FDMA Point-to-Multi-Point Fibre Access System for Latency Sensitive Applications

Intent-Based Kubernetes Configuration via LLMs: Current Trends and Open Challenges

Alessio Sacco, Cristian Zilli, Guido Marchetto
Department of Control and Computer Engineering, Politecnico di Torino, Italy
Emails: {alessio_sacco, cristian.zilli, guido.marchetto}@polito.it

***Abstract*—The advent of Large Language Models (LLMs) is progressively transforming how complex tasks across various domains can be automated, with a notable potential impact on cloud computing operations. In this domain, LLMs might be used, for example, to configure Kubernetes (K8s) clusters via the generation of manifest files – structured configuration files defining the containerized environment. However, despite the considerable advances in LLMs' text generation, this task conceals several challenges that prevent operators from achieving a fully automated process. In this paper, we present the current trends in solving these gaps, quantitatively evaluate the accuracy of LLM-based approaches to generate K8s manifests starting from human intents, and discuss open challenges that make benchmarking and automation still complex. Experiments over three open-source LLMs demonstrate how intent-based K8s manifest generation can be effectively achieved through model fine-tuning, but also what open issues remain and must be addressed prior to having an autonomous and self-healing K8s infrastructure managed via Agentic AI.**

I. Introduction

Kubernetes (K8s), the default open-source cloud platform, enables automating the deployment, scaling, and management of containerized (or microservice-based) applications [1]. Although the K8s platform simplifies running and managing workloads of all sizes and styles, it still involves complex manual configurations typically through the definition of *manifest* files, structured YAML-style files where main deployment features are mentioned [2]. The definition of such features, along with opportune network and application policies, demands significant expertise and can be challenging even for experienced users, resulting in the typical "trial and error" process [3].

The emergence of powerful Large Language Models (LLMs) (*e.g.,* Meta's Llama 3 [4], GPT-4o [5], Deepseek [6]), represents a disruptive innovation if they are used to simplify cloud operations and make them accessible to a broader range of users, letting operators write intents (*i.e.,* declarative set of goals and outcomes that an infrastructure should meet) rather than configuration files [7], [8]. As LLMs see a wide range of applications, such as natural language generation, understanding, text summarization, classification, and code generation [9], their adoption also looks very promising for creating autonomous self-healing clouds, where AI-driven approaches can detect, localize, and mitigate faults in K8s environments with minimal human intervention. However, despite the growing importance of cloud computing and the demonstrated capabilities of LLMs in automated code generation, there remain notable challenges in designing and using an appropriate process specifically targeting cloud-native applications. Also, the performance characteristics of different LLMs in this context are not yet well understood. Establishing a rigorous and representative pipeline is essential to enable meaningful comparisons, guide model development, and advance the state of the art in LLM-driven code generation for cloud-native systems.

In this paper, we present our approach towards an LLMs-driven K8s environment, where the orchestrator is capable of making real-time decisions to ensure adherence to user intents. By harnessing the capabilities of potentially multiple LLMs, the solution can make cloud computing operations more intuitive and accessible, covering the entire deployment cycle. To this end, we demonstrated how fine-tuning [10], a known technique used to customize pre-trained LLMs, can make the models domain-specific and refine their understanding and performance. We empirically observed over three open-source LLMs, namely Mistral-7B [11], DeepSeek-Coder-6.7B [12], and LLaMA3-8B [4], that when fine-tuned over a cloud-specific dataset, their generative capabilities improve and in some cases double. These obtained results indicate that this approach is promising, but at the same time, there remain open challenges, such as the fact that fine-tuning is often expensive in terms of resources and time, and performance starts decreasing when implications are applied (e.g., LoRA [13]). Similarly, fine-tuning requires a considerable dataset, which is not always available and which should be tailored to the specific task the LLM is solving. Such a dataset is also needed for opportune benchmarking activities and to develop a holistic framework that can allow more LLM agents to interact dynamically with the cloud and proactively intervene.

The rest of the paper is structured as follows. Section II introduces the intent-based paradigm and how it aligns with the needs of cloud automation. Section III outlines the fine-tuning process for pre-trained LLMs, while Section IV illustrates the results of our fine-tuned LLMs. Section V examines key theoretical and practical challenges that must be addressed for effective integration and evaluation of LLM agents in cloud environments and Section VI concludes the paper.

II. LLM to automate K8s deployment

A. Overall Structure

Intent-based cloud programming is a paradigm that focuses on specifying what the desired outcome is, rather than explicitly defining how to achieve it. Instead of writing low-

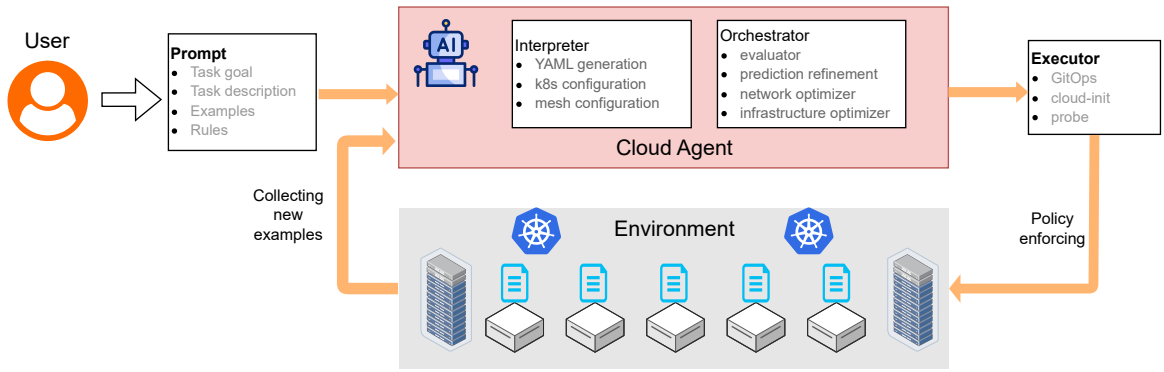


Fig. 1: The envisioned LLMs-enabled system where the cloud agent coordinates interactions between the K8s system components and serves as the core logic interface. The user engages with this interface through a problem description (as a prompt) that conditions the subsequent instructions and configurations.

level code to manage cloud infrastructure and workflows, programmers declare their high-level goals, possibly in natural language (e.g., deploying a scalable web application or ensuring compliance with security policies), and the cloud platform automatically determines the optimal configuration and actions to meet those intents. This approach leverages automation, orchestration, and AI-driven decision-making to abstract complexity, reduce operational overhead, and increase agility. It aligns with trends like Infrastructure as Code (IaC) (*e.g.,* Terraform [14]), serverless computing, and policy-as-code, but shifts the focus even further from manual configuration to goal-oriented declarations. To this end, the operator writes intents (*i.e.,* declarative set of goals and outcomes that the infrastructure should meet) rather than a working code [8]. One notable approach is Intent-Based Networking (IBN) [15], where network configuration and management are simplified to have minimal external intervention (e.g., Cisco IBN [16] and LLNet [17]). This approach promotes automation, consistency, and resilience through feedback-driven mechanisms, offering a scalable means to reduce complexity and facilitate human–system interaction in dynamic operational environments.

Cloud computing operations often involve complex manual configurations, particularly in service deployment of containerized environments (*e.g.,* microservices). K8s offers an interface where administrators declare the desired state of the system through config files in YAML format, a structured file describing the main deployment features. YAML's simplicity, readability, and support for hierarchical data make it ideal for describing Kubernetes resources like Pods, Deployments, and Services. Each YAML file outlines the structure and settings of a resource, allowing Kubernetes to continuously reconcile the actual cluster state with the desired configuration.

Building on this paradigm, large language models (LLMs) can further simplify the configuration process. By introducing an LLM as an intermediary between the administrator and the `kubectl` tool, natural language input from the administrator can be translated into the corresponding YAML configuration. This YAML file is then submitted to the Kubernetes API server, enabling a more intuitive and accessible workflow for defining system behavior [18]. As summarized in Fig. 1, we envision a system composed of multiple LLMs, capable of interpreting human intents, translating them into K8s-specific config files, orchestrating the cloud environment with these files, and ensuring compliance with the intents through frequent monitoring. Our proposed architecture constitutes a closed-loop automation system for K8s cloud environments, which naturally fits within the vision of autonomous infrastructures controlled by AI Agents (AIOps [19]), where the responsibility of implementing and maintaining services is increasingly shifted from humans to automated systems. The primary objective is to enable self-healing cloud environments where AI-powered systems can automatically detect, pinpoint, and resolve faults with little to no human involvement.

B. Intent-Based K8s Configuration: An Example

Listing 1: An example of a user input.

```
You are a cloud-native engineering expert. Given the
    question, respond exclusively with complete and
    correctly formatted YAML configuration files
    for Kubernetes. If details are missing, assume
    the most logical defaults.
Question:
Create a YAML for a pod named redis. Pod runs a
    single container with image redis. Use emptyDir
    volume mounted at /data/redis, persisting
    throughout the pod's life even if containers are
    restarted.
```

Listing 2: The desired output for input in Listing 1.

```
apiVersion: v1
kind: Pod
metadata:
name: redis
spec:
    containers:
        - name: redis # *
    image: redis
    volumeMounts:
        - name: redis-storage # *
    mountPath: /data/redis
    volumes:
        - name: redis-storage # *
    emptyDir: {}
```

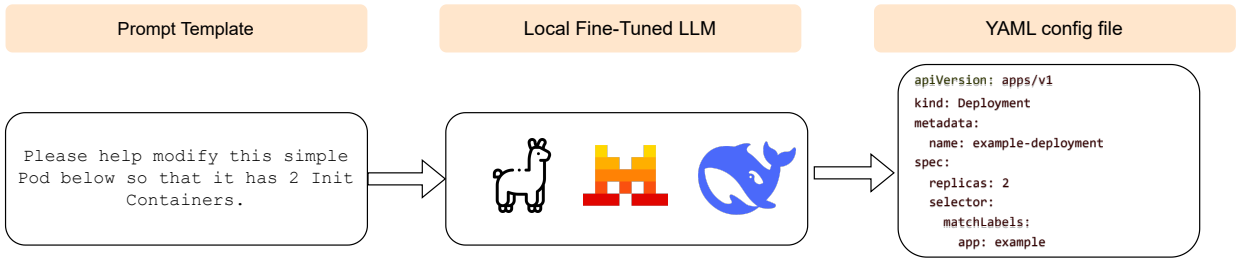


Fig. 2: An example problem LLM-aided YAML generation, including a problem specification in natural language, a fine-tuned LLM, and the YAML output from the LLM.

In this paper, we specifically focus on the intent expression, translation, and activation process, trying to evaluate how a specialized (*e.g.,* fine-tuned as described in Section III) LLM can interpret human language (Fig. 2). The main idea is to obtain an LLM that generates K8s manifest files (YAML-compliant) starting from a high-level user intent expressed in natural language. Although the figure provides an example of how this interpretation and generation work, we also present a simple example illustrating how a fine-tuned LLM can serve as an abstraction layer between a human user and a Kubernetes cluster. As shown in Listing 1, an administrator submits a prompt having: (i) a brief role prompting section [20], useful for setting the tone and context of the interaction for the LLM, and (ii) the deployment requirements using unstructured natural language. The corresponding YAML configuration, generated by the LLM, reflects these specifications (Listing 2 in this example). For the model to fulfill this role, it is expected to (i) comprehend the intent of the prompt, (ii) infer and complete missing details where necessary, and (iii) generate a valid, deployable YAML configuration that satisfies the stated requirements.

III. Fine-tuning existing LLMs

LLMs have demonstrated impressive capabilities in natural language understanding, reasoning, and text generation across a broad range of general-purpose tasks. However, their performance often degrades when applied to domain-specific problems, where the pre-trained knowledge may prove insufficient.To overcome this limitation, it becomes necessary to augment the capabilities of pre-trained LLMs. Several strategies have been explored for this purpose, including fine-tuning, few-shot prompting, Chain-of-Thought (CoT) prompting [21], and Retrieval-Augmented Generation (RAG) [22]. Among these methods, fine-tuning, and specifically supervised fine-tuning (SFT), is particularly effective when a labeled dataset is available [10]. Fine-tuning involves taking a language model that has been trained on general language tasks and further training it on a specialized dataset tailored to a specific domain. This process allows the model to better grasp the nature of the problem, thereby enhancing its task-specific performance.

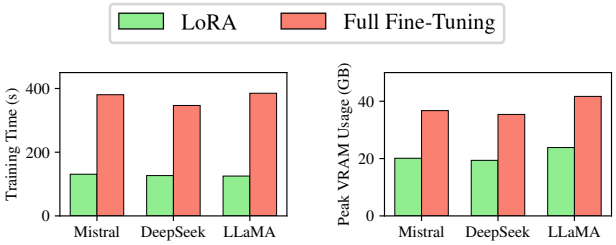
In the context of our work, which focuses on generating YAML configurations for K8s, SFT is deemed the preferable approach given the availability of a labeled dataset. Indeed, the alternative model customization techniques were considered less suitable for this task: few-shot prompting is ineffective due to the high variability in Kubernetes configurations and the limited generalizability of small example sets [18]; CoT prompting is less applicable to configuration files generation as little to none sequential reasoning is necessary; and RAG is not feasible in this setting, given the absence of a structured and comprehensive external knowledge base relevant to Kubernetes configuration.

The fine-tuning process is comprised of a sequence of steps: (i) a forward pass is performed using the dataset, (ii) a loss function, typically cross-entropy, is computed to quantify the discrepancy between the model's predictions and the ground-truth labels, (iii) the loss gradient is propagated backward through the network, and (iv) the model's trainable parameters are updated based on this gradient.

While effective, fine-tuning an LLM presents significant practical challenges, particularly as model size increases and memory and storage demands become critical, given the necessity to update potentially tens to hundreds of billions of parameters. This results in substantial memory consumption to store gradients for each parameter at every training step, and increases computational overhead. Low-Rank Adaptation (LoRA) [13] addresses these issues by freezing the original model parameters during fine-tuning and introducing a set of lightweight, trainable low-rank matrices, referred to as "adapters." These adapters are integrated into the model architecture prior to training and are the only components updated through backpropagation, thereby significantly reducing the memory footprint and computational cost associated with full-parameter fine-tuning. LoRA allows for control over the learning process through hyperparameters such as (i) rank r, which determines size and consequently expressiveness of the low-rank adapters, and (ii) α, a factor used to scale the magnitude of weight updates undergone by the model, and therefore control the influence of LoRA on the original model.

IV. Results

To assess the effectiveness of fine-tuning LLMs in this context, we consider three architectures: (i) Mistral-7B [11], (ii) DeepSeek-Coder-6.7B [12], and (iii) LLaMA3-8B [4]. We perform both full model fine-tuning and LoRA fine-tuning and



(a) Per-Epoch Training Times (b) Peak VRAM Usage

Fig. 3: Comparison between LoRA fine-tuning and full fine-tuning, in terms of (a) per-epoch training times and (b) peak VRAM usage.

evaluate the differences between the two approaches in terms of training times, VRAM usage and overall performance. In our configuration, the LoRA (see Section III) rank (*r*) and scaling factor (*alpha*) are both set to 16, with adaptations applied to the models' attention layers. The training phase is set to a duration of 5 epochs with both approaches; we settled for a low number of epochs to avoid overfitting the small data set. The learning rate *lr* is set to 0.0002 and batch size is set to 2. All models are fine-tuned on 75% of a publicly available dataset [18], which contains natural language prompts describing desired Kubernetes pod configurations paired with their corresponding YAML files. The remaining 25% of such a dataset is reserved for evaluation. Because the dataset is labeled, we use a supervised fine-tuning approach, using the HuggingFace Transformers [23] ecosystem for model access, training, and efficient hardware utilization.

We start measuring the per-epoch training times and VRAM usage, reporting results for the LoRA setting and the complete model fine-tuning in Figure 3. Our setup consists of a single server equipped with one Nvidia A40 GPU. Comparing the training time of LoRA and a full model fine-tuning, we can observe that LoRA speeds up the process by a factor of ∼2.7–3 on average for the three LLMs. Furthermore, LoRA reduces the memory footprint by ∼45%.

Next, we evaluate the accuracy of LLMs in generating config files. As in [18], we evaluate generative models using three metrics: (i) *BLEU* [24], a precision-oriented metric that assesses n-gram overlap between the generated and reference texts. It incorporates a brevity penalty to penalize excessively short outputs and aggregates n-gram scores using a geometric mean, yielding a value between 0 and 1, where higher scores indicate closer alignment with the reference; (ii) *Edit Distance* based on Gestalt Pattern Matching [25], which quantifies the number of line insertions and deletions required to transform the generated output into the reference. This score is normalized to fall within the range $[0, 1]$, with higher values indicating better performance; and (iii) *Key-Value Match (KVM)*, recent papers [26] dealing with YAML files proposed to only consider the match for the keys. As in [18], we consider the wildcard version, i.e., Key-Value Wildcard Match (KV-WCM), a YAML-aware evaluation metric that compares key-value pairs in a lenient manner, allowing for minor, non-functional differences such as formatting or ordering. Allowing to specify what matters and what does not as annotation, KV-WCM also produces a score between 0 and 1, where higher values reflect greater semantic similarity.

We report in Fig. 4 the accuracy of (i) the vanilla, (ii) LoRA fine-tuned, (iii) the full LLM fine-tuning in terms of the three mentioned metrics. As can be observed, fine-tuning consistently enhances model performance across all metrics, with LoRA degrading the overall performance. However, the degree of improvement varies from case to case, depending on metric and model: an outstanding example of this is observed with the KV-WCM metric (Figure 4c), where the LoRA fine-tuned and fully fine-tuned Mistral models, respectively, achieve a $\sim 2.2\times$ and $\sim 2.4\times$ improvement over the baseline, while LLaMA's score improves by $\sim 2\times$ with LoRA and $\sim 2.35\times$ with full tuning. DeepSeek, on the other hand, only observes marginal gains, at most $\sim 1.09\times$ with full fine-tuning. In terms of BLEU and Edit Distance scores, the improvements are on average less substantial (especially when using LoRA), with the exception of the LLaMA Edit Distance score, where the model performs ~ 1.55–$1.67\times$ better.

We conclude that fine-tuning benefits our goal of automatically generating config files starting from intents and that, in the case of having only a small dataset for fine-tuning, it is recommended to avoid LoRA for generating structured files. At the same time, it is worth noting that a larger, high-quality dataset might even lead to higher accuracy, and this is needed for consolidating K8s deployment implementations, as stated in the following section.

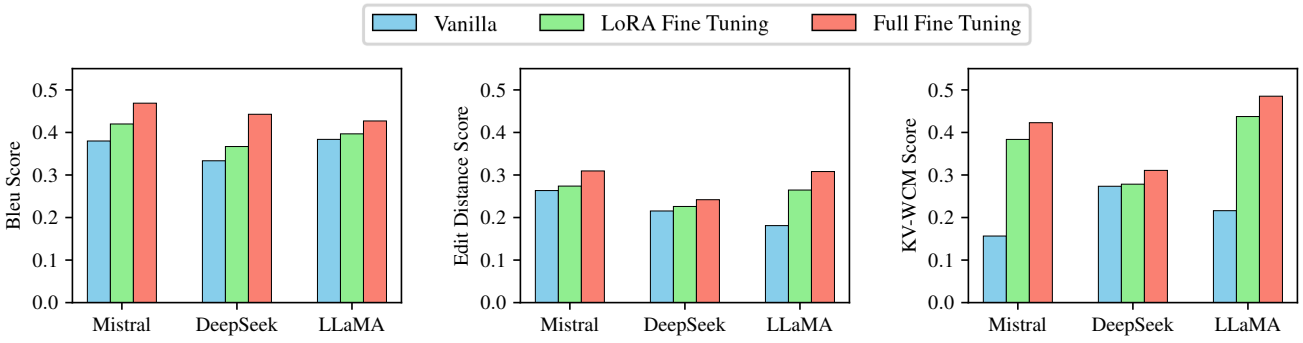
V. Current Challenges

A. Difficulty in Generating a Comprehensive K8s Dataset

While fine-tuning seems a reasonable approach, it requires some samples as input. This poses an important challenge to open-source projects, as many cloud providers, *e.g.,* Google, Alibaba, and Meta, keep this data confidential. The development of robust and generalizable models for cloud deployment automation and K8s configuration generation requires access to large-scale, high-quality datasets. Such datasets should encompass a diverse range of real-world deployment scenarios, including various cloud providers, services, and architectural patterns. Given the complexity and variability inherent in cloud-native systems, small or narrowly scoped datasets fail to capture the breadth of configurations and deployment strategies encountered in practice. A substantial dataset is critical not only for fine-tuning LLMs capable of handling the nuances of infrastructure-as-code but also for enabling meaningful benchmarking and evaluation of model performance in realistic settings.

B. Lack of Problem-Specific Metrics

While LLM-based applications related to natural language processing are gaining popularity thanks to a multitude of evaluation metrics and frameworks [27], [28], this is not the case for our task, where it is challenging to measure the accuracy of GenAI. For example, a valid YAML file can have two



(a) BLEU Score (b) Edit Distance Score (c) KV-Wildcard Score

Fig. 4: Comparison between vanilla, LoRA fine-tuned, and fully fine-tuned versions of Mistral, DeepSeek, and LLaMA, in terms of (a) BLEU Score, (b) KV-Wildcard Score, and (c) Edit Distance Score. A higher score denotes better performance for all the metrics.

specifications in a specific order or the reverse order, and both versions are perfectly interpreted by K8s. However, current metrics for measuring the distance between the ground truth and the output of the model might consider an error in this "unexpected order," despite being valid infrastructure-wise. Some YAML-specific metrics exist to move beyond the outdated notion of *exact* matching, which would require the generated YAML to be exactly the same as the reference YAML. For example, *Key-Value Exact Match* considers that order doesn't matter in YAML, thus loads both the generated and reference YAML files into dictionaries and checks if the resulting dictionaries are the same or not, so the output is either 0 (not match) or 1 (exact match). A similar approach is *Key-Value Wildcard Match (KV-WCM)* that, with the help of labeling in the reference YAML file, allows specifying what matters and what is not critical (we have used this metric in Section IV). For example, sometimes it is acceptable to start a cluster with `image: ubuntu:22.04` or `ubuntu:20.04`, so the label in reference YAML could be `image: ubuntu:22.04 # v in ['20.04', '22.04']` and either version will be considered correct. Similarly, also in KV-WCM the score ranges from 0 to 1, and the higher, the better.

However, key match and wildcard match often ignore hierarchical structure, so keys may be incorrectly flagged or missed if they appear in different levels of nesting.

C. LLMs Benchmarking

A wide range of benchmarking tools is currently available for evaluating LLMs, where modern solutions typically involve testing models with a series of tasks, often in the form of exam-style questions, and assigning scores based on task completion accuracy or quality. Most benchmarks are designed to assess a specific dimension of model capability, such as natural language understanding (*e.g.,* SuperGLUE [27]), reasoning (*e.g.,* HellaSWAG [29]), or code generation (*e.g.,* HumanEval [28] and CodeXGLUE [30]). As a result, evaluations are often highly specialized but limited in scope, targeting particular skill sets, and it is quite hard to assess the capacity of LLMs from different angles than those targeted.

Moreover, the rapid pace of LLM development frequently leads to benchmark saturation, where state-of-the-art models quickly achieve near-perfect scores, reducing the benchmarks' discriminatory power. To address these challenges, broader and more diverse evaluation frameworks have been proposed, such as BIG-bench [31], which cover a wider array of tasks and difficulty levels. However, even these efforts remain incomplete: while larger, more comprehensive benchmarks have been devised, these can nonetheless fail to adequately assess model proficiency on specific tasks, such as this K8s configuration generation, meaning complementary analysis is still required.

More specific to our task, StructEval [32] allows for evaluation of structured output generation from natural language specifications, *i.e.,* whether a generated output adheres to syntax, constraints, and specifications of structural files like JSON, YAML, or HTML. It does not, however, validate the semantic correctness; it only validates the syntax corrections of a YAML configuration. CloudEval-YAML [18], on the other hand, is a tailor-made benchmark for the generation of cloud configurations and offers, aside from similarity-based evaluation metrics, K8s unit testing procedures to test the correctness. Although it represents the most promising tool in our use case, the number of problem cases is relatively limited compared to the broad spectrum of possible K8s configuration requirements. Expanding the benchmark is certainly necessary, but it would involve significant work and additional unit testing engineering.

In summary, properly evaluating and comparing LLMs is often a delicate and nuanced process, which requires a solid understanding of both model and task type, together with careful selection of the appropriate benchmark(s) depending on the desired evaluation and specific task of the LLM.

D. Agentic AI

AI Agents, typically powered by LLMs, are designed to autonomously manage and interact with external tools and their operational environments. These agents, when approaching a problem, leverage their powerful planning and reasoning

capabilities to formulate a course of action and interact with the environment by: (i) collecting feedback (*e.g.,* usage metrics, event logs) to augment their decision-making process and (ii) utilizing available tools (*e.g.,* calls to in-house or external APIs) to complete each sub-task composing the devised strategy. Agent frameworks such as AutoGPT [33], BabyAGI [34], CrewAI [35] and LangGraph [36] facilitate prototyping and deployment of such systems. However, their effective use still demands substantial human involvement in defining objectives, workflows, and agent roles, which requires a deep understanding and knowledge of the target environment. In the context of cloud infrastructure management, AI Agents represent an increasingly valuable opportunity; several works have explored potential applications of these models with promising results (such as interfacing humans to orchestrators through natural language [37] or Root Cause Analysis [38], [39]). Nonetheless, these systems face critical limitations, such as performance inconsistency, insufficient reliability when dealing with the highly dynamic and heterogeneous scenarios in cloud management, or lack of robust safeguards to ensure safe operations [40]. Solving these issues requires identifying the common challenges [41] and key principles for both design and evaluation of agents [19], [42] for AIOps. However, current solutions towards a standardized, comprehensive framework that integrates one or more AI Agents in cloud infrastructures are still in a nascent stage, and a proper interface between humans and AI orchestrators is missing.

E. Problems with current versions of LLMs

Data manipulation. Recent research shows that LLMs struggle with data manipulation [43], [44], and even powerful LLMs such as GPT-4o, DeepSeek, make errors when dealing with numeric problems, e.g., finding the highest number or counting. An approach based on Chain-of-Code [44] proposes forcing the LLM to write and execute code to improve LM reasoning performance in arithmetic tasks. While current approaches are based on selectively simulating the interpreter by generating the expected output of certain lines of code, future AI agents can be empowered to run the whole (or portions of the) code in a prototype or digital twin before answering through a dedicated interpreter and executor.

Thinking. LLMs are now evolving to include specialized variants explicitly designed for reasoning tasks—Large Reasoning Models (LRMs) such as OpenAI's o1/o3, DeepSeek-R1, Claude 3.7 Sonnet Thinking, and Gemini Thinking, where this *"thinking"* mechanism is typically achieved via a long Chain-of-Thought (CoT) with self-reflection. Despite these claims and performance advancements, research is conducted to empirically evaluate how these models are capable of generalizable reasoning, or the inherent limitations of current reasoning approaches [45]. As recently pointed out by Apple [46], recent Large Reasoning Models (LRMs) face a complete accuracy collapse beyond certain complexities, especially for planning tasks, and the reasoning effort increases with problem complexity up to a point, but then declines despite having an adequate token budget. Also, when problems reach high complexity with longer compositional depth, LRMs and LLMs experience complete performance collapse (thus, LRMs show an advantage only for problem complexity that is moderate but not simple). Thus, AI agents controlling cloud infrastructures seem quite premature at this stage, and opportune actions crafting should be devised, e.g., a mixture of experts. Recent studies have also highlighted the "overthinking phenomenon" [47], where models produce verbose outputs even after finding the solution, creating significant inference computational overhead. As shown in [48], there has been a lot of recent success in using Reinforcement Learning (RL) to improve the reasoning ability of language models, by using a collection of questions with ground truth answers and rewarding the model for getting the correct answer.

In conclusion, the exploration of techniques for guiding or structuring how the model "thinks" presents new opportunities for enhancing model capabilities, incorporating adaptability, flexibility, critical reflection, and error correction. However, some open research questions on guiding this reasoning are still present: Can we incentivize the model to produce faithful reasoning paths during RL training while avoiding reward hacking behavior? How to define a reward appropriate during RL training or inference without human intervention? Self-correction can happen within a chain-of-thought or can be encouraged to happen explicitly during multi-turn RL [49]. In light of current limitations and issues, how can we train the model to correct itself without hallucination or regression when the ground truth is not available?

VI. Conclusion

In this paper, we presented the design of a scalable, automated LLM-aided platform that can efficiently generate Kubernetes (K8s) configuration files by fine-tuning open-source LLMs to optimize such code generation. The paper sheds light on current challenges in the cloud automation process, such as limitations in current LLMs and available datasets. In the future, we plan to instruct AI agents to break the actions into sub-tasks so as to consider more complex settings and to evaluate and compare the correctness of the generated YAML configurations through deployment testing.

Acknowledgment

This work was partially supported by the FABRIC testbed [50].

References

[1] G. Sayfan, *Mastering Kubernetes: Master the art of container management by using the power of Kubernetes*. Packt Publishing Ltd, 2018.

[2] C. Carrión, "Kubernetes scheduling: Taxonomy, ongoing issues and challenges," *ACM Computing Surveys*, vol. 55, no. 7, pp. 1–37, 2022.

[3] T. Anderson. (2021) Google admits kubernetes container tech is so complex. Accessed 28 Jun. 2025. [Online]. Available: https://www.theregister.com/2021/02/25/google_kubernetes_autopilot/

[4] A. Grattafiori, A. Dubey, A. Jauhri *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[5] A. Hurst, A. Lerer, A. P. Goucher, A. Perelman, A. Ramesh, A. Clark *et al.*, "Gpt-4o system card," *arXiv preprint arXiv:2410.21276*, 2024.

[6] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu *et al.*, "DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning," *arXiv preprint arXiv:2501.12948*, 2025.

[7] A. Dubey, C. P. Singh, and D. Nadig, "Leveraging large language models for intent-based generation of cloud-native configurations," in *2024 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, 2024, pp. 1–6.

[8] R. Birkner, D. Drachsler-Cohen, L. Vanbever, and M. Vechev, "Net2Text: Query-Guided Summarization of Network Forwarding Behaviors," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI '18)*, 2018, pp. 609–623.

[9] D. Nam, A. Macvean, V. Hellendoorn, B. Vasilescu, and B. Myers, "Using an llm to help with code understanding," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (ICSE '24)*, 2024, pp. 1–13.

[10] X.-K. Wu, M. Chen, W. Li, R. Wang, L. Lu, J. Liu *et al.*, "LLM Fine-Tuning: Concepts, Opportunities, and Challenges," *Big Data and Cognitive Computing*, vol. 9, no. 4, p. 87, 2025.

[11] A. Q. Jiang *et al.*, "Mistral 7b," *arXiv preprint arXiv:2310.06825*, 2023.

[12] D. Guo, Q. Zhu, D. Yang, Z. Xie, K. Dong, W. Zhang *et al.*, "Deepseek-coder: When the large language model meets programming–the rise of code intelligence," *arXiv preprint arXiv:2401.14196*, 2024.

[13] E. J. Hu, Y. Shen *et al.*, "Lora: Low-rank adaptation of large language models," *ICLR*, vol. 1, no. 2, p. 3, 2022.

[14] M. Howard, "Terraform–automating infrastructure as a service," *arXiv preprint arXiv:2205.10676*, 2022.

[15] A. Leivadeas and M. Falkner, "A Survey on Intent-Based Networking," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 1, pp. 625–655, 2023.

[16] Cisco Systems. (2023) Intent-based networking (ibn). Accessed: 2025-06-17. [Online]. Available: https://www.cisco.com/c/en/us/solutions/intent-based-networking.html

[17] A. Angi, A. Sacco, and G. Marchetto, "LLNet: An Intent-Driven Approach to Instructing Softwarized Network Devices Using a Small Language Model," *IEEE Transactions on Network and Service Management*, 2025.

[18] Y. Xu *et al.*, "CloudEval-YAML: A Practical Benchmark for Cloud Configuration Generation," in *Proceedings of Machine Learning and Systems (MLSys '24)*, vol. 6, 2024, pp. 173–195.

[19] M. Shetty, Y. Chen, G. Somashekar, M. Ma, Y. Simmhan *et al.*, "Building AI Agents for Autonomous Clouds: Challenges and Design Principles," in *Proceedings of the 2024 ACM Symposium on Cloud Computing (SoCC '24)*, 2024, pp. 99–110.

[20] Z. M. Wang, Z. Peng, H. Que, J. Liu, W. Zhou, Y. Wu *et al.*, "Rolellm: Benchmarking, eliciting, and enhancing role-playing abilities of large language models," *arXiv preprint arXiv:2310.00746*, 2023.

[21] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi *et al.*, "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," *Advances in neural information processing systems (NeurIPS '22)*, vol. 35, pp. 24 824–24 837, 2022.

[22] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal *et al.*, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," *Advances in neural information processing systems (NeurIPS '20)*, vol. 33, pp. 9459–9474, 2020.

[23] T. Wolf, L. Debut, V. Sanh, J. Chaumond *et al.*, "Transformers: State-of-the-Art Natural Language Processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. ACL, 2020, pp. 38–45.

[24] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "BLEU: a Method for Automatic Evaluation of Machine Translation," in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics (ACL)*, 2002, pp. 311–318.

[25] J. W. Ratcliff and D. E. Metzener, "Pattern matching: The gestalt approach," *Dr. Dobb's Journal*, pp. 46–?, Jul. 1988.

[26] I. Predoaia *et al.*, "Towards processing yaml documents with model management languages," in *ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*. ACM, 2024, pp. 970–979.

[27] A. Wang, Y. Pruksachatkun, N. Nangia, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman, "Superglue: A stickier benchmark for general-purpose language understanding systems," *Advances in Neural Information Processing Systems 32 (NeurIPS '19)*, vol. 32, 2019.

[28] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. D. O. Pinto, J. Kaplan *et al.*, "Evaluating Large Language Models Trained on Code," *arXiv preprint arXiv:2107.03374*, 2021.

[29] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi, "Hellaswag: Can a machine really finish your sentence?" in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

[30] S. Lu, D. Guo, S. Ren, J. Huang, A. Svyatkovskiy *et al.*, "CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation," in *35th Conference on Neural Information Processing Systems (NeurIPS 2021) Track on Datasets and Benchmarks*, 2021.

[31] A. Srivastava, A. Rastogi, A. Rao, A. A. Shoeb, A. Abid, A. Fisch *et al.*, "Beyond the imitation game: Quantifying and extrapolating the capabilities of language models," *Transactions on machine learning research*, 2023.

[32] J. Yang, D. Jiang, L. He, S. Siu, Y. Zhang, D. Liao *et al.*, "StructEval: Benchmarking LLMs' Capabilities to Generate Structural Outputs," https://github.com/TIGER-AI-Lab/StructEval, 2025, accessed: Jun. 28, 2025.

[33] SignificantGravitas, "AutoGPT: Build, Deploy, and Run AI Agents," [Online]. Available: https://github.com/Significant-Gravitas/AutoGPT, [Accessed: May 31, 2025].

[34] Y. Nakajima, "BabyAGI: An Autonomous AI Agent Framework," [Online]. Available: https://github.com/yoheinakajima/babyagi, 2023, [Accessed: Jun. 28, 2025].

[35] T. Renelle *et al.*, "CrewAI: A Multi-Agent Framework for Autonomous AI Teams," [Online]. Available: https://github.com/tyleryen/crew-ai, 2023, [Accessed: Jun. 28, 2025].

[36] LangChain Inc, "LangGraph: Build resilient language agents as graphs," [Online]. Available: https://github.com/langchain-ai/langgraph, 2025, gitHub repository, MIT License. [Accessed: Jun. 28, 2025].

[37] A. Vitui and T.-H. Chen, "Empowering aiops: Leveraging large language models for it operations management," *arXiv preprint arXiv:2501.12461*, 2025.

[38] T. Ahmed *et al.*, "Recommending root-cause and mitigation steps for cloud incidents using large language models," in *IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 1737–1749.

[39] Y. Zhang, M. Kale, A. Subramanian, V. Ganti, A. Rao, Y. Lin *et al.*, "Automated root causing of cloud incidents using in-context learning with gpt-4," *arXiv preprint arXiv:2401.13810*, 2024.

[40] Z. Yang, A. Bhatnagar, Y. Qiu, T. Miao, P. T. J. Kon, Y. Xiao, Y. Huang, M. Casado, and A. Chen, "Cloud infrastructure management in the age of ai agents," *arXiv preprint arXiv:2506.12270*, 2025.

[41] Q. Cheng, D. Sahoo, A. Saha, W. Yang, C. Liu, G. Woo *et al.*, "Ai for it operations (aiops) on cloud platforms: Reviews, opportunities and challenges," *arXiv preprint arXiv:2304.04661*, 2023.

[42] Y. Chen, M. Shetty, G. Somashekar, M. Ma, Y. Simmhan, J. Mace *et al.*, "AIOpsLab: A Holistic Framework to Evaluate AI Agents for Enabling Autonomous Clouds," in *Eighth Conference on Machine Learning and Systems (MLSys '25)*, 2025.

[43] A. Maatouk *et al.*, "Large Language Models for Telecom: Forthcoming Impact on the Industry," *IEEE Communications Magazine*, vol. 63, no. 1, pp. 62–68, 2025.

[44] C. Li, J. Liang, A. Zeng, X. Chen, K. Hausman, D. Sadigh *et al.*, "Chain of Code: Reasoning with a Language Model-Augmented Code Emulator," in *International Conference on Machine Learning (ICML '24)*. PMLR, 2024, pp. 28 259–28 277.

[45] S. I. Mirzadeh, K. Alizadeh, H. Shahrokhi, O. Tuzel, S. Bengio, and M. Farajtabar, "GSM-Symbolic: Understanding the Limitations of Mathematical Reasoning in Large Language Models," in *The Thirteenth International Conference on Learning Representations (ICLR 25)*, 2025.

[46] P. Shojaee, I. Mirzadeh, K. Alizadeh, M. Horton, S. Bengio, and M. Farajtabar, "The illusion of thinking: Understanding the strengths and limitations of reasoning models via the lens of problem complexity," *arXiv preprint arXiv:2506.06941*, 2025.

[47] Y. Sui, Y.-N. Chuang, G. Wang, J. Zhang, T. Zhang, J. Yuan *et al.*, "Stop overthinking: A survey on efficient reasoning for large language models," *arXiv preprint arXiv:2503.16419*, 2025.

[48] L. Weng, "Why We Think," [Online]. Available: https://lilianweng.github.io/posts/2025-05-01-thinking/, May 2025, [Accessed: Jun. 28, 2025].

[49] A. Kumar *et al.*, "Training language models to self-correct via reinforcement learning," *arXiv preprint arXiv:2409.12917*, 2024.

[50] I. Baldin, A. Nikolich, J. Griffioen, I. I. S. Monga, K.-C. Wang, T. Lehman, and P. Ruth, "Fabric: A national-scale programmable experimental network infrastructure," *IEEE Internet Computing*, vol. 23, no. 6, pp. 38–47, 2019.