
Self-Consistency	-0.75	+1.17
Ours	+8.11	+11.29

Table 2: Improvement of our method over each baseline, by budget.

Method	#Rounds	Struct. (%)	Δ Struct.	Attr. (%)	Δ Attr.
Direct Q Generation	5	43.08	-5.87 (↓11.99%)	35.05	-4.68 (↓11.78%)
Best-of-N	5	41.28	-7.67 (↓15.67%)	33.13	-6.60 (↓16.61%)
Self-Consistency	5	38.59	-10.36 (↓21.16%)	31.12	-8.61 (↓21.67%)
Ours	5	**48.95**	–	**39.73**	–
Direct Q Generation	10	44.38	-6.61 (↓12.96%)	35.39	-7.32 (↓17.14%)
Best-of-N	10	43.52	-7.47 (↓14.65%)	35.81	-6.90 (↓16.16%)
Self-Consistency	10	38.41	-12.58 (↓24.67%)	31.54	-11.17 (↓26.15%)
Ours	10	**50.99**	–	**42.71**	–
Direct Q Generation	15	46.32	-8.53 (↓15.53%)	36.47	-9.25 (↓20.23%)
Best-of-N	15	43.12	-11.73 (↓21.39%)	35.10	-10.62 (↓23.23%)
Self-Consistency	15	38.19	-16.66 (↓30.37%)	31.43	-14.29 (↓31.26%)
Ours	15	**54.85**	–	**45.72**	–

Table 1: Main results on AMBIG-IAC (GPT-4o-mini). Structure is measured by normalized GED, and attributes are measured by embedding similarity.

K=5, we outperform the strongest baseline (Direct Q Generation) by +5.87 on structure and +4.67 on attribute. The performance gap widens at higher budgets at K=15. Notably, Self-Consistency performs the worst among baselines, suggesting that question consensus does not align well with informativeness in the IaC domain, where the configuration space is highly structured, and question quality depends on targeting specific axes of ambiguity rather than broad agreement.

RQ2. Most methods benefit from additional rounds, as shown in Table 2, with our method gaining the most on both structure and attribute on average across budgets. Self-Consistency even shows a slight decrease, suggesting that consensus-based selection saturates quickly. The widening gap indicates that our method makes more effective use of each additional question. Baselines generate questions without explicit awareness of which dimensions remain ambiguous, so later questions are more likely to be redundant or low-value. In contrast, our disagreement-driven approach continually re-estimates where ambiguity concentrates after each answer, directing subsequent questions to the most informative remaining differences.

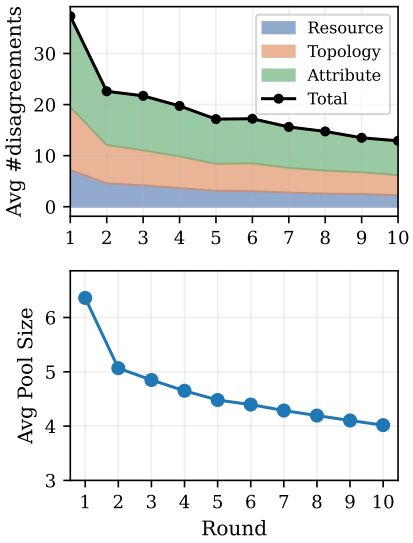

Figure 3: Per-round dynamics.

RQ3. As shown in Table 3, our method achieves the best results across both GPT-4o-mini and GPT-4.1-mini. The overall improvement observed with the stronger model holds for all methods, but the relative ordering remains unchanged, with our method consistently achieving the best scores on both models. This pattern indicates that the benefits of our approach arise from its structured disambiguation mechanisms rather than model-specific effects.

RQ4. We performed an additional study to understand how the candidate pool evolves over interaction rounds, shown in Figure 3. Disagreement counts across all three dimensions decrease steadily (top), confirming that each clarification round effectively resolves uncertainty. Resource-level disagreements are resolved fastest, followed by topology and then attributes, consistent with the hierarchical structure of IaC ambiguity: higher-level decisions constrain lower-level ones. The average pool size also shrinks over rounds (bottom), reflecting convergence toward a shared interpretation.

Figure 4 analyzes the regeneration mechanism. Most tasks require 3–5 regenerations (top), indicating that the candidate pool is typically exhausted within 2–3 questions, as each question is sufficiently discriminative to eliminate a substantial fraction of candidates.

Model	Method	Struct. (%)	Attr. (%)
GPT-4o-mini	Direct Q Generation	43.08	35.05
	Best-of-N	41.28	33.13
	Self-Consistency	38.59	31.12
	Ours	**48.95**	**39.73**
GPT-4.1-mini	Direct Q Generation	50.80	39.83
	Best-of-N	53.06	41.48
	Self-Consistency	52.58	40.66
	Ours	**57.40**	**45.10**

Table 3: Results across backbone LLMs (K=5).

Method	#Rounds	Struct. (%)	Attr. (%)
Ours w/o RR	5	47.18	37.83
Ours	5	**48.95**	**39.73**
Ours w/o RR	10	49.74	40.75
Ours	10	**50.99**	**42.71**
Ours w/o RR	15	51.65	42.14
Ours	15	**54.85**	**45.72**

Table 4: Ablation: round-robin (RR) cross-dimension balancing.

The bottom plot suggests a positive correlation between regeneration frequency and performance. This suggests that conditioned regeneration—producing new candidates that are consistent with accumulated Q&A history—is an effective mechanism for progressive refinement, as each cycle introduces candidates that are better aligned with the user's intent.

RQ5. Without round-robin ("Ours w/o RR"), disagreements are selected purely by entropy, making this ablation the closest analogue in our study to prior active task disambiguation methods that prioritize clarifications by information gain alone (Kobalczyk et al., 2025). In IaC, however, ambiguity is structured across resource composition, topology, and attributes, so a flat entropy ranking can over-focus on one axis while leaving others under-explored. Adding round-robin consistently improves both metrics, with the gap growing at higher budgets on both structure and attribute, as shown in Table 4. This shows that, in IaC, effective clarification depends not only on asking informative questions but also on covering structurally distinct axes of ambiguity. Even when individual entropy scores favor one dimension, allocating questions across all three leads to more balanced and complete disambiguation.

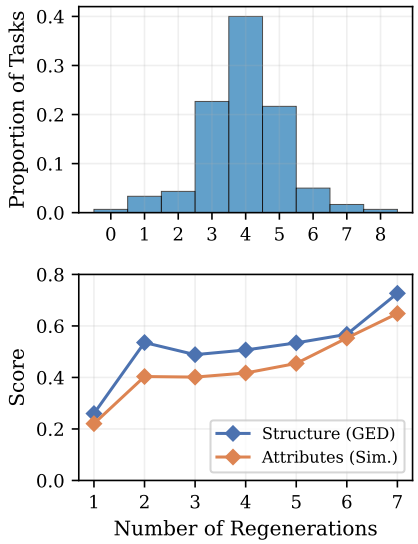

Figure 4: Regeneration analysis.

7 Related Work

LLM agents for cloud. Recent work has explored LLM-based systems for cloud management, including IaC generation benchmarks (Kon et al., 2025), agentic systems for autonomous cloud operations and incident analysis (Shetty et al., 2024; Chen et al., 2025b; 2024; Yang et al., 2025b; Xiang et al., 2025; Peng et al., 2025), provider-integrated assistants such as Azure Copilot, Gemini for Google Cloud, and Amazon Q (Microsoft, 2025; Google Cloud, 2025; Amazon Web Services, 2025), and broader discussions of cloud AI agents (Yang et al., 2025a). These efforts demonstrate the promise of cloud-facing agents for generation, diagnosis, and operational assistance. However, they generally assume that the given tasks are always well specified, and do not study how an agent should resolve ambiguous user intent before acting. This gap is important in practice: in one recent public incident, an AI-assisted Terraform workflow deleted production infrastructure (Grigorev, 2026).

Active task disambiguation. The closest conceptual precedent is recent work on active task disambiguation (Kobalczyk et al., 2025), which frames clarification as selecting questions that maximally reduce uncertainty over the space of plausible solutions. Rather than relying on zero-shot question generation alone, this line of work estimates question utility from candidate solutions and uses the resulting partitions to guide interaction. Our method is inspired by this perspective, but adapts it to IaC, where ambiguity is structured across resource composition, topology, and attributes. This structure matters because, as our ablation comparing round-robin balancing against non-RR flat entropy ranking in Section 6 shows,

selecting questions by entropy alone is insufficient, and balancing across disagreement dimensions leads to better final specifications.

Clarification via fine-tuning. Other prior work on ambiguity and clarification primarily targets bounded QA or preference-elicitation settings, where the interpretation space is smaller than in IaC. AmbigQA formulates ambiguity as a non-interactive task of enumerating multiple valid interpretations and rewrites for open-domain questions (Min et al., 2020). Subsequent works learn clarification behavior through future-turn preference optimization, self-training, or multi-turn policy learning (Zhang et al., 2025; Andukuri et al., 2024; Chen et al., 2025a). A related direction uses persistent memory and user modeling to infer goals across longer software-engineering interactions (Zhou et al., 2025). These methods are important precedents, but they rely on fine-tuning, synthetic supervision, or learned user models over relatively stable task distributions. In contrast, cloud IaC is a fast-evolving domain where provider APIs and specifications change rapidly, high-quality training data is scarce, and ambiguity is structured across resource, topology, and attribute decisions.

Clarification in code. The closest engineering analogues are code-oriented systems that resolve ambiguity by comparing competing realizations of the user's intent and querying the user on distinguishing evidence (Mu et al., 2024; Lahiri et al., 2023). Recent work in software-engineering agents further shows that interaction improves performance on underspecified tasks (Vijayvargiya et al., 2025). These results demonstrate the value of clarification in engineering domains. However, they still benefit from executable artifacts, generated tests, or inexpensive iterative repair. IaC differs because execution itself provisions real infrastructure, so ambiguity must be resolved before deployment rather than after failure.

8 Conclusion

We presented a multi-level disambiguation framework for interactive IaC generation that leverages the hierarchical structure of cloud configurations to guide multi-turn clarification, along with AMBIG-IAC, a 300-task benchmark of validated IaC tasks with ambiguous prompts. Our method consistently outperforms structure-agnostic baselines, with gains that scale with the interaction budget and generalize across models.

Future Work. While this work focuses on cloud IaC, the underlying principle—prioritizing higher-level disambiguation in hierarchically structured output spaces—may apply more broadly. Potential domains include database schema design (entity selection constrains relationships, which constrain column attributes), API workflow orchestration (service selection constrains dataflow, which constrains per-step parameters), and CI/CD pipeline generation (platform choice constrains stage structure, which constrains per-stage configuration).

9 Limitations

Computational cost. Our approach requires generating and maintaining multiple candidate configurations, as well as computing disagreements across them to guide question generation. Compared to simpler baselines that directly reason in the question space, this introduces higher token usage and computational overhead. However, we argue that this trade-off is justified in settings such as cloud infrastructure configuration, where misconfigurations can lead to significant financial or operational risks. Moreover, as model efficiency improves and inference costs decrease, the relative overhead of sampling multiple candidates is likely to diminish.

Focus on IaC intent elicitation over syntactic correctness. Our work primarily targets intent elicitation rather than ensuring syntactic validity of the generated programs. Accordingly, our evaluation emphasizes structural and semantic alignment with user intent, rather than strict adherence to provider-specific infrastructure-as-code (IaC) grammars. In practice, IaC languages vary across providers and evolve over time, making syntactic correctness a separate and complex challenge. We view this as largely orthogonal to our contribution: once user intent is accurately captured, syntactic validity can be addressed through complementary techniques such as stronger language models, constrained decoding, or

retrieval-augmented generation (RAG). Importantly, resolving ambiguity in user intent is a prerequisite for generating correct and deployable configurations.

References

Amazon Web Services. Amazon Q in AWS services, 2025. URL https://aws.amazon.com/q/.

Amazon Web Services. What is infrastructure as code? https://aws.amazon.com/what-is/iac/, March 2026. Accessed: 2026-04-01.

Chinmaya Andukuri, Jan-Philipp Fränken, Tobias Gerstenberg, and Noah Goodman. STar-GATE: Teaching language models to ask clarifying questions. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=CrzAj0kZjR.

Maximillian Chen, Ruoxi Sun, Tomas Pfister, and Sercan O Arik. Learning to clarify: Multi-turn conversations with action-based contrastive self-training. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=SIE6VFps9x.

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. Automatic root cause analysis via large language models for cloud incidents. EuroSys '24, 2024.

Yinfang Chen, Manish Shetty, Gagan Somashekar, Minghua Ma, Yogesh Simmhan, Jonathan Mace, Chetan Bansal, Rujia Wang, and Saravan Rajmohan. Aiopslab: A holistic framework to evaluate ai agents for enabling autonomous clouds, 2025b. URL https://arxiv.org/abs/2501.06706.

Sam Davidson, Li Sun, Bhavana Bhasker, Laurent Callot, and Anoop Deoras. Multi-iac-eval: Benchmarking cloud infrastructure as code across multiple formats, 2025. URL https://arxiv.org/abs/2509.05303.

Zhongjun Ding, Yin Lin, and Tianjing Zeng. Ambisql: Interactive ambiguity detection and resolution for text-to-sql, 2025. URL https://arxiv.org/abs/2508.15276.

Firefly. State of iac 2025. https://www.firefly.ai/state-of-iac-2025, 2025. Accessed: 2026-04-01.

Gartner. Gartner forecasts worldwide it spending to grow 10.8% in 2026. https://www.channel-impact.com/gartner-forecasts-worldwide-it-spending-to-grow-10-8-in-2026/, February 2026. Accessed: 2026-04-01.

Google Cloud. Gemini for Google Cloud, 2025. URL https://cloud.google.com/products/gemini.

Alexey Grigorev. How I dropped our production database and now pay 10% more for AWS. https://alexeyondata.substack.com/p/how-i-dropped-our-production-database, March 2026. Substack post. Accessed: 2026-03-31.

HashiCorp. What is infrastructure as code with terraform? https://developer.hashicorp.com/terraform/tutorials/aws-get-started/infrastructure-as-code, June 2025a. Accessed: 2026-04-01.

HashiCorp. Terraform, 2025b. URL https://developer.hashicorp.com/terraform.

Katarzyna Kobalczyk, Nicolas Astorga, Tennison Liu, and Mihaela van der Schaar. Active task disambiguation with llms, 2025. URL https://arxiv.org/abs/2502.04485.

Patrick Tser Jern Kon, Jiachen Liu, Yiming Qiu, Weijun Fan, Ting He Lei Lin, Haoran Zhang, Owen M. Park, George S. Elengikal, Yuxin Kang Ang Chen, Mosharaf Chowdhury, Myungjin Lee, and Xinyu Wang. Iac-eval: a code generation benchmark for cloud infrastructure-as-code programs. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, 2025.

Shuvendu K. Lahiri, Sarah Fakhoury, Aaditya Naik, Georgios Sakkas, Saikat Chakraborty, Madanlal Musuvathi, Piali Choudhury, Curtis von Veh, Jeevana Priya Inala, Chenglong Wang, and Jianfeng Gao. Interactive code generation via test-driven user-intent formalization, 2023. URL https://arxiv.org/abs/2208.05950.

Microsoft. Microsoft copilot in Azure, 2025. URL https://azure.microsoft.com/en-us/products/copilot.

Sewon Min, Julian Michael, Hannaneh Hajishirzi, and Luke Zettlemoyer. AmbigQA: Answering ambiguous open-domain questions. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 5783–5797, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.466. URL https://aclanthology.org/2020.emnlp-main.466/.

Fangwen Mu, Lin Shi, Song Wang, Zhuohao Yu, Binquan Zhang, ChenXue Wang, Shichao Liu, and Qing Wang. Clarifygpt: A framework for enhancing llm-based code generation via requirements clarification. *Proc. ACM Softw. Eng.*, 1(FSE), July 2024. doi: 10.1145/3660810. URL https://doi.org/10.1145/3660810.

Jingjia Peng, Yiming Qiu, Patrick Tser Jern Kon, Pinhan Zhao, Yibo Huang, Zheng Guo, Xinyu Wang, and Ang Chen. Automated lifting for cloud infrastructure-as-code programs. In *2025 IEEE/ACM International Workshop on Cloud Intelligence & AIOps (AIOps)*, 2025.

Manish Shetty, Yinfang Chen, Gagan Somashekar, Minghua Ma, Yogesh Simmhan, Xuchao Zhang, Jonathan Mace, Dax Vandevoorde, Pedro Las-Casas, Shachee Mishra Gupta, Suman Nath, Chetan Bansal, and Saravan Rajmohan. Building ai agents for autonomous clouds: Challenges and design principles. In *Proceedings of the 15th ACM Symposium on Cloud Computing*, 2024.

Kalahasti Ganesh Srivatsa, Sabyasachi Mukhopadhyay, Ganesh Katrapati, and Manish Shrivastava. A survey of using large language models for generating infrastructure as code, 2024. URL https://arxiv.org/abs/2404.00227.

Sanidhya Vijayvargiya, Xuhui Zhou, Akhila Yerukola, Maarten Sap, and Graham Neubig. Interactive agents to overcome ambiguity in software engineering, 2025. URL https://arxiv.org/abs/2502.13069.

Yiming Xiang, Zhenning Yang, Jingjia Peng, Hermann Bauer, Patrick Tser Jern Kon, Yiming Qiu, and Ang Chen. Automated bug discovery in cloud infrastructure-as-code updates with llm agents. In *2025 IEEE/ACM International Workshop on Cloud Intelligence & AIOps (AIOps)*, 2025.

Zhenning Yang, Archit Bhatnagar, Yiming Qiu, Tongyuan Miao, Patrick Tser Jern Kon, Yunming Xiao, Yibo Huang, Martin Casado, and Ang Chen. Cloud infrastructure management in the age of ai agents. *SIGOPS Operating Systems Review*, 2025a. doi: 10.1145/3759441.3759443. URL https://doi.org/10.1145/3759441.3759443.

Zhenning Yang, Hui Guan, Victor Nicolet, Brandon Paulsen, Joey Dodds, Daniel Kroening, and Ang Chen. Automated cloud infrastructure-as-code reconciliation with ai agents, 2025b. URL https://arxiv.org/abs/2510.20211.

Michael JQ Zhang, W. Bradley Knox, and Eunsol Choi. Modeling future conversation turns to teach LLMs to ask clarifying questions. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=cwuSAR7EKd.

Xuhui Zhou, Valerie Chen, Zora Zhiruo Wang, Graham Neubig, Maarten Sap, and Xingyao Wang. Tom-swe: User mental modeling for software engineering agents, 2025. URL https://arxiv.org/abs/2510.21903.

A Evaluation Metrics

This appendix provides formal definitions for the evaluation metrics. The evaluation pipeline has two stages: (1) construct graphs from specs and compute GED for the structure score, and (2) use the GED node alignment to compute embedding-based attribute similarity.

A.1 Preliminaries

Spec representation. Each spec consists of three fields: **Resources** ($\mathcal{R}$), a label-to-address mapping; **Topology** ($\mathcal{T}$), a label-to-dependency-labels mapping; and **Attributes** ($\mathcal{A}$), a label-to-key-value mapping. Labels are derived from Terraform instance names (e.g., `aws_vpc.main` $\rightarrow$ `main`), with the full address used on collision. Both specs undergo label normalization before comparison.

Resource type extraction. Given a Terraform address a, the resource type $\tau(a)$ is the provider-type prefix: e.g., $\tau(\texttt{aws\_vpc.main}) = \texttt{aws\_vpc}$. For data sources, the `data.` prefix is retained: $\tau(\texttt{data.aws\_iam\_policy\_document.p1}) = \texttt{data.aws\_iam\_policy\_document}$.

Graph construction. Each normalized spec is converted to a labeled directed graph $G = (V, E)$ where each resource label becomes a node with attribute $\text{type}(v) = \tau(\mathcal{R}[\text{label}])$, and each topology dependency becomes a directed edge.

A.2 Structure Score: Graph Edit Distance

We compute the GED between the reference and generated graphs using NetworkX's `optimize_edit_paths`, which iteratively yields improving solutions; we take the best found within a 30-second timeout per task. Node substitutions are free when the resource types match and cost 1 otherwise; node and edge insertions/deletions each cost 1. The structure score is normalized to $[0, 1]$:

$$S_{\text{struct}} = \max\left(0,\ 1 - \frac{\text{GED}}{|V_{\text{ref}}| + |E_{\text{ref}}| + |V_{\text{gen}}| + |E_{\text{gen}}|}\right) \tag{1}$$

where 1.0 indicates identical structure and 0.0 indicates maximum dissimilarity. The optimal edit path also produces a node alignment, which is reused for attribute comparison below.

A.3 Attribute Score: Embedding Similarity

For each aligned node pair, we serialize the resource type and attributes into a canonical string (keys sorted lexicographically, e.g., `type=aws_vpc, cidr_block=10.0.0.0/16`), embed both with a sentence transformer (`all-MiniLM-L6-v2`, 384-dim, L2-normalized), and compute cosine similarity (clamped to $[0, 1]$). Special cases:

- Both nodes have empty attributes: sim $= 1.0$.
- Unmatched reference nodes (deletions): sim $= 0$.
- Extra generated nodes (insertions): sim $= 0$.

The overall attribute score is the arithmetic mean across all nodes:

$$S_{\text{attr}} = \frac{1}{|\mathcal{N}|} \sum_{v \in \mathcal{N}} \text{sim}(v) \tag{2}$$

where $\mathcal{N}$ includes matched pairs, unmatched reference nodes, and extra generated nodes.