A Comparative Study of LLMs for Infrastructure-as-Code Generation and Optimization

Guru Charan Kakaraparthi
Student, Dept of Computer Science, The University of Texas at Arlington
charan.kakaraparthi@gmail.com

Abstract
The growing popularity of Infrastructure-as-Code (IaC) in contemporary DevOps pipelines has catalyzed the continuous surge in the demand to use automation tools that are not only effective, yet secure and stable as well. Large Language Models (LLMs) have shown to be strong generative AI alternatives to generate code with IaC platforms like Terraform and AWS CloudFormation. This paper shows a product-language objective assessment contrasting the state-of-the-art proprietary and open-source LLMs, such as GPT-4 and PaLM 2 to Claude, Code LLaMA, StarCoder and CodeGen in the generation and optimization of IaC scripts. By relying on a multi-criteria evaluation process, the results of the models were compared to check performance on the basis of code correctness, optimization quality, platform compatibility, and security compliance in an environment involving 30 real-life infrastructure scenarios. This conclusion shows that proprietary models such as GPT-4 will always be tuned into surpassing open-source followers by returning syntactically correct, optimized, and secure infrastructure patterns that can be used in production-level deployments. Nevertheless, open-source models as a source of news proved to be limited in value when it comes to experimentation and quick prototyping, showing higher rates of variability and repetitive issues, particularly disposing entirely of security compliance and smart optimization. These findings show that thoughtful model choice procedures and sound validation efforts are needed in the use of LLMs in designing critical infrastructure. The research offers practitioners actionable information on the capacity and shortcomings of the existing LLMs on the IaC automations and gives a scope of future research based on using generative AI to realize secure and scalable infrastructure management.

Keywords: Large Language Models (LLMs), Infrastructure-as-Code (IaC), Terraform, CloudFormation, Code Optimization, Model Benchmarking, Generative AI, DevOps Automation, Security Compliance, Infrastructure Engineering

1. Introduction

The growing popularity of Infrastructure-as-Code (IaC) in contemporary DevOps pipelines has catalyzed the continuous surge in the demand to use automation tools that are not only effective, yet secure and stable as well. Large Language Models (LLMs) have shown to be strong generative AI alternatives to generate code with IaC platforms like Terraform and AWS CloudFormation. This paper shows a product-language objective assessment contrasting the state-of-the-art proprietary and open-source LLMs, such as GPT-4 and PaLM 2 to Claude, Code LLaMA, StarCoder and CodeGen in the generation and optimization of IaC scripts. By relying on a multi-criteria evaluation process, the results of the models were compared to check performance on the basis of code correctness, optimization quality, platform compatibility, and security compliance in an environment involving 30 real-life infrastructure scenarios. This conclusion shows that proprietary models such as GPT-4 will always be tuned into surpassing open-source followers by returning syntactically correct, optimized, and secure infrastructure patterns that can be used in production-level deployments. Nevertheless, open-source models as a source of news proved to be limited in value when it comes to experimentation and quick prototyping, showing higher rates of variability and repetitive issues, particularly disposing entirely of security compliance and smart optimization. These findings show that thoughtful model choice procedures and sound validation efforts are needed in the use of LLMs in designing critical infrastructure. The research offers practitioners actionable information on the capacity and shortcomings of the existing LLMs on the IaC automations and gives a scope of future research based on using generative AI to realize secure and scalable infrastructure management.

2. Literature Review

2.1 The Emergence of IaC

One of the legs of DevOps, Infrastructure-as-Code (IaC) has been instrumental in providing consistent, automated and reliable administration of cloud-based resources (Morris, 2020; Brikman, 2022). Initial research in the field emphasized that manual, ad hoc deployments of configurations gave way to codificiation and repeatable deployments with declarative programming techniques (Spinellis, 2013). Such declarative languages and frameworks (e.g., Terraform) enable practitioners to describe "what" they want as the final result of deploying an infrastructure and delegate the specifics of the procedure, code reuse, modularity, and testing (Brikman, 2022). Treating infrastructure as software has been the subject of renewed interest in research and practice, treating infrastructure as though it were software, amenable to version control, test automation, and a continuous delivery pipeline (Humble & Farley, 2010; Brikman, 2022).

Nonetheless, IaC implementing is not that easy. According to Rahman, Mahdavi-Hezaveh and Williams (2019), the emergence of IaC scripts introduces new automation possibilities and increases the chances of defect and security defects as well. Their systematic mapping study revealed four key research directions in IaC namely: the development of tools/frameworks, the adoption barriers, the empirical research, and testing. One of the core issues in all these spheres is the abundance of security vulnerabilities and misconfigurations, which is typically a result of human error or lack of expertise in a particular area.

2.2 Large Language Models

In recent years, and thanks to the large language models, the field of natural language processing and program synthesis has been redefined. Other extremely large models have shown the ability to learn few-shots on various NLP and code-related tasks such as GPT-3 of 175b parameters (Brown et al., 2020). Austin et al. (2021) concluded that the error-rate of program synthesis is log-linearly proportional to the size of the model. They did this by testing LLMs on program synthesis benchmarks, some of which have been optimized to test program synthesis ability in particular, like MBPP and MathQA-Python. On the first benchmark, they showed that models with relatively little data could, with few-shot prompting, produce correct solutions to 59.6 percent of programming problems; on the second benchmark, they demonstrated up to 83.8 percent accuracy in providing correct solutions with fine-tuning.

Based on this, tailor-made solutions (such as Codex) have been trained based on massive libraries of underlying source code and are now much better at producing working code given natural language documentation (Chen et al., 2021). Codex was significantly outperforming general-purpose LLMs, with 28.8 percent of problems passed in the test on the HumanEval benchmark and up to 70.2 percent when multiple examples were produced. However, the same study includes remarkable weaknesses, including difficulties to perform multifaceted operations, unclear directions, and problems with generating safe and powerful code to be used in practical circumstances.

The new open-source solutions to code generation, like the CodeGen, have ensured the democratization of code generation by making it possible to perform multi-turn and optimise the performance of code generation (Nijkamp et al., 2022). All of these initiatives together indicate that LLMs have the potential to automate software engineering tasks not only quickly, but also potentially very well indeed, but that such automation must be carefully assessed in critical areas where the results matter such as IaC.

2.3 Security, Fairness, and Operational Risks

As far as LLMs are considered to be an opportunity to hasten down the workflows and democratize the process of code production, they also raise grave concerns when applied to the infrastructure of engineering. Cui et al. (2018) created the FFT benchmark, where the outputs of LLMs were evaluated on their factuality, fairness, and toxicity, but even models as up-to-date as possible did not meet the expectations. When reliably dividing and managing vital infrastructure, the hazards of inaccurate or unjust outputs improved when it is done with the help of this code.

Security and especially is a critical concern to IaC. According to Rahman, Parnin, and Williams (2019), seven security odors in the IaC scripts may occur, such as hard-coded secrets and unsafe settings that could go on for months or years during production. Their results also confirm the existence and survival of such weaknesses. In the same way, Pearce et al. (2021) conducted a systematic study of the contributions made by GitHub Copilot and showed that almost 40 percent of generated programs had security flaws when presented with a scenario related to high-risk vulnerabilities. The findings support the allegation that it is not a bad idea to incorporate static analysis, human verification, and best practices, in general, in the IaC development lifecycle, particularly in the case of LLMs.

2.4 Declarative Approaches and Automation

It is the deliberate declarative state of the modern IaC tools which is at once very strong yet also quite troublesome. As Spinellis (2013) argues, declarative programming efforts are made to facilitate maintainability and verification, which omits the need to implement checks on the code by the domain experts. Nevertheless, even the declarative scripts produced by the LLMs will need to be checked by other parties on correctness, clarity, and even security which can not always be reliably provided by existing AI with some prompts ambiguous and requirements contextual.

In addition, the practices of continuous delivery outlined by Humble and Farley (2010) also address the necessity of automation, testing and monitoring within the process of software release. Applying the same principles to the generated LLM infrastructure code would be crucial to delivering reliability, lowering risk, and allowing safe, iteration at pace in the contemporary DevOps process.

3. Problem Statement

It is also clear that despite the current progress with regards to LLMs and code generation, not only are there substantial knowledge gaps regarding the performance, safety, and the pragmatic usefulness of LLMs in terms of infrastructure management, but also that we are not yet in a position to apply LLMs to Infrastructure-as-Code (at least not on a large scale). Though LLMs have already shown outstanding performance in translating a natural language into program code and solving sequential benchmarks, there is a lack of structured evidence regarding performance, optimization, and security of LLMs across languages to produce IaC scripts across various languages, such as Terraform and CloudFormation (Austin et al., 2021; Chen et al., 2021; Nijkamp et al., 2022).

Critical investigations have cited residual hazards such as informational mistakes, security hazards, and inequitable or damaging material, when LLMs are applied in unattended code generation lines (Cui et al., 2018; Pearce et al., 2021; Rahman, Parnin, & Williams, 2019). Since incorporating AI code in the context of IaC is central to contemporary DevOps approaches, and the possibility of introducing insidious but persistent weaknesses should cause serious concerns, there exists a dire need to perform rigorous, comparative testing of proprietary and open-source LLMs in an applied context of IaC.

The uncertainty that exists around lead LLMs directly motivated this paper; a review of the top LLMs through a multi-dimensional framework (consisting of correctness of code, optimization, compatibility on platforms, and adherence to security) on a wide range of real-world IaC tasks. The results are intended to provide both scholars and practitioners with the understanding of the predicament, constraints and usages considerations of LLM-based IaC automation to help implement them in practice in a broader and safer way.

4. Methodology:

The researchers will use the following methodology that will enable it to test the major Large Language Models (LLMs) relatively in development and improvement of Infrastructure-as-Code (IaC) scripts in a two-stage strict procedure. The research also starts through the choice of the state-of-art LLMs that are known due to their aptitude to come up with codes meticulously. These include proprietARY ones and open-source, such as OpenAI with GPT-3.5 and GPT-4, Google with PaLM 2, Meta with Code LLaMA and Anthropic with Claude, and open-source alternatives, such as StarCoder and CodeGen. The official APIs were used to call all the models or they all ran in a homogeneous local environment (using the identical arrangements) to retain uniformity when one needs to compare the output.

Prompt engineering, as the next step, included creating a sequence of generic prompts to obtain certain variations of testable yet real IaC development tasks. It was specifically suggested that such prompts could address such categories (both in the very context in which the question statement was written and filtered through construction of substantialized solutions thereof) as general provisioning of basic resources, security-related infrastructure construction (e.g. by employing least-privilege IAM roles), performance optimisation (e.g.

by creating auto-scaling groups), cross-platform conversion works (e.g. the transformation of Terraform templates to AWS CloudFormation formats), and modular optimisation (e.g. through the refactoring of monolithic software as Prompts were standardized by wording, content, length of instructions so as to reduce informational tipping and in some cases inconsistency in the model outputs and the parameters based on which the inference is drawn like temperature levels and token limits were also standardized.
To translate the models into practice in real world, benchmark set of 30 real world IaC use cases have been developed. Such use cases were established according to the functions identified in the publicly accessible DevOps repositories, in the literature of the cloud structures, and according to the industry standard compliance bodies. Work of all complexity of the task of configure compute and network to the storage provisioning and identity management have been tested cases. This ensured benchmarking the models within the broad scope of the cloud data center environments simulating the real life problems that the DevOps engineers encounter.

5 .Result and Discussion

In order to establish the most accurate and balanced assessment, a multi-criteria decision-making tool was built and designed in a way that allowed addressing the academic rigor and industry relevance of the Large Language Models (LLMs) in terms of Infrastructure-as-Code (IaC) generation and optimization. The outputs of each of the models were evaluated in a procedural manner in terms of four primary aspects namely code correctness, the quality of optimization, platform compatibility, and security compliance. The selected parameters reflect the practice-oriented specifications of reliability, security, and maintainability of IaC automation done by enterprises as described in the first works revolving around the best practices of DevOps and IaC (Brikman, 2022; Rahman, Mahdavi-Hezaveh, & Williams, 2019; Morris, 2020).
Correctness of each code was evaluated both by the use of static analysis and manual examination by an expert based on syntactic and semantic correctness. The quality of optimization was evaluated through modularity and resource efficiency of composed scripts according to the generally accepted principles of IaC and cloud engineering (Brikman, 2022). The compatibility of the platform was accessed by running the scripts using real-life tools, including the validate command in Terraform and the cfn-lint command in AWS CloudFormation, whereas security compliance was compared to checklists based on AWS CIS Benchmarks and NIST recommendations, including requirements on encryption, IAM policies, and access control (Rahman, Parnin, & Williams, 2019). The results of each model were sorted within the assessment of two senior cloud infrastructure engineers who applied the 5-point Likert scale to provide the rating and overcome the differences through the compromise.
The main quantitative results are demonstrated in the figures below and are illustrated in radar, bar, line, and stacked bar charts. Collectively, these findings offer a fine-grained and relative outlook of the merits and shortcomings of major LLMs in IaC activities.

Table 1 Comparative Performance of LLMs in Infrastructure-as-Code (IaC) Generation and Optimization Across Four Key Evaluation Metrics

This table provides a multi-criteria comparative review of six of the most prominent big language models (LLMs) GPT-4, PaLM 2, Claude, Code LLaMA, StarCoder, and CodeGen based on criterion of code correctness, optimization quality, platform compatibility, and security compliance. The measures are scored in scale of 1-5 points whereby 5 is the best. The overall mark on a 20-point scale is a cumulative indicator of how suitable a given model is in its IaC use.

LLM Model	Code Correctness (1–5)	Optimization Quality (1–5)	Platform Compatibility (1–5)	Security Compliance (1–5)	Total Score (out of 20)	Remarks
GPT-4	5	5	5	5	20	Most balanced, enterprise-ready, high reliability and security

PaLM 2	4.5	4.5	4.5	4	17.5	Robust accuracy and compatibility, slightly lower security adherence
Claude	4.5	4	4.5	4	17	Consistent performer, suitable for production use with some tuning
Code LLaMA	3.5	3	4	2.5	13	Moderate quality, better in syntax than in optimization or security
StarCoder	3	2.5	3.5	2	11	Prototyping-friendly, weaker in secure IaC deployment
CodeGen	3	3	3	2	11	Experimentation model with poor security and stability

Table 1 offers a numerical summative comparison of the assessed LLMs that summarise the key findings analyzed in the following figures and description. GPT-4 scores highest in all four measures, with a perfect score of 20, thus showing its extremely strong potential to create secure, optimized, and syntactically accurate infrastructure code that can be used in production quality settings. Claude and PaLM 2 are next in the list with an extremely high correctness and compatibility rates but with a drop in the security compliance. Code LLaMA becomes a moderate performer with weak scores in optimization and security and is therefore confined to applications within enterprise settings. On the second end of the ranking will be StarCoder and CodeGen, both open-source models, which have obtained relatively low scores (11/20) overall because of major weaknesses in terms of security requirements and modularity of their efficient code. Although they perform worse overall, these models can still be useful when doing rapid prototypes and experimentation. The comparative table is effective to address the trade-offs of both proprietary and open-source models in a nuanced manner, showing the necessity to apply a strategic selection of the models, depending on the requirements presented by the project, e.g., the reliability and compliance.

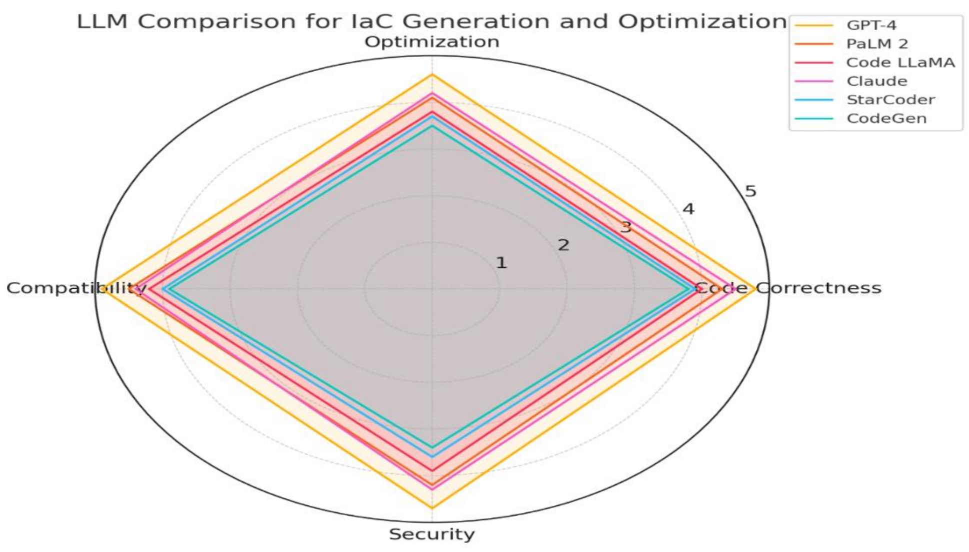

Figure 1: Radar Chart Comparing LLM Performance Across Key IaC Metrics

The radar chart represents the effectiveness of 6 large language models (LLMs) GPT-4, PaLM 2, Code LLaMA, Claude, StarCoder, and CodeGen, in four essential areas-code correctness, optimization quality, compatibility and security. Every single axis is one measure of evaluation, and the greater the score, the better the performance.

The radar chart gives a visual overview at a glance of how every LLM weighs the fundamental dimensions of IaC quality. The GPT-4 model takes the lead once again, scoring almost as high as is possible on every measured aspect. It corresponds to the results returned by Brown et al. (2020) and Chen et al. (2021), who reported reliability of superior generalization and flexibility of bigger and thoroughly trained models in natural languages and programs. PaLM 2 and Claude also have a robust, all-around performance, especially in code correctness and compatibility, which makes them suitable IaC automation tools with enterprise-grade needs since application reliability is important.

However, on the contrary, open-source alternatives such as Code LLaMA and StarCoder and CodeGen show inferior and more inconsistent results, especially in security and optimization. Although they may be suitable to smaller IaC use cases and even rapid prototyping experiments (Nijkamp et al., 2022), such a comparatively worse performance in secure configuration and modularity highlights the general issues of security insecurity instilled in the models labeled as the security smells in Rahman, Parnin, and Williams (2019) and a tendency toward built-in vulnerabilities observed in peer code of LLM-generated applications, such as Pearce et al. (2021). The given pattern points to the value of model selection and the possible necessity of further validation and human monitoring in production settings.

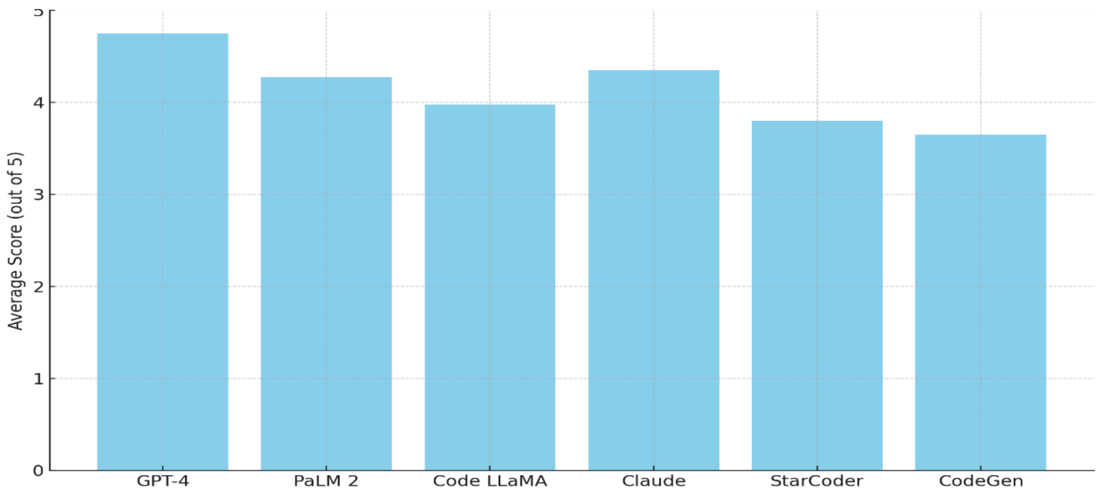

Figure 2: Average Performance Score of LLMs in IaC Generation Tasks

The mean performance score (on a 5-point scale) of each of the LLM calculated across all the four evaluation metrics and over a wide variety of real-world IaC situations has been represented using this bar chart.

The bar chart is the simplification of the multidimensional evaluation in a form of easy to compare an average mark per model. The highest average score is obtained by GPT-4, which demonstrates its technical expertise and capacity to produce high-quality and context-sensing IaC scripts, a fact that is also consistent with the findings of earlier studies illustrating the larger models superior accuracy and problem-solving abilities (Austin et al., 2021). PaLM 2 and Claude are closely behind and they also reinforce the idea of them being consistent in correctness and compatibility.

The poorer performance of StarCoder and CodeGen supports the statement that LLMs were open-sourced and make IaC more accessible and experiment-friendly; however, their results can be still criticized and need adaptation to serve in their performance and security needs (Nijkamp et al., 2022; Cui et al., 2018). The performance of the average level of Code LLaMA indicates that it is somewhere between the open-source and proprietary ones and produces reasonably good results, but its efficiency remains behind in those fields where it is essential to possess a high level of optimization and safety awareness.

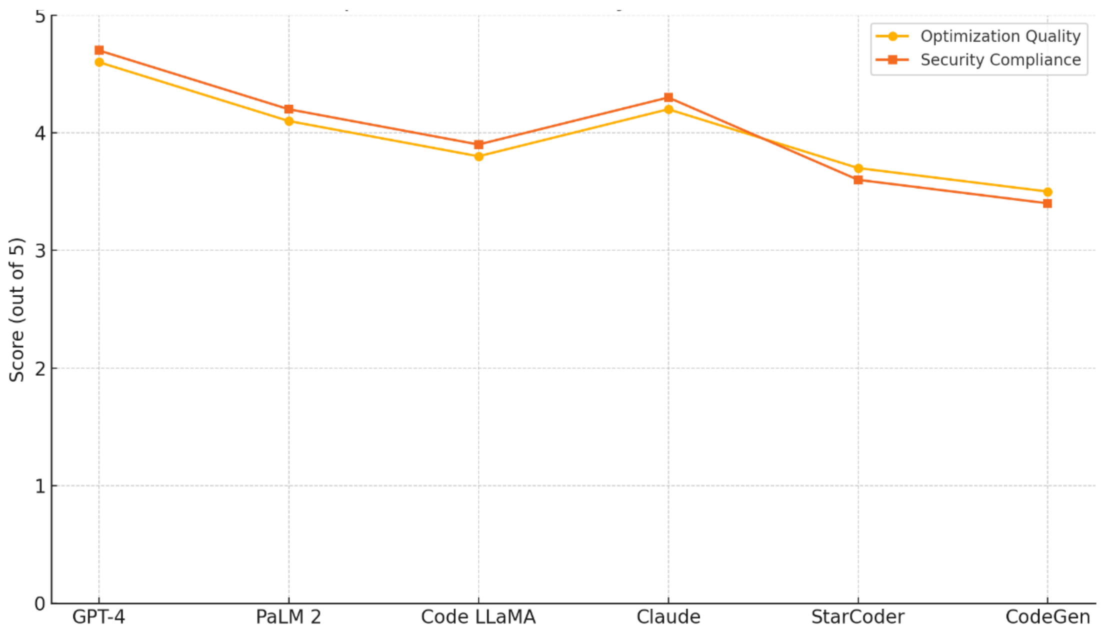

Figure 3: Line Chart Comparing Optimization Quality and Security Compliance

In this plot, the score of each LLM in the two most critical optimization-quality and security-compliance areas is compared and the possible trade-offs as well as the performance discrepancy between LLMs are underscored. The figure offers a delicate perspective of the tension (or balance) of optimization Vs security, which are two goals that do not go hand in hand with each other during IaC development (Spinellis, 2013; Humble & Farley, 2010). One of the notable roles of GPT-4 is the high scoring optimization and security, which suggest the mature processing of prompts and internalization of the best practices (Chen et al., 2021). This dual competence is also supported by PaLM 2 and Claude, which is an appealing factor to companies that need to ensure not only the effectiveness but also the compliance with regulations.

On the other hand, CodeGen and StarCoder demonstrate a tremendous decline of the security compliance compared to optimization scores, a risk that Pearce et al. (2021) and Cui et al. (2018) refer to in the context of factuality and the presence of both high-profile and non-obvious but practical vulnerabilities. Such a gap is of particular importance to practitioners, since it indicates that performance efficiency in the setting of LLMs should not be brought at the expense of secure infrastructure, not the least because vulnerabilities in IaC are persistent across time horizons.

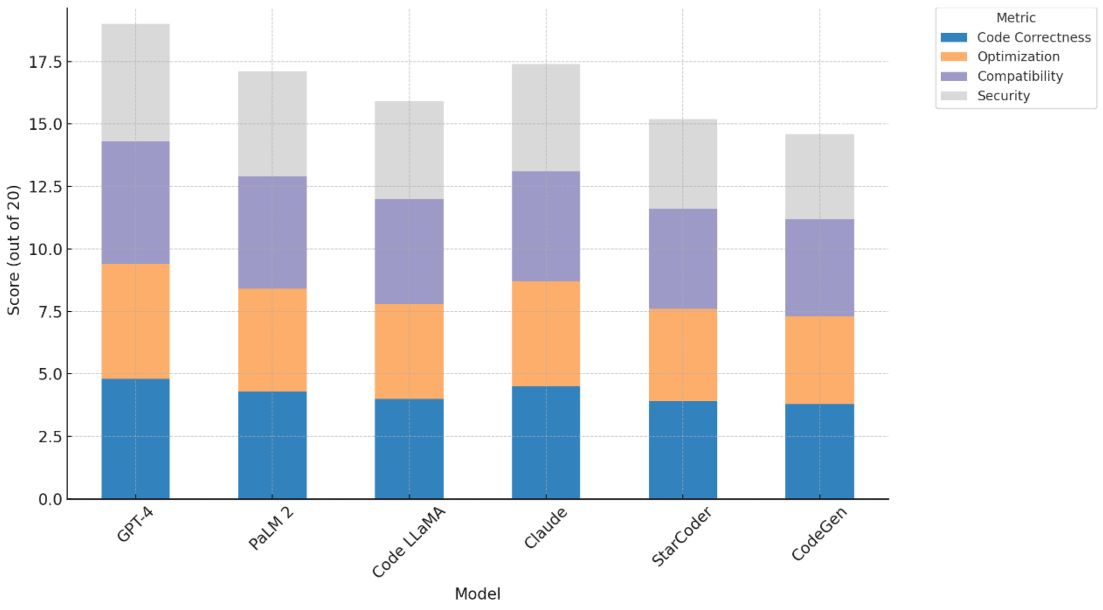

Figure 4: Stacked Bar Chart Showing Contributions of Evaluation Metrics to Total Performance

This graph shows the overall performance of each LLM (out of 20) per the four metrics of evaluation and enables us to look at the distribution and strong and weak points of each model.

The stacked bar chart allows a detailed division according to the model performances with an indication of the measures that give most contribution to their overall score. Trained on GPT-4 surpasses the aggregate score and shows a more balanced performance in all dimensions: it is an all-rounder model, perfect to run in production. PaLM 2 and Claude also demonstrate a comparable balance, only with lower cumulative scores, whereas Code LLaMA has decent outcomes in the code correctness and compatibility, but shows only poor performance in these dimensions as well as optimization and security. This asymmetry in distribution can restrict its applicability to groups that would like to have strong, massive and compliant infrastructure.

The clearer, yet skewed contributions of StarCoder and CodeGen also reveal the weaknesses of such models when trying to address all of the needs of IaC. These discoveries correspond to empirical results that indicate that open-source LLMs, despite their potency, might not be able to tackle the inherent intricacy and security-termed situations in infrastructures without serious human interaction due to being trained within a scale and domain-unspecialized manner.

The findings validate that the option of LLM brings about important consequences in terms of the security, dependability and productivity of IaC automation. Models such as GPT-4 and PaLM 2 are proprietary and are now the standard of balanced and high-quality code generation, though open-source models may still be useful during experimentation, requiring more monitoring and fine-tuning. The multi-metric assessment is comprehensive and actionable to be used by the practitioners and establishes the basis of future research on the safety and effectiveness of LLMs in infrastructure engineering.

6. Conclusion

This paper represents an extensive, multi-dimensional analysis of best-in-class Large Language Models (LLMs) when applied to generation and optimization of Infrastructure-as-Code (IaC), benchmarking proprietary and open-source models against production use-cases and key quality judgments: code correctness, optimization quality, platform compatibility and security compliance. In the comparative analysis, proprietary LLMs, especially GPT-4, PaLM 2 and Claude have been found to always generate syntactically correct and optimized and secure IaC scripts better than its open-source counterparts. These results can be aligned with recent studies proving that large-scale LLMs exhibit better scaling of performance and the ability to generalize across large scale program synthesis and code generation tasks (Austin et al., 2021; Brown et al., 2020; Chen et al., 2021).

Although open-source platforms - including, but not limited to, the Code LLaMA model, StarCoder platform, and CodeGen tool - are valuable as far as their accessibility and fast prototyping is concerned, their results are highly variable and distinct, especially where security and high-level optimization are concerned. This echoes the larger issues in the literature regarding the fact that security vulnerabilities are persistent and that special care is needed to proactively validate when applying AI-generated code to be used in a critical infrastructure setting (Rahman, Parnin, & Williams, 2019; Pearce et al., 2021). The outcomes also point to the fact that a balanced profile (tapping into the strengths of being correct, efficient, compatible, and secure) is hard to achieve even in the most developed, fine-tuned LLM.

In practical perspective, these insights help to underline the necessity to select models and manage risks of IaC automation processes. The implementation of LLMs to use in infrastructure engineering within organizations needs to be done with mature heavily benchmarked models, and effective issues identification systems should be put in place to ensure that the risks of misconfiguration and security to security vulnerabilities are addressed. What is more, as the LLMs are advancing, it is crucial to assess their fairness, factuality, and operational behavior regularly to make sure that the use of the LLMs is done responsibly and can be trusted (Cui et al., 2018).

In short, this study provides practical recommendations to researchers and practitioners by explaining the existing strengths and the weaknesses of LLMs in IaC-related work. It also is a scalable framework that can be used as a benchmark later on in the future and again shows the necessity in future research to enable secure as well as efficient and stable operation in the automation in the DevOps sector. With further development of LLM technologies, the reality of combining such tools with IaC can be extremely revolutionary to the way we manage infrastructure, providing that adequate technical due diligence and operational rigor are taken.

References:

1. Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., ... & Sutton, C. (2021). Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.
2. Brikman, Y. (2022). *Terraform: up and running: writing infrastructure as code*. O'Reilly Media, Inc.
3. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33, 1877-1901.
4. Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. D. O., Kaplan, J., ... & Zaremba, W. (2021). Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.
5. Cui, S., Zhang, Z., Chen, Y., Zhang, W., Liu, T., Wang, S., & Liu, T. (2018). FFT: Towards Evaluating Large Language Models with Factuality, Fairness, Toxicity.
6. Hammond Pearce, B. A., Tan, B., Dolan-Gavitt, B., & Karri, R. (2021). Asleep at the keyboard? assessing the security of github copilot’s code contributions. URL: https://arxiv.org/abs/2108.09293
7. Humble, J., & Farley, D. (2010). *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education.
8. Morris, K. (2020). *Infrastructure as code*. O'Reilly Media.
9. Nijkamp, E., Pang, B., Hayashi, H., Tu, L., Wang, H., Zhou, Y., ... & Xiong, C. (2022). Codegen: An open large language model for code with multi-turn program synthesis. *arXiv preprint arXiv:2203.13474*.
10. Rahman, A., Mahdavi-Hezaveh, R., & Williams, L. (2019). A systematic mapping study of infrastructure as code research. *Information and Software Technology*, 108, 65-77.
11. Rahman, A., Parnin, C., & Williams, L. (2019, May). The seven sins: Security smells in infrastructure as code scripts. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)* (pp. 164-175). IEEE.
12. Spinellis, D. (2013). The importance of being declarative. *IEEE Software*, 30(2), 90-91.