Automating Cloud Infrastructure Provisioning with Semantically-Enriched Large Language Models

Weslley Paulo*
wps3@cin.ufpe.br
Universidade Federal de Pernambuco
Centro de Informática - CIn
Recife, Pernambuco

Breno Vasconcelos*
bjrv@cin.ufpe.br
Universidade Federal de Pernambuco
Centro de Informática - CIn
Recife, Pernambuco

Carlos Ferraz*
cagf@cin.ufpe.br
Universidade Federal de Pernambuco
Centro de Informática - CIn
Recife, Pernambuco

ABSTRACT

The complexity of provisioning multi-cloud infrastructure has created a significant automation bottleneck, and while Large Language Models (LLMs) offer a promising solution, they consistently fail to generate reliable and deployable Infrastructure as Code (IaC) due to inherent ambiguity. To address this critical reliability gap, we propose a novel methodology that significantly improves IaC generation by augmenting LLM prompts with structured semantic context. Our approach utilizes OWL ontologies to formally model key infrastructure concepts, grounding the LLM in a machine-readable representation of the domain. This semantic enrichment provides the specific, structured context needed to resolve ambiguity and enhance the accuracy of the generated Terraform code. We evaluate our approach on the IAC-EVAL benchmark, comparing our semantically-enriched method against standard prompting strategies. Experimental results demonstrate a definitive improvement: our approach achieves a mean functional accuracy of **64.3%**, a **126.4%** increase over the baseline average of **28.4%**. Syntactic validity also improved dramatically, with Terraform plan validation rates increasing by an average of **29.6%**. These findings showcase that formal semantic grounding is a critical and highly effective technique for building reliable, LLM-driven automation for complex cloud environments.

KEYWORDS

Infrastructure as Code, Large Language Models, Prompt Engineering, Semantic Web, Ontologies, OWL, Cloud Computing, Code Generation, Natural Language Processing

1 INTRODUCTION

Recent advances in large language models (LLMs) enable the automation of software development tasks [19], including the generation of Infrastructure as Code (IaC) from natural language specifications. Automating this process is especially valuable in cloud environments, where the complexity and variability of infrastructure APIs among providers present significant challenges for manual configuration [13]. This automation becomes even more critical when organizations need to rapidly respond to changing operational conditions—such as service outages, performance degradation, regional availability issues, compliance changes, or cost optimization opportunities—that may necessitate immediate migration between cloud providers. These dynamic requirements demand infrastructure code that is not only functionally correct but also adaptable across different provider environments with minimal modification. However, while LLMs have demonstrated impressive capabilities in code generation [23, 33, 35], their outputs often lack consistency, domain grounding, and cross-provider portability—especially when prompts are vague, underspecified, or ambiguous.

In this context, our work investigates how semantically enriching LLM prompts can improve the generation of IaC artifacts. We propose a methodology that introduces structured knowledge into the prompting process via ontologies. Specifically, we leverage OWL (Web Ontology Language) [22] to model core infrastructure concepts such as virtual machines, instance types, cloud providers, regions, and pricing models. These ontologies encode the intent of infrastructure requests in a machine-readable, formally grounded representation, which is then processed into a structured summary and embedded into the prompt.

Our approach enhances the LLM's ability to interpret infrastructure requirements accurately and generate consistent code aligned with the user's intent. Instead of relying solely on free-text descriptions, the model receives additional contextual cues derived from ontology-based semantic relationships. This strategy is especially effective in reducing ambiguities [30] and enforcing structure in the generated IaC, allowing the model to better generalize across providers and configurations. In essence, we treat ontology-driven prompting as a form of soft constraint injection into the language generation pipeline.

To validate the effectiveness of our semantically-enriched methodology, we adopt the IaC-Eval benchmark [17]. Its comprehensive validation mechanisms and stratified complexity levels provide a robust and standardized testbed for quantitatively assessing improvements in the generation of functionally correct and consistent infrastructure code. This allows us to rigorously measure the impact of semantic grounding on model performance over a wide range of real-world scenarios.

The overall methodology is illustrated in Figure 1. The diagram illustrates our proposed process for enhancing IaC generation with LLMs through semantic enrichment. The workflow consists of five key stages:

(1) **Agnostic Infrastructure Ontology**: Formal representation of cloud infrastructure concepts using OWL ontologies that

*Both authors contributed equally to this research.

In: Proceedings of the Brazilian Symposium on Multimedia and the Web (WebMedia'2025). Rio de Janeiro, Brazil. Porto Alegre: Brazilian Computer Society, 2025.

ISSN 2966-2753

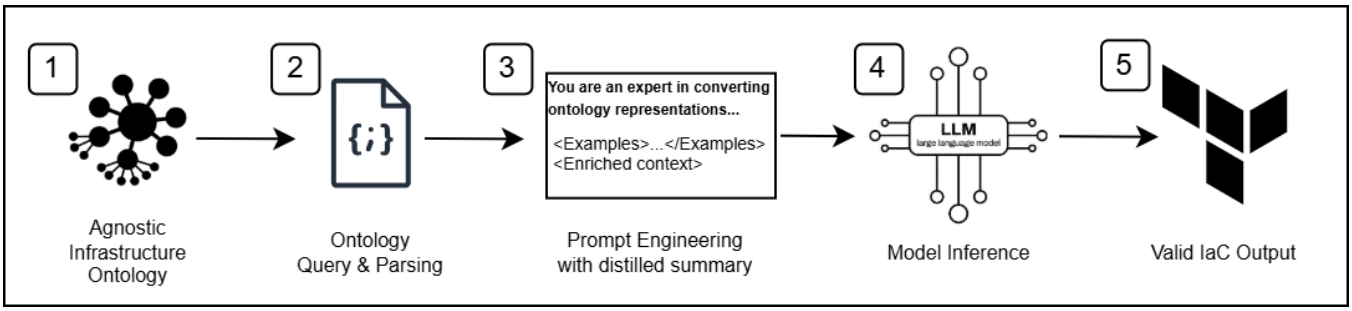

Figure 1: Overview of Semantically-Enriched IaC Generation Methodology.

model resources, relationships, and provider-specific mappings.

(2) **Ontology Query & Parsing**: Extraction and transformation of semantic data from the ontology into structured, machine-readable representations tailored to specific infrastructure scenarios.
(3) **Prompt Engineering**: Construction of enhanced prompts that combine natural language requirements with the structured semantic context derived from the ontology.
(4) **Model Inference**: Processing of the semantically-enriched prompt by the Large Language Model to generate IaC artifacts with improved domain awareness.
(5) **Valid IaC Output**: Production of syntactically correct and semantically appropriate IaC code that accurately reflects the infrastructure requirements.

2 BACKGROUND AND RELATED WORK

The intersection of cloud infrastructure automation and generative artificial intelligence is a rapidly expanding field of research. To contextualize our contribution, we organize the review around three core axes: (i) the use of LLMs for IaC generation (Section 2.1); (ii) semantic technologies for cloud resource modeling and management (Section 2.2); and (iii) emerging synergies that combine LLMs with formal ontologies and other structured knowledge sources to improve reliability and grounding (Section 2.3).

2.1 Large Language Models for Infrastructure as Code Generation

The advent of LLMs has revolutionized numerous domains, exhibiting proficiency in generating coherent content, including syntactically accurate code [27, 35]. Several studies [11, 15, 32] analyze the potential of LLMs to automate software engineering tasks, including the generation of IaC, with a significant focus on industry-standard tools like Terraform. The ability to automate IaC creation has received considerable attention, as it streamlines IT infrastructure management, fostering automation and consistency while mitigating manual errors [29]. This automation becomes even more critical as organizations adopt multi-cloud strategies, where the complexity of managing resources across multiple providers escalates [28].

However, studies evaluating LLM performance in this area reveal a fundamental weakness: despite their impressive capabilities, models often produce responses with inconsistencies or inaccuracies [31]. Accuracy and reliability decrease dramatically as the complexity of the requirements increases. The core problem, consistently identified in the literature [32], is the LLMs' lack of domain knowledge and their difficulty in handling the ambiguity inherent in natural language. Without structured context, models often "hallucinate" resources, use incorrect parameters, or fail to establish correct dependencies. This reliability gap is the main obstacle to adopting LLMs for IaC provisioning in production scenarios. Our work directly addresses this gap by proposing that the key to reliable IaC generation lies not just in smarter prompts, but in semantically-enriched prompts.

2.2 Semantic Technologies for Cloud Resource Management

Systematic reviews [1, 3] of the field demonstrate that ontologies provide robust solutions to persistent cloud challenges like provider interoperability, service discovery, and configuration management. The primary benefit highlighted by this line of research is the ability of ontologies to capture not only resources but also their intricate relationships, dependencies, and constraints. It is precisely this formal modeling capability that establishes the foundation of our methodology. While these prior works focused on using ontologies for decision support systems or direct management, our work advances this idea by using the ontology as a structured source of truth to ground and guide the code generation process of LLMs.

2.3 Enhancing LLM Generation with Formal Ontologies

To overcome the reliability limitations of LLMs, a promising research avenue has explored grounding these models in structured knowledge sources. The central hypothesis is that by providing LLMs with factual, semantic context, it is possible to significantly reduce ambiguity and the incidence of "hallucinations," thereby guiding the generation process toward more accurate and reliable outcomes.

Studies in adjacent domains have already validated the effectiveness of this synergy. In the field of Question-Answering (Q&A) systems [2, 5], for example, researchers demonstrated a substantial increase in the accuracy of LLM responses by using an ontology to

validate and repair model-generated queries. These studies establish a fundamental principle: the formal semantics of an ontology act as a "guardrail," constraining the LLM's possible output space to a set that is factually consistent with the domain model. Despite the success of this approach in other areas, its direct application to the domain of IaC generation remained, until now, underexplored. Our work positions itself precisely in this gap, adapting and empirically validating the technique of enriching prompts with context derived from formal ontologies (OWL) specifically for the generation of Terraform code.

3 METHODOLOGY

To validate our approach, we conducted experiments using stratified sampling of the 458 scenarios from the dataset. Each scenario consists of a specific infrastructure request and its corresponding valid Terraform implementation, which serves as the ground truth for validating the code generated by the LLMs. We selected approximately 30% of scenarios from each complexity level through stratification [10], and this approach was necessary for three main reasons:

(1) Each scenario required manual construction of OWL ontologies (in Turtle syntax), as it is an intensive process that demands specialized knowledge;
(2) Inference costs associated with the LLMs imposed practical limitations;
(3) For each scenario, we generated multiple samples (10) to ensure greater statistical robustness in the evaluation of results.

This experimental design resulted in approximately 23,000 total inferences across all model-strategy combinations, providing a comprehensive statistical basis for our comparative analysis.

3.1 IaC-Eval Assessment Framework

The framework incorporates three primary evaluation components:

(1) **Syntactic Validation**: Verifies whether the generated Terraform code is syntactically valid and can be processed by the `terraform plan` command. This basic validation ensures that the code respects the HCL (HashiCorp Configuration Language) [14] syntax and can be interpreted by Terraform.
(2) **Semantic Validation**: Assesses whether the generated code fulfills the user's infrastructure intent specification. IaC-Eval uses the Open Policy Agent (OPA) [9] to validate the correspondence between the user's intent specification (written in the Rego language) and a dependency graph extracted from the generated Terraform code. If no errors are found during this validation process, the IaC program is considered functionally correct.
(3) **Complementary Metrics**: Beyond the above validations, IaC-Eval incorporates additional metrics to evaluate code quality:
   - **BLEU**: Measures syntactic similarity by comparing n-gram overlap between generated and reference code. Though BLEU has recognized limitations when applied outside machine translation [20, 25], it provides a basic metric of textual similarity.
   - **CodeBERTScore**: Addresses BLEU's limitations by computing semantic similarity using contextual embeddings. Unlike BLEU's exact matching, CodeBERTScore leverages pretrained models to capture the functional equivalence of code snippets that may differ syntactically but accomplish the same tasks.

3.2 Experimental Protocol

Our evaluation focused on the most highly-rated and publicly available [8] LLM models, providing a comprehensive view of current LLM capabilities in assisting infrastructure-as-code generation. We compared two experimental configurations:

(1) **Baseline Approaches**: We utilized the three strategies provided by the IaC-Eval benchmark:
   - **Chain-of-Thought (COT)** [16]: Prompts that guide the model through a structured reasoning process, encouraging it to break down the task into logical steps.
   - **Few-Shot Prompting (FSP)** [7]: Prompts that include examples of similar infrastructure scenarios and their corresponding Terraform implementations, enabling the model to learn from examples.
   - **Zero-Shot**: Direct prompts with minimal instructions (e.g., "Create an Amazon Redshift cluster resource with a single node") without examples or step-by-step guidance.
(2) **semantically-enriched Variant**: Our proposed approach that augments prompts with semantically structured knowledge extracted from domain-specific ontologies.

3.3 Dataset

When implementing our 30% stratified sampling, we carefully maintained the original complexity distribution, ensuring that each level retained its proportional representation in the final dataset. This methodological approach preserved the diversity and representativeness of the original benchmark while making the study feasible within our resource constraints.

As shown in Table 1, the relative distribution by complexity level in our sample closely mirrors that of the complete dataset, providing a balanced foundation for evaluating model performance by the full spectrum of infrastructure complexity.

Table 1: Stratification of the IaC-Eval dataset by complexity level scenarios with 30% sampling.

Level	Original	Sample (30%)	Percentage
1	45	14	10.1%
2	95	29	20.9%
3	113	34	24.5%
4	58	17	12.2%
5	72	22	15.8%
6	75	23	16.5%
Total	458	139	100%

3.4 Ontology Construction

Based on the 139 stratified scenarios, a domain-specific ontology for IaC was constructed. The ontology is composed of three main parts for each scenario:

(1) **Core**: Responsible for defining the fundamental classes, properties, and relationships of the infrastructure domain. This foundational layer establishes the abstract conceptual model that serves as a provider-agnostic representation of cloud infrastructure components. The core ontology captures essential infrastructure concepts such as computational resources, networking components, storage systems, and security policies, along with their hierarchical relationships and interdependencies.

Listing 1: Core Ontology Example (ontology_core.ttl)

```
# --- Classes ---
infra:CloudResource rdf:type owl:Class .

infra:Network rdf:type owl:Class ;
    rdfs:subClassOf infra:CloudResource ;
    rdfs:label "Network" .

infra:VirtualMachine rdf:type owl:Class ;
    rdfs:subClassOf infra:CloudResource ;
    rdfs:label "Virtual Machine" .

# --- Object Properties (Relations) ---
infra:hasSubnet rdf:type owl:ObjectProperty ;
    rdfs:domain infra:Network ;
    rdfs:range infra:Subnet ;
    rdfs:label "has subnet" .

infra:usesSecurityGroup rdf:type owl:
    ObjectProperty ;
    rdfs:domain infra:VirtualMachine ;
    rdfs:range infra:SecurityGroup ;
    rdfs:label "uses security group" .
```

(2) **Mapping**: Contains the necessary mappings to the final resources in each cloud provider, enabling better contextualization and higher accuracy in artifact generation.

Listing 2: Provider Mapping Example (aws_mappings.ttl)

```
# AWS EC2 Instance Mapping
infra:aws_vm_mapping rdf:type owl:
    NamedIndividual , infra:ProviderMapping
    ;
    rdfs:comment "Secure and resizable
        compute capacity in the cloud" ;
    infra:mapsConcept infra:VirtualMachine ;
    infra:forProvider "AWS" ;
    infra:providerServiceName "EC2" ;
    infra:terraformResourceType "aws_instance
        " ;
    infra:keyTerraformArguments "ami", "
        instance_type", "
        vpc_security_group_ids" ;
    infra:typicallyConnectsTo infra:
        aws_sg_mapping, infra:
        aws_subnet_mapping ;
    infra:exposesVariable "instance_type", "
        image_id", "key_name" ;
    infra:suggestedOutput "ec2_public_ip", "
        service_url" .
```

(3) **Context**: Provides additional contextual information about the scenario, such as the specific cloud provider, region, and any relevant constraints or requirements.

Listing 3: Context Example (context.ttl)

```
# VPC Simple Example
infra:myVPC rdf:type owl:NamedIndividual ,
    infra:Network ;
    infra:name "my-vpc" ;
    infra:cidrBlock "10.0.0.0/16" ;
    infra:enableDnsSupport "true"^^xsd:
        boolean ;
    infra:hasSubnet infra:mySubnet .

infra:myVM rdf:type owl:NamedIndividual ,
    infra:VirtualMachine ;
    infra:name "my-web-server" ;
    infra:usesSecurityGroup infra:myWebSG ;
    infra:hasInstanceType infra:
        myInstanceType ;
    infra:locatedInZone infra:myZone ;
    infra:locatedInRegion infra:myRegion .
```

While other structured data formats like JSON schemas could enforce syntax, we selected ontologies as our semantic foundation due to their unique advantages in representing complex infrastructure domains [21]. Unlike simpler data structures, ontologies provide formal semantics with logical foundations that enable automated reasoning, consistency checking, and inference capabilities—critical for validating cross-provider compatibility. Ontologies excel at modeling complex relationships and constraints between infrastructure components, facilitating the explicit representation of dependencies that are often only implied in natural language descriptions. Furthermore, their extensible nature makes them ideal for the evolving cloud ecosystem, where new services and features are continually introduced.

3.5 Semantically-enriched Prompting

Our semantically-enriched prompting strategy involves augmenting the input prompts to the LLM with relevant semantic context derived from the OWL ontology. To make this knowledge accessible to LLMs, we developed a specialized parser implemented in Python using the RDFLib library [18] that transforms the formal OWL representations into organized, distilled knowledge representations. This parser performs targeted SPARQL [24] queries against the ontology, extracting precise provider-specific mappings and resource relationships while preserving essential semantic relationships.

For example, when a scenario involves AWS resources, the parser executes SPARQL queries like:

Listing 4: SPARQL Query Example for Provider Mapping

```
PREFIX infra: <http://www.example.com/ontologies/
    infrastructure#>

SELECT DISTINCT ?mapping ?mappedConcept ?p ?o
WHERE {
  ?mapping rdf:type infra:ProviderMapping ;
```

```
        infra:mapsConcept ?mappedConcept ;
        infra:forProvider "AWS" ;
        ?p ?o .
  FILTER(?p != rdf:type && ?p != infra:mapsConcept
        && ?p != infra:forProvider)
}
ORDER BY ?mapping ?p
```

This query extracts all mappings specific to AWS, including their corresponding Terraform resource types, required arguments, and interconnections with other resources. The parser then processes these results into a structured, human-readable format that summarizes each resource's characteristics:

Listing 5: Structured Resource Summary Example

```
Ontology Resource Name: "www"
Ontology Resource Type: DNSRecord
Ontology Resource arguments and variables:
    1. recordName = "example53.com"
    2. recordType = "A"
    3. belongsToZone = primary
    4. aliasedTo = main
    5. evaluateTargetHealth = true
Cloud provider: AWS
Cloud provider service: Route53
Terraform Resource type: aws_route53_zone
Key terraform arguments: "zone_id", "name", "type"
    , "alias"
Typically connects to: aws_route53_record
Resource example:
    resource "aws_route53_record" "www" {
      zone_id = aws_route53_zone.primary.zone_id
      name = "example.com"
      type = "A"
      alias {
        name = aws_elb.main.dns_name
        zone_id = aws_elb.main.zone_id
        evaluate_target_health = true
      }
    }
```

These structured summaries are then incorporated into the prompts, providing the LLM with explicit domain knowledge that includes:

- Precise Terraform resource types for the specific cloud provider
- Required and optional arguments for each resource
- Relationships and dependencies between resources
- Contextual examples that demonstrate proper syntax and structure

This approach grounds the LLM in accurate, provider-specific knowledge while maintaining a standardized format that models can easily process. By including these semantically rich descriptions, we significantly reduce ambiguity in the prompt interpretation phase, enabling the model to produce more syntactically valid and semantically appropriate Terraform configurations.

3.6 Model Configuration

To ensure reproducible outputs, we standardized inference parameters for all evaluated models. Modern LLMs provide hyperparameters [6] that control generation randomness:

- **Temperature**: Controls randomness in token selection (0-1 range)
- **Top-k/Top-p**: Limit candidate token sets during generation
- **Max tokens**: Defines maximum response length

Since top-k and top-p parameters are not available in all evaluated models, we used only the universally supported temperature [26] parameter to maintain experimental consistency.

We configured temperature to 0.1—a low value chosen to maximize determinism and reduce variability across runs, which is essential for code generation tasks where consistency matters more than creativity. The maximum number of tokens was set to 4096 to accommodate complex infrastructure specifications while preventing overly verbose outputs.

This configuration ensures reproducible results for different model architectures while maintaining the generation quality necessary for accurate IaC evaluation.

4 EVALUATION AND RESULTS

In this section, we present a comprehensive evaluation of our semantically-enriched approach against traditional prompting strategies over the 139 stratified scenarios. We selected four state-of-the-art LLMs: Anthropic Claude 3.7 Sonnet, Google Gemini 2.0 Flash, OpenAI GPT-4.1, and DeepSeek Reasoner, representing a diverse cross-section of contemporary foundation models with demonstrated code generation capabilities.

For each scenario, we validated the generated Terraform code against the IaC-Eval's validation mechanism. All performance metrics in the figures and tables below are presented as percentages from 0 to 100, where higher values indicate better accuracy.

4.1 Overall Performance

Figure 2 presents an aggregated comparison of the four prompting strategies for all models and complexity levels. This consolidated view allows us to assess the overall effectiveness of each approach independent of model-specific variations. The results reveal a performance gap between traditional prompting techniques and our semantically-enriched approach. While the three conventional strategies (Chain-of-Thought, Few-Shot and Zero-Shot prompting) cluster around similar performance levels with only minor variations between them, the enriched strategy demonstrates dramatic improvements on all metrics.

Particularly noteworthy is the significant improvement in IaC-Eval accuracy achieved by the enriched approach (64.3%) compared to the best-performing baseline strategy, Chain-of-Thought (28.4%), representing a 126.4% improvement. This substantial improvement in functional correctness is accompanied by corresponding gains in the BLEU score, where the enriched approach more than doubles the syntactic similarity to reference implementations. The smaller but still significant improvement in CodeBERTScore (F1) (from 84.0% to 88.6%) indicates that even traditional approaches maintain reasonable structural similarity to reference code, but the semantic enrichment helps models capture more nuanced aspects of the implementation. These consistent improvements in all evaluation dimensions suggest that ontological grounding provides comprehensive benefits to the code generation process rather than merely optimizing for specific metrics.

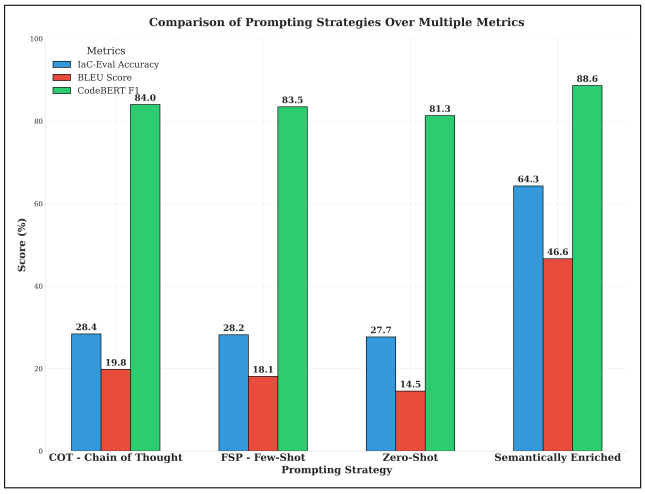

Figure 2: Comparison of Prompting Strategies by Evaluation Metric (%).

Table 2 presents a detailed breakdown of model performance using our semantically-enriched strategy:

Table 2: Overall performance metrics of enriched strategy prompting (%).

Rank	Name	BLEU	CodeBERTScore	IaC-Eval
1	**DeepSeek Reasoner**	46.81	**88.81**	**67.14**
2	GPT-4.1	46.48	88.74	65.00
3	Claude 3.7 Sonnet	**47.36**	88.25	62.86
4	Gemini 2.0 Flash	45.87	88.66	62.14

DeepSeek Reasoner achieves the highest functional correctness (67.14% IaC-Eval accuracy), while Claude 3.7 Sonnet demonstrates superior syntactic similarity (47.36% BLEU score). Notably, all models achieve remarkably consistent CodeBERTScore values (88.25% to 88.81%), indicating that semantic enrichment standardizes structural code quality across different model architectures. This consistency, combined with varying strengths in specific metrics, suggests that ontological grounding provides comprehensive semantic foundation while preserving individual model characteristics.

Table 3 presents the percentage of generated Terraform configurations that successfully passed the `terraform plan` validation command across all prompting strategies, indicating the code's basic syntactic validity.

Table 3: Terraform Plan Validation Accuracy for different prompting strategies (%).

Model	Zero-shot	COT	FSP	Enriched
DeepSeek Reasoner	57.86	**77.86**	72.86	**89.29**
Claude 3.7 Sonnet	**70.00**	76.43	**75.61**	87.14
GPT-4.1	52.86	56.43	53.57	84.29
Gemini 2.0 Flash	42.86	47.14	42.14	81.43

The Terraform plan validation results reveal several important insights about model capabilities. The semantically-enriched approach consistently produces the highest percentage of syntactically valid configurations across all models, with DeepSeek Reasoner achieving the best overall result at 89.29%. Claude 3.7 Sonnet demonstrates remarkable consistency, being the only model to exceed 70% validation in the zero-shot condition while maintaining strong performance across all strategies.

To fully assess the impact of our semantically-enriched approach, we compared the performance of all four prompting strategies for each model. Figure 3 presents these results, showing the IaC-Eval accuracy percentage for each model-strategy combination.

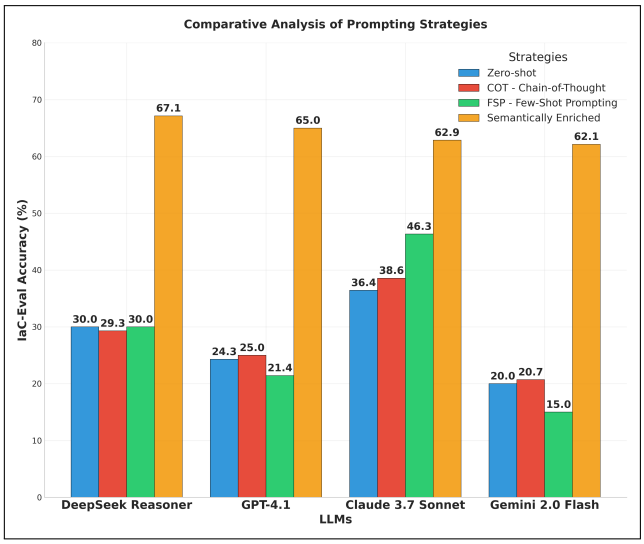

Figure 3: IaC-Eval Accuracy by Prompting Strategy and Model (%).

The results demonstrate a substantial improvement when using our semantically-enriched approach compared to traditional prompting strategies. All models show substantial gains in accuracy, with the most remarkable improvements observed in models that initially performed less effectively with standard prompting techniques. Specifically, our semantically-enriched approach improved DeepSeek Reasoner's accuracy by 37.10%, GPT-4.1 by 40.0%, Gemini 2.0 Flash by 41.4%, and Claude 3.7 Sonnet by 16.6% compared to their respective best baseline strategy performance.

These results confirm that providing structured semantic knowledge dramatically enhances the model's ability to generate functionally correct infrastructure code, with the most significant relative improvements observed in initially lower-performing models. This suggests that semantic enrichment particularly helps bridge knowledge gaps and reduce ambiguity for all models, regardless of their baseline capabilities.

4.2 Per Complexity Level Analysis

We conducted a complexity-level analysis to assess how performance varies with increasing infrastructure complexity—a critical consideration as generative models often struggle with context retention and hallucination in more complex scenarios. This granular

evaluation reveals where semantic enrichment provides the most significant benefits and identifies threshold points where additional support mechanisms may be required.

Figure 4 provides a comprehensive visualization of model performance by complexity level and prompting strategy. The figure reveals several critical insights about the effectiveness of semantic enrichment:

- **Distribution by Strategy:** Traditional strategies (COT (Figure 4.A), FSP (Figure 4.B), and Zero-Shot (Figure 4.C)) exhibit substantial variability in accuracy between models, with values typically ranging from 20% (Gemini) to 38% (Claude). In contrast, the semantically-enriched (Figure 4.D) approach demonstrates remarkably consistent performance among all models, with accuracy rates clustered between 62% and 67%, indicating that semantic grounding significantly reduces model-specific variance.
- **Complexity Degradation Pattern:** All prompting strategies show declining performance as infrastructure complexity increases, but with markedly different degradation rates. Traditional approaches experience precipitous drops, often falling below 0.1% accuracy at complexity levels 5-6. For instance, GPT-4.1 with zero-shot prompting declines from 42% at level 1 to merely 4% at level 6. The semantically-enriched approach exhibits a more graceful degradation, maintaining reasonable performance (33% with GPT-4.1) even at the highest complexity levels.
- **Resilience to Complexity:** The vertical spread between baseline and enriched approaches widens at higher complexity levels, suggesting that semantic enrichment provides increasingly valuable support as task complexity grows. This resilience is likely attributable to the structured semantic context, which provides clearer specifications for resource dependencies. These relationships become exponentially more intricate in complex scenarios, making the semantic guidance particularly valuable.
- **Consistency Benefits:** The narrower interquartile range for the enriched approach at all complexity levels indicates more predictable and deterministic outputs—a crucial characteristic for production infrastructure management where reliability is paramount.

These findings suggest that semantically-enriched prompting acts as an effective complexity buffer, partially insulating the generation process from the compounding ambiguities that typically challenge LLMs in complex infrastructure specification tasks. The consistent performance improvement for all models regardless of their baseline capabilities indicates that semantic enrichment represents a generalizable approach for enhancing LLM-driven IaC generation.

4.3 Prompting Strategy Consistency Analysis

To better understand the consistency and reliability of different prompting strategies, we conducted a statistical analysis over multiple runs of each approach. Figure 5 presents the distribution of IaC-Eval accuracy scores for all evaluated models, highlighting not just the average performance but also the variability and consistency of each prompting approach:

The boxplot visualization in Figure 5 reveals differences in both performance and consistency between traditional and semantically-enriched approaches. The relatively wide interquartile ranges for traditional strategies indicate considerable performance variability, with maximum accuracy rarely exceeding 45% in experimental runs.

In stark contrast, the semantically-enriched approach achieves a mean accuracy of 64.3%—representing more than a 100% relative improvement over conventional strategies. Beyond this accuracy gain, the boxplot shows a much tighter distribution for the enriched strategy, indicating not only superior performance but significantly greater consistency and reliability. This reduced variability is particularly valuable in production environments where predictable infrastructure provisioning is essential. The results strongly suggest that ontological grounding provides LLMs with the precise semantic context needed to navigate the complex requirements of valid Terraform specifications, effectively reducing ambiguity in instruction interpretation while supplying the technical context necessary for accurate code generation.

5 CONCLUSION

This work introduced a novel approach for enhancing IaC generation through semantic enrichment of LLM prompts using formal ontologies. Our methodology leverages OWL ontologies to model cloud infrastructure concepts, relationships, and provider-specific mappings in a structured, machine-readable format. By transforming these semantic representations into enriched prompts, we provided LLMs with explicit domain knowledge that significantly improved their ability to generate valid and functionally correct Terraform code.

The empirical evaluation over 139 scenarios from the IaC-Eval benchmark demonstrated that this approach substantially outperforms traditional prompting strategies. The quantitative results reveal improvements in all evaluated metrics. Our semantically-enriched approach achieved a mean IaC-Eval accuracy of 64.3%, representing a 126.4% improvement over the baseline average of 28.4% for traditional prompting strategies. This improvement was consistent for all four evaluated LLMs, which saw accuracy gains ranging from 26% to over 42% compared to their best baseline performance.

Similarly large improvements were observed in syntactic validity, with Terraform plan validation rates increasing by an average of 29.64% for all models, reaching as high as 89.29% for DeepSeek Reasoner.

The complexity-level analysis revealed that semantic enrichment provides particularly significant benefits for moderately complex infrastructure scenarios (levels 3-4), where the performance gap between enriched and traditional approaches is most pronounced.

Beyond raw performance improvements, our approach demonstrated significantly enhanced consistency and reliability, as evidenced by the narrower interquartile range in the statistical analysis. This reduced variability is particularly valuable in production environments, where predictable and deterministic infrastructure provisioning is essential. The results confirm our hypothesis that formal semantic grounding can effectively bridge the gap between high-level infrastructure requirements and low-level implementation details, providing LLMs with the contextual anchoring needed

Figure 4: Performance of Models by Complexity Level and Strategy.

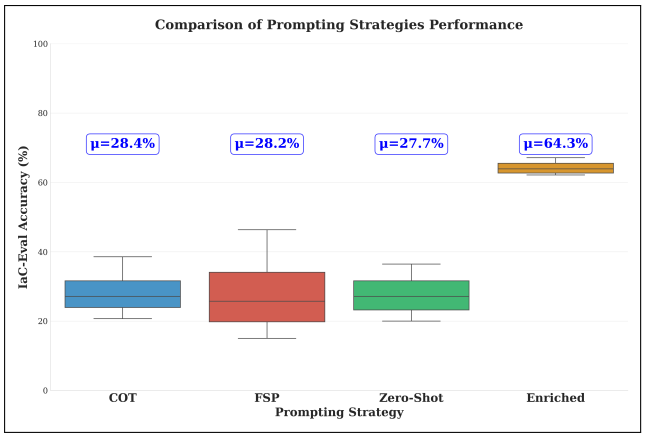

Figure 5: Distribution of IaC-Eval accuracy by prompting strategy.

to generate more accurate, consistent, and functionally correct IaC artifacts. This work contributes to the reliable automation of cloud infrastructure provisioning, particularly in multi-cloud environments where consistency and provider-specific accuracy are paramount.

5.1 Future Work

Building upon these findings, our future work will explore several promising avenues to further enhance LLM-driven IaC generation:

- **Learning from Failed Cases:** A deeper qualitative analysis of scenarios where ontology enrichment did not yield improvements will be crucial. This involves categorizing types of failures (e.g., incorrect resource dependencies, invalid arguments, cross-provider inconsistencies) to identify specific gaps in the ontological modeling or limitations in the LLM's ability to interpret complex semantic relationships. This will guide iterative improvements to the ontology and prompt design.
- **User Interface and Interaction Models:** Exploring how these semantically-enriched prompts can be integrated into user-friendly interfaces will be important. This includes designing interaction models that guide users in formulating precise natural language requests and visualizing the underlying semantic context, potentially allowing for user-driven adjustments to the ontological input.
- **Domain-Specific Fine-Tuning:** We plan to investigate efficient fine-tuning techniques [4, 12] with datasets annotated using our ontological framework, potentially creating IaC-specific model variants with enhanced domain understanding [34]. This approach could reduce context window requirements, improve handling of provider-specific nuances, and potentially maintain high performance even with less detailed prompts. A comparative analysis between prompt engineering and fine-tuning approaches would provide valuable insights into the most effective strategies for different complexity levels and deployment scenarios.

These next steps are designed to address the identified limitations and further solidify the role of formal semantic grounding in developing more reliable, adaptable, and interpretable LLM-driven DevOps workflows.

REFERENCES

[1] JohnBosco Agbaegbu, Oluwasefunmi Tale Arogundade, Sanjay Misra, and Robertas Damaševičius. 2021. Ontologies in Cloud Computing—Review and Future Directions. *Future Internet* 13, 12 (2021). https://doi.org/10.3390/fi13120302

[2] Dean Allemang and Juan Sequeda. 2024. Increasing the LLM Accuracy for Question Answering: Ontologies to the Rescue! arXiv:2405.11706 [cs.AI] https://arxiv.org/abs/2405.11706

[3] D Androcec, N Vrcek, and J Seva. 2012. Cloud computing ontologies: A systematic review. *Proceedings of the third …* (2012). https://www.researchgate.net/profile/Neven-Vrcek/publication/268062404_Cloud_Computing_Ontologies_A_Systematic_Review/links/54ca54120cf2c70ce521c39b/Cloud-Computing-Ontologies-A-Systematic-Review.pdf

[4] D.M. Anisuzzaman, Jeffrey G. Malins, Paul A. Friedman, and Zachi I. Attia. 2025. Fine-Tuning Large Language Models for Specialized Use Cases. *Mayo Clinic Proceedings: Digital Health* 3, 1 (2025), 100184. https://doi.org/10.1016/j.mcpdig.2024.11.005

[5] Christina Antoniou and Nick Bassiliades. 2025. Utilizing LLMs and ontologies to query educational knowledge graphs. In *Proceedings of the 28th Pan-Hellenic Conference on Progress in Computing and Informatics (PCI '24)*. Association for Computing Machinery, New York, NY, USA, 287–295. https://doi.org/10.1145/3716554.3716598

[6] Chetan Arora, Ahnaf Ibn Sayeed, Sherlock Licorish, Fanyu Wang, and Christoph Treude. 2024. Optimizing large language model hyperparameters for code generation. *arXiv preprint arXiv:2408.10577* (2024).

[7] Patrick Bareiß, Beatriz Souza, Marcelo d'Amorim, and Michael Pradel. 2022. Code Generation Tools (Almost) for Free? A Study of Few-Shot, Pre-Trained Language

Models on Code. arXiv:2206.01335 [cs.SE] https://arxiv.org/abs/2206.01335
[8] Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E. Gonzalez, and Ion Stoica. 2024. Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference. arXiv:2403.04132 [cs.AI] https://arxiv.org/abs/2403.04132
[9] Cloud Native Computing Foundation. 2025. Open Policy Agent (OPA). https://www.openpolicyagent.org/.
[10] William G. Cochran. 1977. *Sampling Techniques* (3rd ed.). Wiley.
[11] Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M. Zhang. 2023. Large Language Models for Software Engineering: Survey and Open Problems . In *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)*. IEEE Computer Society, Los Alamitos, CA, USA, 31–53. https://doi.org/10.1109/ICSE-FoSE59343.2023.00008
[12] Yingqiang Ge, Wenyue Hua, Kai Mei, jianchao ji, Juntao Tan, Shuyuan Xu, Zelong Li, and Yongfeng Zhang. 2023. OpenAGI: When LLM Meets Domain Experts. In *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (Eds.), Vol. 36. Curran Associates, Inc., 5539–5568. https://proceedings.neurips.cc/paper_files/paper/2023/file/1190733f217404edc8a7f4e15a57f301-Paper-Datasets_and_Benchmarks.pdf
[13] MIchele Guerriero, Martin Garriga, Damian A. Tamburri, and Fabio Palomba. 2019. Adoption, Support, and Challenges of Infrastructure-as-Code: Insights from Industry. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 580–589. https://doi.org/10.1109/ICSME.2019.00092
[14] Hashicorp Group, an IBM company. 2025. HCL - HashiCorp Configuration Language). https://github.com/hashicorp/hcl/blob/main/hclsyntax/spec.md.
[15] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large Language Models for Software Engineering: A Systematic Literature Review. *ACM Trans. Softw. Eng. Methodol.* 33, 8, Article 220 (Dec. 2024), 79 pages. https://doi.org/10.1145/3695988
[16] Harshit Joshi, José Cambronero Sanchez, Sumit Gulwani, Vu Le, Ivan Radiček, and Gust Verbruggen. 2023. Repair is nearly generation: multilingual program repair with LLMs. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence (AAAI'23/IAAI'23/EAAI'23)*. AAAI Press, Article 573, 10 pages. https://doi.org/10.1609/aaai.v37i4.25642
[17] 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. 2024. IaC-Eval: A Code Generation Benchmark for Cloud Infrastructure-as-Code Programs. In *Advances in Neural Information Processing Systems*, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (Eds.), Vol. 37. Curran Associates, Inc., 134488–134506. https://proceedings.neurips.cc/paper_files/paper/2024/file/f26b29298ae8acd94bd7e839688e329b-Paper-Datasets_and_Benchmarks_Track.pdf
[18] D Krech. 2006. Rdflib: A python library for working with rdf. *Online https://github.com/RDFLib/rdflib* (2006).
[19] Michael R. Lyu, Baishakhi Ray, Abhik Roychoudhury, Shin Hwei Tan, and Patanamon Thongtanunam. 2025. Automatic Programming: Large Language Models and Beyond. *ACM Trans. Softw. Eng. Methodol.* 34, 5, Article 140 (May 2025), 33 pages. https://doi.org/10.1145/3708519
[20] Benjamin Marie. 2022. BLEU: A Misunderstood Metric from Another Age. https://towardsdatascience.com/bleu-a-misunderstood-metric-from-another-age-d434e18f1b37. Accessed November 2022.
[21] Beniamino Di Martino, Giuseppina Cretella, and Antonio Esposito. 2013. Semantic and Agnostic Representation of Cloud Patterns for Cloud Interoperability and Portability. In *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, Vol. 2. 182–187. https://doi.org/10.1109/CloudCom.2013.123
[22] DL McGuinness and F Van Harmelen. 2004. OWL web ontology language overview. *W3C recommendation* (2004). https://static.twoday.net/71desa1bif/files/W3C-OWL-Overview.pdf
[23] Senthamarai N, Jeyaselvi M, and Hemamalini V. 2025. Automatic Cloud Formation Using LLM. In *2025 International Conference on Intelligent and Cloud Computing (ICoICC)*. 1–6. https://doi.org/document/11052114
[24] Bastian Quilitz and Ulf Leser. 2008. Querying Distributed RDF Data Sources with SPARQL. In *The Semantic Web: Research and Applications*, Sean Bechhofer, Manfred Hauswirth, Jörg Hoffmann, and Manolis Koubarakis (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 524–538.
[25] Ehud Reiter. 2018. A Structured Review of the Validity of BLEU. *Computational Linguistics* 44, 3 (Sept. 2018), 393–401. https://doi.org/10.1162/coli_a_00322
[26] Matthew Renze. 2024. The Effect of Sampling Temperature on Problem Solving in Large Language Models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 7346–7356. https://doi.org/10.18653/v1/2024.findings-emnlp.432
[27] Robson Santos, Italo Santos, Cleyton Magalhaes, and Ronnie de Souza Santos. 2024. Are We Testing or Being Tested? Exploring the Practical Applications of Large Language Models in Software Testing . In *2024 IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE Computer Society, Los Alamitos, CA, USA, 353–360. https://doi.org/10.1109/ICST60714.2024.00039
[28] Dhruv Seth, Harshavardhan Nerella, Madhavi Najana, and Ayisha Tabbassum. 2024. Navigating the Multi-Cloud Maze: Benefits, Challenges, and Future Trends. *International Journal of Global Innovations and Solutions (IJGIS)* (jun 10 2024). https://ijgis.pubpub.org/pub/plmsrs5y.
[29] Tomasz Szandała. 2025. AIOps for Reliability: Evaluating Large Language Models for Automated Root Cause Analysis in Chaos Engineering. In *Computational Science – ICCS 2025 Workshops*, Maciej Paszynski, Amanda S. Barnard, and Yongjie Jessica Zhang (Eds.). Springer Nature Switzerland, Cham, 323–336.
[30] Anfu Tang, Laure Soulier, and Vincent Guigue. 2025. Clarifying Ambiguities: on the Role of Ambiguity Types in Prompting Methods for Clarification Generation. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Padua, Italy) *(SIGIR '25)*. Association for Computing Machinery, New York, NY, USA, 20–30. https://doi.org/10.1145/3726302.3729922
[31] Liang Zhang, Katherine Jijo, Spurthi Setty, Eden Chung, Fatima Javid, Natan Vidra, and Tommy Clifford. 2024. Enhancing Large Language Model Performance To Answer Questions and Extract Information More Accurately. arXiv:2402.01722 [cs.CL] https://arxiv.org/abs/2402.01722
[32] Quanjun Zhang, Chunrong Fang, Yang Xie, Yaxin Zhang, Yun Yang, Weisong Sun, Shengcheng Yu, and Zhenyu Chen. 2024. A Survey on Large Language Models for Software Engineering. arXiv:2312.15223 [cs.SE] https://arxiv.org/abs/2312.15223
[33] Tianyi Zhang, Shidong Pan, Zejun Zhang, Zhenchang Xing, and Xiaoyu Sun. 2025. Deployability-Centric Infrastructure-as-Code Generation: An LLM-based Iterative Framework. arXiv:2506.05623 [cs.SE] https://arxiv.org/abs/2506.05623
[34] Jiawei Zheng, Hanghai Hong, Feiyan Liu, Xiaoli Wang, Jingsong Su, Yonggui Liang, and Shikai Wu. 2024. Fine-tuning Large Language Models for Domain-specific Machine Translation. arXiv:2402.15061 [cs.CL] https://arxiv.org/abs/2402.15061
[35] Álvaro Barbero Jiménez. 2024. An evaluation of LLM code generation capabilities through graded exercises. arXiv:2410.16292 [cs.SE] https://arxiv.org/abs/2410.16292