Cloud Infrastructure Management in the Age of AI Agents

Zhenning Yang[1*], Archit Bhatnagar[1*], Yiming Qiu[1,2*], Tongyuan Miao[1]
Patrick Tser Jern Kon[1], Yunming Xiao[1], Yibo Huang[1], Martin Casado[3], Ang Chen[1]
[1]*University of Michigan* [2]*UC Berkeley* [3]*Andreessen Horowitz*

ABSTRACT

Cloud infrastructure is the cornerstone of the modern IT industry. However, managing this infrastructure effectively requires considerable manual effort from the DevOps engineering team. We make a case for developing AI agents powered by large language models (LLMs) to automate cloud infrastructure management tasks. In a preliminary study, we investigate the potential for AI agents to use different cloud/user interfaces such as software development kits (SDK), command line interfaces (CLI), Infrastructure-as-Code (IaC) platforms, and web portals. We report takeaways on their effectiveness on different management tasks, and identify research challenges and potential solutions.

1 INTRODUCTION

Cloud computing has transformed the technology sector—today, 94% enterprises use the cloud [1]. However, managing the cloud infrastructure remains a challenging task. Cloud tenants (e.g., EA Games, Home Depot) need to customize their infrastructure for diverse workloads, but cloud providers (e.g., Amazon/Microsoft/Google) only expose a shim management layer to third-party users without revealing system internals. Management is also a continuous endeavor across the entire infrastructure lifecycle—provisioning resources, runtime monitoring, and resource updates—each with its own requirements and challenges.

To handle these tasks, tenants employ teams of DevOps (i.e., Development/Operation) engineers to supervise their cloud infrastructure. Four cloud management *modalities* have gained popularity, tuned for DevOps engineers with different experience and perferences: *(i)* cloud software development kits (SDK) libraries, used for imperative programming; *(ii)* command line interface (CLI) embedded into user terminals; *(iii)* infrastructure-as-code (IaC)[11] configurations that encode cloud resources in a declarative manner; as well as *(iv)* web portal clicks (ClickOps), the "no-code/low-code" option. Although these options are all built atop low-level RESTful cloud APIs, they present higher-level interfaces and are easier to use than RESTful API invocations.

Nevertheless, the complexity of the cloud ensures that all of these options still come with a substantial learning curve. Lifecycle management remains tedious and error-prone, often requiring manual trial-and-error steps. DevOps engineers often find themselves performing repetitive tasks such as reading cloud documentation, understanding user requirements, debugging failures, and checking policy compliance (e.g., GDPR). Management challenges further intensify as more and more organizations embrace multi-cloud deployments [12, 18] to avoid vendor lock-in. This additionally requires DevOps engineers to master significantly different cloud environments, which creates further burden.

We present a vision where human engineers are assisted by AI for cloud management. Recent advances in *AI agents* built on top of large language models (LLMs) have shown great promise in carrying out complex tasks. AI agents enhance LLMs with additional capabilities—e.g., reasoning loops [19, 36], external tool usage [28], and memory management [25]—to interpret user instructions, make complex decisions, generate execution plans, and invoke external services to interact with the environment. As AI agents become increasingly powerful, we believe that now is the right time to rethink DevOps engineering in light of this trend.

We are motivated by several questions along this direction: *Can AI agents potentially serve as cloud DevOps engineers, reducing human burden and improving productivity? What types of tooling might be the best fit for cloud AI agents? What are the research challenges that we must overcome in developing these agents, and what are some potential solutions?*

We believe that cloud management has a few characteristics that are a suitable match for an agentic design. First, compared to traditional software engineering tasks, cloud management tasks are highly structured and repetitive, which helps constrain the problem search space of agents. The outcome of cloud management is also easier to analyze and validate [20], especially when compared with existing code generation tasks that rely on large amounts of test cases to cover various program paths. Compared to popular use cases of AI agents such as web automation, cloud management already presents a range of programmable interfaces, making it an ideal fit for interaction with coding agents. Moreover, cloud providers also offer extensive documentation for their services, a treasure trove for agents to distill domain knowledge (e.g., using RAG [21] or web-based search [6]).

However, developing AI agents for cloud management also creates significant challenges. Given the criticality of cloud infrastructures, AI automation must not compromise on efficiency, reliability, and scalability of cloud operations. An agentic design needs to go beyond simply prompting an LLM model and hoping for the best; rather, various guardrails and a combination of neural and symbolic steps are needed for high assurance. Furthermore, different cloud management

*Equal contribution

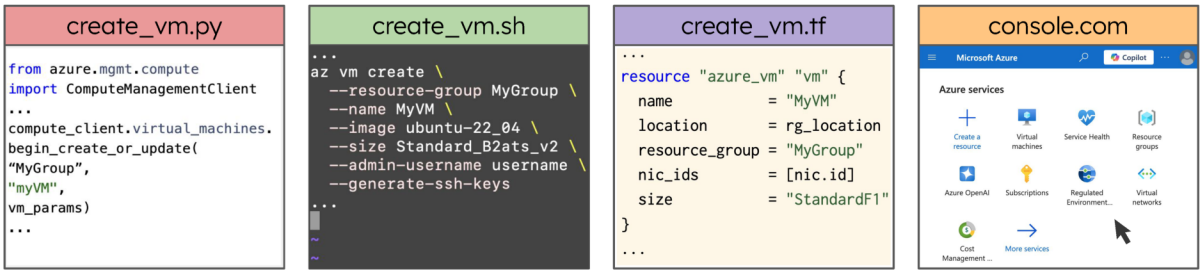

Figure 1: Four cloud/user interaction modalities, with simplified code snippets for API SDK, CLI, and IaC, alongside a screenshot of the web portal. We built several AI agents each targeting one of these modalities.

modalities may present different opportunities and hurdles for an agentic design. Agent effectiveness could further depend on the nature of the task (e.g., resource creation vs. update). This paper represents an initial foray into this direction, presenting our recent study and discussing the lessons learned. In the rest of this paper, we start by showcasing a “battle” among several preliminary AI agents operating in different modalities, and identify where they work well or fall short. We then discuss research challenges and potential roadmaps for an effective agentic design. We hope that this work would spur additional discussion in the systems community, leading to future work on the next generation of AI agent-based cloud management tools that achieve unprecedented levels of automation.

2 BATTLE OF THE AGENTS

In this section, we present the current cloud management practice by introducing the four classes of cloud/user interfaces, then examine a typical lifecycle of cloud infrastructure, and finally, conduct a preliminary case study on AI agents’ effectiveness on management tasks. These lessons help us build intuition on the design gaps and research directions.

2.1 The warriors: management modalities

The four most popular modalities of cloud management, shown in Figure 1, are built on a common layer—RESTful APIs exposed by cloud providers. The RESTful APIs manipulate cloud resources as data objects via HTML methods—e.g., `get` APIs retrieve cloud data (e.g. resources, cost, logs) in JSON format, while `post`, `put`, and `delete` create, update, and destroy resources, respectively. Cloud-level RESTful APIs effectively form the “system call” layer of cloud/user interaction—if the cloud allows users to perform any task, that task must eventually map to some API invocations. Underneath this API layer, cloud providers implement their services in a vendor-specific manner, with few details exposed to the tenants. However, leveraging RESTful APIs requires users to directly handlr raw HTTP requests—e.g., processing request headers, initiating authentication, parsing responses, and dealing with low-level concerns like rate-limiting, retries and asynchronous polling. Even the deployment of a single cloud resource could involve a complex sequence of API invocations and auxiliary scripts. Therefore, DevOps engineers typically interface with higher-level management modalities, described below.

Software development kits (SDK). All major cloud providers offer SDKs for popular programming languages (e.g., Python, Java, Go), which wrap the raw RESTful APIs into easier-to-use libraries for DevOps engineers. For instance, as shown in Figure 2(a), with the Azure Python SDK, a single library call can create and configure virtual machines (VMs), hiding multiple RESTful operations behind this method. SDK programs are an imperative approach to cloud management, allowing developers to use their familiar programming languages to manipulate cloud resources.

Command line interfaces (CLI). Another modality is to embed common cloud commands into an intuitive, shell-like CLI interface. In Figure 2(b), the Azure CLI provides the ‘az create’ family for creating resources (e.g., VMs, subnets, network interface cards); likewise, the ‘az list’ command retrieves existing cloud states, and ‘az monitor’ manages log events and performance metrics. CLI is well suited for interactive, one-off tasks such as small-scale “canned” tests and queries, while full-fledged cloud management applications requires more programmatic control and object-oriented integrations, e.g., as provided in the SDK.

Infrastructure-as-Code (IaC). IaC tools provide a higher level of abstraction, shielding even more complexities from the developer via a state-centric design. Popular IaC frameworks include Terraform [10], OpenTofu [8], Pulumi [9], Crossplane [4], and CloudFormation [3], with Terraform leading the market. An IaC program (e.g., a snippt shown in Figure 2(c)) declares the intended cloud state, such as an infrastructure with a certain number of Azure virtual machines, connected with network interface cards and guarded by firewalls. IaC tools compile the program into sequences of RESTful API invocations to automatically move the cloud infrastructure from its current state to the intended state. This reduces the cognitive burden not only because users no longer need to reason about the step-by-step execution for a given task (e.g., which APIs to use in order to create or

update a resource), but also because most IaC tools are inherently cloud-agnostic—that is, its program syntax remains the same across different cloud providers. An IaC developer can therefore manage their multi-cloud deployments (e.g., in AWS and Azure) using the same framework.

ClickOps. Last but not least, as shown in Figure 2(d), cloud providers also expose orchestration capabilities via web portals, which are graphic user interfaces (GUI) that visualize cloud configurations, accepting UI clicks to interact with the cloud. This option is commonly known as "ClickOps," as the DevOps engineers have to click through the web portals to manage their infrastructure. As an advantage, GUIs do not require any programmatic interactions, so even operators with no coding experience will find them accessible. However, ClickOps may not be as efficient since it requires multiple clicks—often in a precise order—to accomplish a task. This is restricted by the speed at which humans can think and interact with their portals. With multi-cloud deployments, each cloud structures its portals/menus differently, which could further increase the cognitive burden.

2.2 The battleground: management tasks

We describe typical lifecycle stages of cloud infrastructure management, with three representative categories that we will use in Section 2.3 to drive our case study.

Cloud infrastructure comes into being after a resource *provisioning* stage, where DevOps engineers instantiate a desired infrastructure by creating its constituent resources (e.g., virtual private clusters (VPCs), subnets, routing tables, virtual machines, and gateways) and interconnecting them to function as a whole. This requires not only configuring the attributes of each resource individually (e.g., choosing a memory-optimized VM with spot priority), but also their dependencies (e.g., a NIC depends on its VM).

Cloud infrastructure is long-lived, so DevOps engineers need to perform periodic *updates* to modify the infrastructure for changing requirements (e.g., adding resources, dynamic scaling). Whereas some resources can be modified in a live manner (e.g., attaching an additional disk to the VM), other modifications will tear down and recreate the resources (e.g., changing the VM type from 'standard' to 'spot'). Modifying one resource also needs to account for its dependencies on other resources, which may require propagating the changes to a larger update radius. Application-level policies, such as fault tolerance or performance objectives, are also important.

At any time, cloud infrastructure needs runtime *monitoring* to track the fleet status (e.g., resource utilization, system performance) and ensure its health (e.g., by collecting telemetry data, diagnosing problems, and rolling out fixes). Quite often, the raw telemetry data needs to be converted in to easier-to-digest formats (e.g., visualization) to further assist DevOps engineers to quickly locate the relevant trends.

	Provisioning		Updates		Monitoring	
Agents	SR	#steps	SR	#steps	SR	#steps
SDK	0.67	4.5	**0.67**	**2.0**	0.80	1.25
CLI	**1.0**	**1.6**	0.67	3.0	0.80	1.0
IaC	1.0	2.0	0.33	5.0	0.40	2.5
Web	0.33	46.0	0.67	20.0	**1.0**	**2.75**

Table 1: Agent performance (success rate (SR) and the average number of steps) on VM management tasks. We highlight in bold the best-performing agent—first ranked by SR, and then the number of steps.

2.3 Tales from the battlefield

We present a case study focused on management operations on a core cloud resource: virtual machines (VMs). We developed and adapted four preliminary AI agent prototypes for each of the four management modalities.

- **SDK agent:** This agent relies on Azure's Python SDK to generate code, leveraging LLMs' strengths in code generation, especially Python programs [37].
- **CLI agent:** It writes Shell scripts that interact with the cloud through Azure's "cloud shell," which provides canned CLI commands.
- **IaC agent**: It uses Terraform [10], one of the most popular IaC tools, and generates Terraform programs to perform management tasks.
- **ClickOps agent:** It navigates the web UI, leveraging screenshots and accessibility tree or AXTree [45] to perform tasks via the cloud provider's console.

The SDK, CLI, and IaC agents use Azure Copilot [7] as the model, which is based upon GPT-4 but specifically tuned for the Azure cloud. The ClickOps agent implementation was adopted from WorkArena [17], and is powered by GPT-4o [24], which is known as an effective model for web-based agents. Table 1 summarizes our findings, as detailed below.

Battle #1: Provisioning. We performed three different tasks with the agents—creating a single VM; creating three VMs under the same network; and connecting the three VMs to a load balancer. For each task we perform eight trials with different prompts to account for stochasticity in agent behavior. For each successful trial, we measure the number of steps that each agent takes on average to complete the task—each step is a single action taken by the agent, such as generating code or executing a browser click. A trial is considered unsuccessful if it takes over 100 steps for the task.

We found the CLI agent to be the most efficient, completing the tasks in 1.6 steps on average, with a high success rate by generating the required command in a single step in most cases. The SDK agent took 4.5 steps on average to generate and execute the Python program at about 67% success rate. The IaC agent took two steps on average to generate the correct Terraform configuration—one step to generate the IaC program, and another to deploy the resources to the cloud. In contrast, the ClickOps agent needed around 30× more

steps than the CLI agent, as each click in the cloud console triggered updates to the web, which then prompted another agent interaction for the next step; overall, the ClickOps approach is the slowest and the most costly for this task.

With more complex provisioning tasks (e.g., creating three VMs under the same virtual network), the ClickOps agent failed to generate the correct sequence of steps. After repeated failures, it eventually reached the maximum step limit we enforced, and terminated without completing the task. This is because provisioning more resources requires more web-based interactions, which amplify the probability of errors. The coding-based agents, however, can programmatically generate code largelt in the same way, regardless of how many resources are contained in the program.

Observation #1: Although AI agents have stochastic behaviors, our preliminary experiment shows that they are rather reliable with smaller tasks. However, errors increase for management tasks that require multiple steps. The ClickOps agent is particularly slow and error-prone for resource creation tasks.

Battle #2: Updates. The agents then attempted three update tasks: two in-place (i.e., live) updates: attaching an additional disk to an existing VM; enabling boot diagnostics for the VM; and a third update that modifies the VM type from 'standard' to 'spot,' which requires tearing down the existing VM and creating a new instance.

We found that the ClickOps agent benefits from the console's natural presentation of existing VM configurations, which helps reduce errors, achieving a much higher success rate (67%) compared to provisioning tasks where they did not "see" a preexisting cloud state. However, it required many (avg=20) clicks to accomplish the tasks. IaC agents, on the other hand, have a state-centric design and always keep a copy of the previous cloud state; in principle, this would help with resource updates, but due to the context window constraints this agent could not pass its entire state to the model. As a result, the IaC agent only achieved 33% success rate; we hypothesize that its effectiveness would increase with longer context windows. The CLI and SDK agents needed additional commands to retrieve state information, and these extra steps increase their error rates and operational overhead compared to resource creation tasks.

As another finding, in-place/live updates (e.g., attaching disks, enabling boot diagnostics) had higher success rates, while updates that required resource recreation tend to trigger failures due to the extra complexity. For instance, modifying an Azure VM from a 'standard' type to a 'spot' instance requires destroying the current resource and creating a new one. The IaC agent outperformed others because such an update only requires modifying a single VM attribute in the IaC program—the teardown and recreation steps are automatically handled by the Terraform framework. However, the SDK, CLI, and ClickOps agents need to navigate each step (i.e., saving the current VM image, destroying the VM, and then creating another using the saved image), and encountered higher failure rates. Concretely, they failed when attempting to save the existing VM's image.

Observation #2: AI agents' ability to access and reason about cloud state is important for resource update tasks, which modify an existing state. This is strongly influenced by the interaction modalities, which directly impact agent effectiveness.

Battle #3: Monitoring. We compared the agents on five different monitoring tasks—obtaining the VM status (e.g., running/stopped), obtaining its public IP, and fetching the state of all attached disks (e.g., size and type). We found that retrieving resource information via different management modalities differs in complexity. For example, obtaining disk information required one step for both SDK and CLI agents, two steps for ClickOps, and as many as eight steps for IaC. On average, the CLI and SDK agents performed similarly, achieving around 80% success rates within one step on average. However, the IaC agent was poorly suited for monitoring tasks, with only 40% success rate,. We found that this agent encountered numerous bugs in the monitoring tasks, such as hallucination that generated non-IaC languages or invocation of deprecated methods. Complex monitoring tasks, such as retrieving a resource dependency graph, are naturally suited to the web interface (2 steps), which provides visual representations not present in SDK/CLI modalities; the ClickOps agent performed almost perfectly on these tasks. As another interesting finding, there exists monitoring services, such as real-time service health checks, that are only available in the web portal; for instance, the Azure Service Health dashboard provides insights into cloud region outages, maintenance, and historical incidents, but other modalities have no such support.

Observation#3: Monitoring tasks require the agents to obtain real-time resource/state information. Modalities vary in how well they expose the current state; IaC's state-centric design only captures the infrastructure composition, but cannot easily retrieve runtime telemetry; thereby struggling the most for monitoring tasks.

Summary: Even though these AI agents are preliminary prototypes, they demonstrated promising results, especially on simpler tasks. That said, agentic failures were still quite common, especially with complex provisioning/update tasks and monitoring tasks. We observed a variety of reasons for failure across different tasks. For coding agents (i.e., SDK/-CLI/IaC), many failures occur due to incorrect resource attributes or an invalid sequence of commands. Upon repeated trials, the agents are sometimes capable of retrieving the error logs to correct these mistakes. For the no-code/low-code

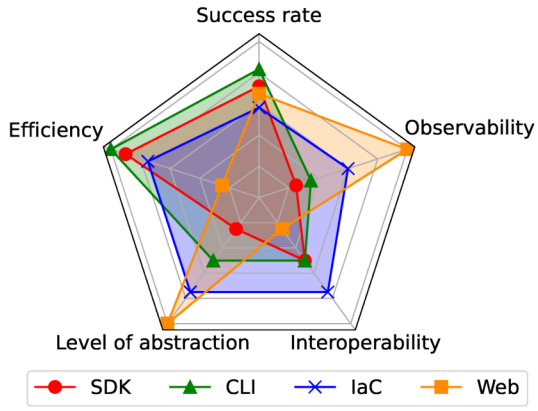

Figure 2: The radar chart summarizes our initial case study of AI agents interacting with the cloud across different modalities. Level of abstraction denotes the amount of details exposed; interoperability denotes the support for cross-cloud operations; observability refers to ease of tracing task execution; success rate denotes the accuracy of task completion; and efficiency denotes the number of steps needed by AI agents to complete the given tasks.

ClickOps agent, misclicks and inability to locate the right click sequence often prevent them from making progress. However, GUI-based automation has proven quite effective for monitoring tasks—the click sequences tend to be simpler and cloud provider portals specifically optimize for monitoring and visualization. Figure 2 summarizes our comparison.

Observation #4: AI agents' ability to handle errors varies by modality. Programmable interfaces like SDK, CLI, and IaC offer precise feedback (e.g., return codes and error logs), while web interfaces often report UI-level errors that are harder to interpret. Recall that for each task we have tried different prompts. We found that carefully-crafted prompt, which include both task-specific hints to support cloud reasoning and modality-specific instructions to navigate environments, help improve task completion.

3 SOLUTION SKETCH

The cloud is a complex and dynamic environment with ever-increasing services and features. New players (e.g., specialized AI cloud) are coming into the market, which will increase management difficulty. We outline a solution sketch for future cloud management agents and identify key research directions and lay out a roadmap for cloud agent design. Figure 3 shows our envisioned design.

3.1 Agent architecture

Since different modalities present different tradeoffs, we envision a system architecture where the agent utilizes multiple

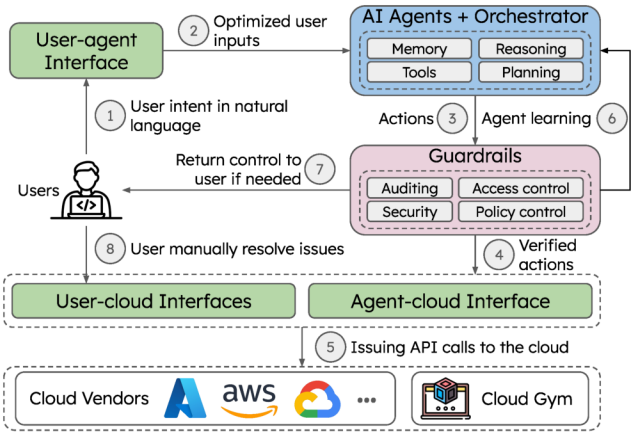

Figure 3: Envisioned agentic system architecture and workflow for cloud infrastructure management.

modalities for different management tasks. Our proposed architecture consists of three components.

User-agent interface. Cloud operations are safety-critical, and mismatches between user intentions and agent actions can cause severe infrastructure damage. For instance, Azure requires destroying and recreating VMs to change their priority from standard to spot, but users might incorrectly assume this update can be done in a live manner. To address these problems, we propose to design a user-agent interface that consumes user prompts, and then outputs advisory messages to clarify user intention and alerts the user of potential side effects of the action. The user can iteratively provide guidance, e.g., using RLHF (reinforcement learning with human feedback), to better align agent actions with user intention.

Agent-cloud interface. Our case studies highlighted that CLI generally excels in efficiency, IaC in large-scale updates, and web interfaces in monitoring. While mastering all modalities creates an undue burden for human operators, an AI agent should be able to easily select and combine interfaces to optimize task execution. If we combine multiple modalities, the agent-cloud interface must reconcile actions taken via each modality. For instance, if not careful, this could lead to resource drifts (e.g., ClickOps modifying IaC-managed resources, without modifying IaC tools' local state) or race conditions (e.g., CLI and ClickOps updating the same resource simultaneously) [30]. We propose that this interface should present a unified cloud state, and expose synchronization primitives for cross-modality interactions (e.g., via locking and transactions), for consistent management actions.

Multi-agent orchestration. Cloud management tasks differ in their complexity, and we believe a *complexity measure* is needed to quantify task difficulty—e.g., using the number of resources, interconnections, single- vs. multi-cloud, the size of the existing cloud data, as some basic metrics. We can then develop specialized "experts" backed by different models for each type of tasks. Management tasks will be routed to the appropriate agents by an orchestrator, depending on the

complexity of the task, the expected timeline for executing the task, and the monetary budget.

3.2 Agent workflow

We propose to divide a cloud agent workflow into two distinct phases: an exploration phase, focused on navigating various execution strategies, and an exploitation phase, dedicated to completing cloud management tasks.

Separating exploration from exploitation. Cloud tasks can involve sequences of slow and expensive provisioning operations, making trial-and-error approaches inefficient both in terms of time overhead and economic cost. The current design philosophy of AI agents, which relies on multi-turn retries to make progress, naturally comes with the risk of further deteriorating the efficiency of cloud system management. We borrow from the longstanding practice of cloud system development, which advocates for the separation of testing and production phases. Concretely, when assigned the task of updating all VPC gateways within the current subscription, AI agents should start with a bold exploration phase that tests different execution strategies within a controlled sandbox environment (e.g., create a new VPC gateway in a test subscription and try out update plans). To further improve interpretability and reliability, once the exploration completes, the agents will articulate its knowledge in symbolic rules. This will effectively form a "metaprogram" that the agent intends to execute on the cloud infrastructure. This symbolic program can be further subjected to type checking, program verification, or testing, to achieve higher assurance.

Optimizing agent exploration. Data scarcity is a major obstacle in advancing AI-driven cloud operations. Simulation environments, or "gyms" [13], offer low-risk arenas for agent exploration. Existing gyms typically focus on gameplay [26, 33] or synthetic tasks [14, 17, 45], where errors have minimal consequences. These self-hosted environments use controlled benchmarks to safeguard the exploration of AI agents. In contrast, real-world cloud experiments are costly and risky, making extensive trial-and-error or reinforcement learning (RL) approaches impractical. We propose to build cloud gyms and benchmarks that replicate the complexity of real cloud setups (e.g., resources, functionalities, and billing models) in a virtual, sandboxed environment, for safe and efficient agent exploration.

Optimizing agent exploitation Cloud environments are dynamic, with changing conditions such as load spikes, resource outages, or pricing fluctuations. The exploitation phase must adapt in real time to these shifts, enabling efficient resource allocation and cost management while maintaining system performance and reliability under evolving circumstances. We propose to leverage workflow learning [33, 35, 44], which "caches" the knowledge gained from previous agent executions to improve the efficiency and agility of the current exploitation phase. Once an agent successfully performs and verifies a sequence of actions, it can extract and save the workflow in agent memory for future reuse. Memorizing validated workflows enables the agent to perform similar tasks more efficiently, alleviating the cold start problem of the exploitation phase. Combined with reasoning [19, 36] and planning [41] techniques, the agent can adapt these workflows to new contexts and execute cloud tasks more effectively.

3.3 Agent guardrails

We propose to investigate agents with different levels of autonomy. While fully autonomous agents [22] will remain a challenging goal, "co-pilot" agents that assist human operators, or semi-autonomous agents that perform multi-step reasoning, are already within reach. Regardless of the level of autonomy, agents will need strong guardrails and effective mechanisms for fault tolerance.

Constraining agents with guardrails. Agent actions must be checked and verified against up-to-date policies to prevent unintended or harmful operations. Regulatory policies (e.g., privacy requirements such as GDPR or data sovereignty regulations) are an important goal in cloud management. Furthermore, cloud providers often have their own requirements [29] and so do tenants (e.g., security best practices). AI agents must ensure policy compliance when managing the cloud resources. Whereas today many such policies are stated in natural language, we envision encoding these policies in formal specifications and checking the metaprogram against these specifications to ensure compliance. Furthermore, we propose equipping AI agents with different access control privileges, constraining their actions. We will add audit trails to agentic operations, such as detailed logs and reports, so that changes can be attributed to certain operations and misbehaviors can be detected precisely.

Fault tolerance. The stochastic nature of AI agents ensures that failures will inevitably occur in some scenarios. For example, our ClickOps agent often got stuck due to incorrect steps taken earlier, entering a loop of repeated failures. WE need to develop better fault-tolerance mechanisms so that AI agents can handle and recover from failures effectively [30]. This includes implementing mechanisms for retrying operations, rolling back unsuccessful changes [27], and applying error correction strategies. The audit trails mentioned above will also provide a starting basis for rollback and recovery mechanisms, allowing agents to diagnose what went wrong so that they can revert the system to a known-good state. Such safeguards will not only contain the blast radius of failures but also enable self-healing mechanisms that allow agents to recognize and recover from their mistakes.

Human-in-the-Loop supervision. Ensuring the safe deployment of cloud AI agents requires a careful balance between autonomy and oversight. Frameworks should incorporate safeguards that allow agents to operate independently for routine tasks while enabling escalation mechanisms for human intervention in high-stakes scenarios. Agents should

also detect when they are stuck or unable to resolve errors, such as repeated failures or inconsistent states, and return control to the user. We propose to encode runtime checks on agentic behavior, and trigger alarms when certain thresholds have been exceeded, so as to minimize risks by combining the efficiency of autonomous decision-making with the reliability of human judgment for critical operations. In other words, the traditional human-cloud interfaces should remain available as a fallback solution, ensuring that users can always intervene and regain full control when necessary.

4 RELATED WORK

Code generation. Existing LLM-based code generation tools [38, 43] offer useful starting points for cloud management tasks. However, cloud management often uses low-resource languages (e.g., cloud SDK/CLI/IaC), whereas today's LLMs excel at more popular languages (e.g., Python/C). While initial progress has been made in generating IaC programs using AI models [20], effective cloud operations also demand continuous monitoring and dynamic updates, beyond resource creation. Likewise, although extensive studies exist in web automation agents [23, 40] (e.g., for online shopping). Cloud operations, by comparison, involve multi-layered dependencies, higher failure impact, and long-horizon objectives like cost optimization, security enforcement, and compliance assurance. As a result, existing code generation techniques fall short in addressing the complexity, safety, and adaptivity required for robust cloud automation.

AI agents. Building on top of large models, AI agents are systems that can perform iterative decision-making while interacting with external environments through tool use, such as APIs, code execution, or command-line interfaces [42]. Recent works have explored enhancing agent capabilities through explicit planning [41] and learning from feedback [32, 35] to improve performance and adaptivity. However, current applications remain largely limited to simplified domains such as web-based shopping [40] or sandboxed game environments [33], where the operational complexity is lower and consequences of failure are not as large. Other work investigates the security vulnerabilities of AI agents, particularly through adversarial attacks [16, 34, 39]. AI agents for the cloud also need better understanding of security risks, so that we can develop defense mechanisms to ensure agent safety and robustness.

AIOps. LLMs have been applied to log analysis and incident diagnosis for cloud operations [14, 15, 31]. Unlike these specialized tools, which primarily focus on data analytics, our vision is to enable autonomous actions, e.g., tool use, reasoning, that will help assist with a wider range of infrastructure management tasks. Cloud platforms are actively integrating LLM-driven chatbots (e.g. Azure Copilot [7], GCP Gemini [5], and AWS Amazon Q [2]) to enhance ClickOps workflows. These tools often use on retrieval-augmented generation to summarize cloud documentation, providing guidance to users but still require them to manually interpret and implement instructions.

5 SUMMARY

Cloud infrastructure management is a critical but tedious task. In this paper, we have made a case for developing AI agents to assist cloud DevOps engineers in these tasks. Our preliminary study with several agents performing a variety of tasks shows that AI agents are a promising candidate for automation, although much more is needed to make them safe, efficient, and reliable. We propose a technical roadmap for addressing various research challenges that need to be addressed for realizing this vision.

REFERENCES

[1] 26 cloud computing statistics, facts & trends for 2023. https://www.cloudwards.net/cloud-computing-statistics/#Sources.

[2] Amazon Q in AWS services. https://aws.amazon.com/q/.

[3] AWS CloudFormation. https://aws.amazon.com/cloudformation/.

[4] Crossplane: Cloud-native framework for platform engineering. https://www.crossplane.io/.

[5] Gemini for Google Cloud. https://cloud.google.com/products/gemini.

[6] LangChain build context-aware reasoning applications. https://www.langchain.com/.

[7] Microsoft copilot in Azure. https://azure.microsoft.com/en-us/products/copilot.

[8] OpenTofu: The open source infrastructure as code tool. https://opentofu.org/.

[9] Pulumi: Infrastructure as code in any programming language. https://www.pulumi.com/.

[10] Terraform by Hashicorp. https://www.terraform.io/.

[11] What is infrastructure as code (IaC). https://learn.microsoft.com/en-us/devops/deliver/what-is-infrastructure-as-code.

[12] Juncal Alonso, Leire Orue-Echevarria, Valentina Casola, Ana Isabel Torre, Maider Huarte, Eneko Osaba, and Jesus L. Lobo. Understanding the challenges and novel architectural models of multi-cloud native applications – a systematic literature review. *J. Cloud Comput.*, 2023.

[13] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[14] Yinfang Chen, Manish Shetty, Gagan Somashekar, Minghua Ma, Yogesh Simmhan, Jonathan Mace, Chetan Bansal, Rujia Wang, and Saravan Rajmohan. Aiopslab: A holistic framework to evaluate ai agents for enabling autonomous clouds, 2024.

[15] Yinfang Chen, Huaibing Xie, Minghua Ma, Yu Kang, Xin Gao, Liu Shi, Yunjie Cao, Xuedong Gao, Hao Fan, Ming Wen, Jun Zeng, Supriyo Ghosh, Xuchao Zhang, Chaoyun Zhang, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Tianyin Xu. Automatic root cause analysis via large language models for cloud incidents. In *Proceedings of the 19th European Conference on Computer Systems (EuroSys'24)*, 2024.

[16] Zhaorun Chen, Zhen Xiang, Chaowei Xiao, Dawn Song, and Bo Li. Agentpoison: Red-teaming llm agents via poisoning memory or knowledge bases, 2024.

[17] Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. Workarena: How capable are web agents at solving common knowledge work tasks?, 2024.

[18] Jiangshui Hong, Thomas Dreibholz, Joseph Adam Schenkel, and Jiaxi Alessia Hu. An overview of multi-cloud computing. In *Web,*

Artificial Intelligence and Network Applications. Springer International Publishing, 2019.

[19] Jie Huang and Kevin Chen-Chuan Chang. Towards reasoning in large language models: A survey, 2023.

[20] Patrick Tser Jern Kon, Jiachen Liu, Yiming Qiu, Weijun Fan, Ting He, Lei Lin, Haoran Zhang, Owen M. Park, George Sajan Elengikal, Yuxin Kang, Ang Chen, Mosharaf Chowdhury, Myungjin Lee, and Xinyu Wang. Iac-eval: A code generation benchmark for cloud infrastructure-as-code programs. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024.

[21] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2021.

[22] Margaret Mitchell, Avijit Ghosh, Alexandra Sasha Luccioni, and Giada Pistilli. Fully autonomous AI agents should not be developed, 2025.

[23] Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. Webgpt: Browser-assisted question-answering with human feedback, 2022.

[24] OpenAI. Gpt-4 technical report, 2024.

[25] Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Memgpt: Towards llms as operating systems, 2024.

[26] Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior, 2023.

[27] Shishir G. Patil, Tianjun Zhang, Vivian Fang, Noppapon C., Roy Huang, Aaron Hao, Martin Casado, Joseph E. Gonzalez, Raluca Ada Popa, and Ion Stoica. Goex: Perspectives and designs towards a runtime for autonomous llm applications, 2024.

[28] Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis, 2023.

[29] Yiming Qiu, Patrick Tser Jern Kon, Ryan Beckett, and Ang Chen. Unearthing semantic checks for cloud infrastructure-as-code programs. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, 2024.

[30] Yiming Qiu, Patrick Tser Jern Kon, Jiarong Xing, Yibo Huang, Hongyi Liu, Xinyu Wang, Peng Huang, Mosharaf Chowdhury, and Ang Chen. Simplifying cloud management with cloudless computing. In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*, 2023.

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

[32] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023.

[33] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models, 2023.

[34] Yifei Wang, Dizhan Xue, Shengjie Zhang, and Shengsheng Qian. Badagent: Inserting and activating backdoor attacks in llm agents, 2024.

[35] Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory, 2024.

[36] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.

[37] Tengfei Xue, Xuefeng Li, Tahir Azim, Roman Smirnov, Jianhui Yu, Arash Sadrieh, and Babak Pahlavan. Multi-programming language ensemble for code generation in large language model, 2024.

[38] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering, 2024.

[39] Wenkai Yang, Xiaohan Bi, Yankai Lin, Sishuo Chen, Jie Zhou, and Xu Sun. Watch out for your agents! investigating backdoor threats to llm-based agents, 2024.

[40] Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents, 2023.

[41] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023.

[42] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

[43] Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. Autocoderover: Autonomous program improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2024.

[44] Haoyu Zhao, Simran Kaur, Dingli Yu, Anirudh Goyal, and Sanjeev Arora. Can models learn skill composition from examples?, 2024.

[45] Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents, 2024.