KubePlaybook: A Repository of Ansible Playbooks for Kubernetes Auto-Remediation with LLMs

Komal Sarda*
York University
Toronto, Ontario, Canada
komal253@yorku.ca

Zakeya Namrud*
York University
Toronto, Ontario, Canada
zakeya10@yorku.ca

Marin Litoiu
York University
Toronto, Ontario, Canada
mlitoiu@yorku.ca

Larisa Shwartz
IBM T. J. Watson Research Center
Yorktown Heights, New York, USA
lshwart@us.ibm.com

Ian Watts
IBM Canada Lab
Markham, Ontario, Canada
ifwatts@ca.ibm.com

ABSTRACT

In the evolving landscape of software development and system operations, the demand for automating traditionally manual tasks has surged. Continuous operation and minimal downtimes highlight the need for automated detection and remediation of runtime anomalies. Ansible, known for its scalable features, including high-level abstraction and modularity, stands out as a reliable solution for managing complex systems securely. The challenge lies in creating an on-the-spot Ansible solution for dynamic auto-remediation, requiring a substantial dataset for in-context tuning of large language models (LLMs). Our research introduces KubePlaybook, a curated dataset with 130 natural language prompts for generating automation-focused remediation code scripts. After rigorous manual testing, the generated code achieved an impressive 98.86% accuracy rate, affirming the solution's reliability and performance in addressing dynamic auto-remediation complexities.

CCS CONCEPTS

• **Computing methodologies → Artificial intelligence**; **Natural language processing**; • **Information systems → Information systems applications**.

KEYWORDS

Kubernetes, Ansible Playbook, LLMs, GPT-4, Auto-remediation, Microservices.

ACM Reference Format:
Komal Sarda, Zakeya Namrud, Marin Litoiu, Larisa Shwartz, and Ian Watts. 2024. KubePlaybook: A Repository of Ansible Playbooks for Kubernetes Auto-Remediation with LLMs. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE Companion '24), May 7–11, 2024, London, United Kingdom.*https://doi.org/10.1145/3629527.3653665

*Both authors contributed equally to this work.

1 INTRODUCTION

In the dynamic software development landscape, the adoption of microservices has transformed scalability, flexibility, and agility [33, 39]. Kubernetes (K8s) [22], a key orchestration tool, plays a crucial role in managing microservices at scale, providing features like auto-scaling and self-healing [3]. For smaller production settings and local environments, MicroK8s, a lightweight K8s distribution, proves valuable [20]. It includes all essential components of a full K8s distribution, such as the API server, kubelet, and kubectl [21]. Kubectl, a vital command-line tool, simplifies the management and interaction with K8s clusters, enabling users to deploy applications, scale resources, create pods, and manage various K8s objects. Despite advancements in autonomic and adaptive computing [18, 35], many cloud services and applications still encounter failures, necessitating manual intervention. Additionally, the decentralized nature of microservices introduces complexity in detecting and resolving root-cause incidents [2, 16].

In microservices environments, automation plays a pivotal role in addressing complex issues swiftly [19]. AI-driven approaches have been integrated into IT operations (AIOps) particularly in self-healing processes using data-driven AI for automating incident life cycles [43]. However, challenges persist in manually creating remediation scripts, often relying on poorly organized troubleshooting guides [17]. For on-call engineers (OCEs) dealing with diverse anomalies across numerous services, the lack of organized guides can be time-consuming [2]. Anomalies, presenting differently or sharing traits across services, along with unique configuration settings, require meticulous attention. Minor script errors can lead to significant discrepancies, emphasizing the need for effective remediation scripts to expedite incident mitigation [18].

In response to these challenges, some researchers have turned to leveraging pre-trained Large Language Models (LLMs) to automatically generate remediation scripts based on identified root causes, a methodology successfully applied in various use cases [4, 5, 40]. Notable LLMs like CodeBert [7], Codex [5], LLaMa [38], GPT-Neo, GPT-NeoX [42], and GPT-4 [32] demonstrate promise for code generation tasks. While these techniques have been extensively applied in general-purpose programming languages, their adoption in IT domain-specific languages, particularly YAML, has received less attention. YAML files play a crucial role in defining and configuring key aspects of IT infrastructure [30]. In specific IT domains, such as those utilizing Ansible-YAML [12] to manage infrastructure, the integration of LLMs can streamline incident response. Companies leverage Ansible playbooks in conjunction with K8s to design intelligent, automated responses to root cause alerts, reducing the burden of routine firefighting tasks on on-call engineers (OCEs).

While progress has been made in leveraging LLMs to generate Ansible playbooks, the focus has predominantly been on enhancing productivity for existing users, with an emphasis on code completion rather than the creation of entirely new code. A critical requirement emerges for specialized Ansible playbook generation LLMs tailored for auto-remediation, functioning as an AI assistant for OCEs [19]. To address challenges related to the cost and maintenance of traditional LLMs, researchers are turning to the few-shot learning capabilities of LLMs [17, 37]. This approach enables incident-specific code generation with minimal examples, eliminating the need for extensive parameter tuning. However, the effectiveness of these models is hindered by the lack of open-source, high-quality prompts and playbook corpora [31]. To bridge this gap, we aim to create the KubePlaybook dataset, dedicated to Ansible playbooks in the context of IT automation and anomaly resolution within cloud-native environments. This dataset is crucial for enhancing the few-shot learning capabilities of LLMs, enabling them to autonomously generate more Ansible Playbooks for auto-remediation throughout the incident life cycle. Addressing these challenges brings us closer to realizing a fully autonomous AIOps environment.

Contribution: This paper introduces KubePlaybook, publicly accessible through a GitHub repository[1], a dataset featuring 130 Ansible playbooks accompanied by natural language (NL) prompts designed for code generation. NL prompts, serving as queries or descriptions, instruct LLMs to generate task-specific code. While the details about LLMs and prompts are not extensively covered due to page limitations, each NL prompt in KubePlaybook describes a root cause along with the operator's query input. Our evaluation process meticulously assesses the effectiveness of NL prompts in generating appropriate Ansible playbooks. Following this, we evaluate the functionality of each playbook by applying them to a sample microservices application to ensure their efficacy. The paper is structured as follows: Section 2 details dataset collection, generation, and description. Section 3 presents an experimental evaluation and results discussion. Subsequently, Section 4 deliberates on our work, highlighting challenges. Section 5 reviews relevant literature, and Section 6 provides conclusion and outlines future research directions.

2 KUBEPLAYBOOK FRAMEWORK

2.1 Data Collection & Generation

The development of the KubePlaybook repository employs a structured process as shown in Figure 1. We initially collect K8s Ansible playbooks from GitHub [9] and Galaxy [8], along with kubectl shell commands and real-time faults. Leveraging Ansible for system management, which is more scalable than traditional shell commands [14, 27], addresses automation challenges. To automate Ansible

[1]https://github.com/K8sPlayBook/KubePlaybook

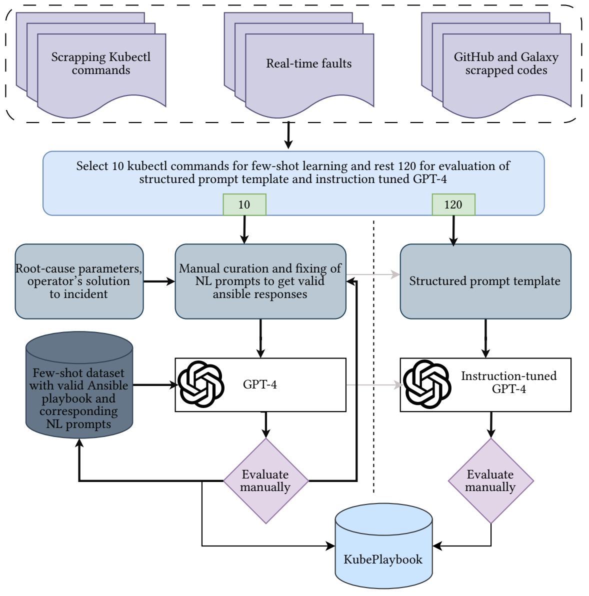
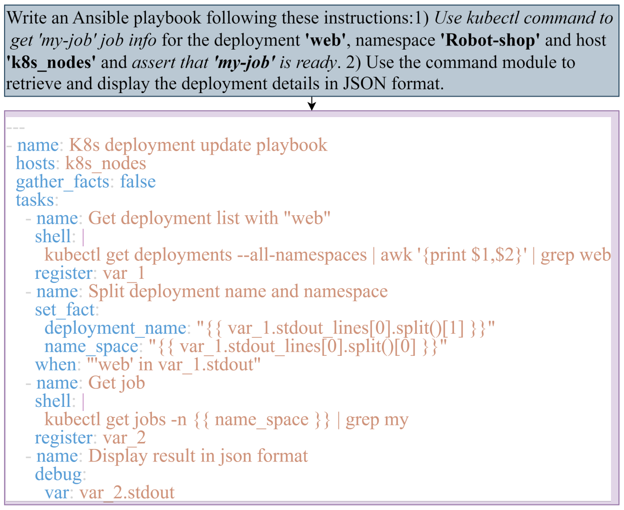

Scrapping Kubectl commands
Real-time faults
GitHub and Galaxy scrapped codes
Select 10 kubectl commands for few-shot learning and rest 120 for evaluation of structured prompt template and instruction tuned GPT-4
10
120
Root-cause parameters, operator's solution to incident
Manual curation and fixing of NL prompts to get valid ansible responses
Structured prompt template
Few-shot dataset with valid Ansible playbook and corresponding NL prompts
GPT-4
Instruction-tuned GPT-4
Evaluate manually
Evaluate manually
KubePlaybook

Figure 1: Overview of Building the Dataset

playbook creation for kubectl commands and faults, a playbooks generation approach using GPT-4 is adopted. An incident remediation dataset is built using root cause alerts and operator input as prompts. Few-shot learning on GPT-4 is applied to streamline the process, following a method adapted by many researchers [17, 37]. Initiating with 10 random kubectl commands, each is transformed into an Ansible playbook using purpose-specific prompts. The generated playbook, serving as the auto-remediation script, undergoes testing and manual examination. Adjustments are made to prompts until the desired Ansible-based remediation playbooks is obtained. A robust prompt template is integrated into KubePlaybook along with corresponding playbooks, and the few-shot learning dataset is updated for deployment with GPT-4. This iterative process is consistently applied to the initial 10 commands, tuning the GPT-4 model with newly generated samples. Careful adjustments are made to both prompts and playbooks, ensuring the creation of precise and context-sensitive playbooks tailored for incident resolution. Having generated and validated 10 structured prompt templates for auto-remediation in microservices, the same prompt structure is applied to the remaining 120 samples. The tuned GPT-4 model generates playbooks for the entire dataset, ensuring a efficient approach to incident resolution in the microservices environment.

2.2 Composition of NL Prompts

To translate Kubectl commands and address real-time faults into valid Ansible playbook, it is crucial to construct well-defined prompts that provide detailed information for the AI model. This process involves crafting precise instructions to guide the model in automatically resolving issues within microservices architectures [30, 41]. Microservices architectures consist of multiple independent services, and the initial step involves specifying precise targets, such as host names, pod names, and deployment names, in prompts for LLM-driven remediation based on root-cause alerts. Various anomaly detection models [34] and multimodal root-cause models [23, 24] are available to identify this information. This approach enables playbook to focus on specific services, minimizing disruptions to others. The next step is to define the desired automation actions, such as adjusting resources, restarting services, or implementing fixes for specific events. This action instructs the LLMs to generate Ansible playbook based on root-cause actions. This information can be derived directly from a root-cause mitigation recommendation model [2] or from operators. An example of our structured prompt and Ansible code is depicted in Figure 2. We selected smallest possible playbook for the example. This prompt incorporates placeholders like `<Host_Name>`, `<Name_space>`, and `<Deployment_name>` as indicators for inserting concrete values or variables, streamlining the management of deployments within a microservices framework. Additionally, the provided prompt serves as a guide for those involved in developing prompt templates using LLMs, demonstrating how to structure prompts to facilitate the automated generation of Ansible playbooks for addressing issues within microservices architectures. For practical application, placeholders should be replaced with actual values, and the resulting Ansible playbook must be customized to meet the unique requirements of the specific microservices environment and deployment details.


Write an Ansible playbook following these instructions:1) *Use kubectl command to get 'my-job' job info* for the deployment **'web'**, namespace **'Robot-shop'** and host **'k8s_nodes'** and *assert that* ***'my-job'*** *is ready*. 2) Use the command module to retrieve and display the deployment details in JSON format.

```
---
- name: K8s deployment update playbook
  hosts: k8s_nodes
  gather_facts: false
  tasks:
    - name: Get deployment list with "web"
      shell: |
        kubectl get deployments --all-namespaces | awk '{print $1,$2}' | grep web
      register: var_1
    - name: Split deployment name and namespace
      set_fact:
        deployment_name: "{{ var_1.stdout_lines[0].split()[1] }}"
        name_space: "{{ var_1.stdout_lines[0].split()[0] }}"
      when: "'web' in var_1.stdout"
    - name: Get job
      shell: |
        kubectl get jobs -n {{ name_space }} | grep my
      register: var_2
    - name: Display result in json format
      debug:
        var: var_2.stdout
```

Figure 2: Sample of NL prompt & its corresponding Ansible playbook generated using GPT-4.

2.3 KubePlaybook Description

The KubePlaybook dataset comprises NL prompts and Ansible playbooks, manifested as text and YAML files, respectively, aligning with targeted K8s commands. Our repository, outlined in Table 1, encompasses 130 Ansible playbooks categorized into three main classes, each featuring pairs of playbooks and corresponding NL prompts. According to Table 1, 62.3% of the playbooks focus on essential K8s commands, 19.2% are sourced from Ansible Galaxy and GitHub, and 18.5% are specifically designed for addressing chaos-engineered operational faults [26]. This categorization highlights the diverse nature of the playbooks in our dataset, covering aspects such as cluster management, real-time fault resolution, and external contributions. Our objective extends beyond constructing an Ansible playbook; we aim to extract meaningful NL prompts from each code. The repository mirrors the categorization format in directories for clarity and ease of navigation. As depicted in Figure 2, defining a placeholder in the prompt generates Ansible code that extracts target deployment details and applies the 'kubectl' command to the specified service. Notably, our dataset has no external package dependencies; it utilizes the latest versions of Ansible and Kubernetes.

Table 1: Repository Overview Description

Categories	List of Ansible Playbooks & Prompts	
Generated Ansible playbook using LLMs Essential & common for (configuration & deployment)	Cluster Management (6), Daemonsets (6), Deployments (6), Events (5), Image name (1), Jobs (3), Logs (8), Namespaces (6), Nodes (11), Pods (10), ReplicaSets (3), Replication (2), Secrets (4), Service Accounts (3), Services (4), StatefulSet (3)	62.3%
Ansible playbooks from Galaxy & GitHub	Collection of K8s tasks (25)	19.2%
Real-time faults	DNS errors (1), DNS fault (1), Node I/O stress (1), Pod API latency (1), Overrides the header values of API requests (1), Node memory hog (1), Resources overload (2), Operational Error (4), Connection refused (6), Access denied (1), Login failure (1), Process crash (1), System stuck (1)	18.5%

3 EXPERIMENTAL SETUP & EVALUATION

3.1 Experiment Configuration

To guarantee the reliability and efficiency of Ansible playbooks generated using GPT-4, a thorough evaluation process precedes their production deployment. Our evaluation involved testing the efficacy of the dataset to adapt few-shot learning on LLMs. We utilized 10 samples for few-shot learning on GPT-4 and conducted evaluations on 120 samples. We utilized the GPT-4 model, specifically opting for the gpt-4-1106 [36]-preview version-recognized as the latest and most proficient model for code comprehension from OpenAI. Testing occurred on a t2.2xlarge EC2 instance with Ubuntu, installing Robot-shop [1] and QoTD [11]. We validated both syntactic correctness and functional soundness during this crucial phase, focusing on the effectiveness of automated deployment. Hyperparameter tuning for GPT-4 was tailored to our needs, utilizing maximum output length with 10 samples for few-shot data, optimizing within token count limits. After iterative experimentation, optimal hyperparameters-temperature [10] at 0.6 and Top-P [10] at (1.0)-were chosen to fine-tune the model's performance for precise YAML configurations from NL instructions. This meticulous testing and tuning ensure robust Ansible playbooks, ready for seamless deployment in real-world production environments.

3.2 Evaluation Methodology

We performed manual testing on each Ansible playbook on our setup to ensure successful execution. The Average Correctness (AC) metric was employed for evaluation, focusing on the precision of individual code blocks or sub-tasks within Ansible playbooks. These sub-tasks, crucial units in Ansible, encompass actions like package installations, service configurations, or file transfers, influencing the overall playbook accuracy. The AC metric is defined by Equation 1, evaluating the accuracy of generated Ansible playbooks. For each Ansible playbook code (*APC*), *AC* computes the accuracy per task by comparing the correctly executed tasks (C) to the total tasks (n) in that *APC*. The accuracy ratios of all tasks across all *APCs* are summed, providing an aggregate view of performance. The final AC value represents the average of these task accuracy ratios, obtained by dividing the sum by the total number of *APCs* (m). This nuanced approach offers insights into the quality of individual code blocks beyond a binary overall judgment.

$$AC = \frac{\sum_{j=1}^{m}(C_j/N_j * 100)}{m} * 100 \tag{1}$$

We refrained from utilizing other lexical metrics such as BLEU-4 [28] or semantic metrics like BERTScore[28] due to the unavailability of ground truth. Table 2 presents the performance metrics, highlighting GPT-4's exceptional accuracy in generating Ansible playbook. With a 98.86% accuracy rate for code pertaining to Kubectl commands. Additionally, when assessed against 24 real-time faults, GPT-4 achieved a 92.36% accuracy rate. This stands in stark contrast to a 60% accuracy rate for identical tasks performed by humans.

Table 2: Evaluation of Ansible playbooks

Task Source	Accuracy
Kubectl command	98.86%
Code was written by a human from GitHub	60%
Real-time fault	92.36%

4 DISCUSSION & CHALLENGES

To foster operator trust in automated code generation for expediting incident resolution, a large, real-world dataset is crucial for fine-tuning models. Towards this, our dataset serves as a valuable starting point. We devised a specific pattern for constructing prompts using operator inputs, recognizing the potential variations across industries. The evaluation involved testing prompt construction and Ansible playbook generation on e-commerce microservice applications like robot-shop and QoTD using Kubectl. However, it's important to note that these results may not universally apply to different microservice applications and tools. Additionally, while the GPT-4 were instruction-tuned and evaluated to generate a specific dataset, variations in results and accuracy may occur for different datasets. Rigorous manual tweaking and testing were performed to ensure the reliability of each playbook on our setup.

5 RELATED WORK & APPLICATION

Benchmark datasets have become crucial for advancing applied AI research, particularly as the demand for evaluating models across diverse applications grows. Among the key contributions to this area, Hendrycks et al.[13] pioneered the assessment of large transformer language models in competitive programming with the introduction of the APPS dataset, featuring 10,000 coding competition problems. The CodeXGLUE [25] and CodeSearchNet [15] datasets further extend the toolset for researchers with tasks ranging from code summarization to code translation, and offering professional annotations for NL queries across several programming languages. However, Ansible was not considered. 20-MAD [6] is a dataset that links the commit and issue data of the Mozilla and Apache projects. Our work distinguishes itself by focusing on the generation of Ansible playbooks, a niche not covered by the aforementioned datasets that primarily utilize NL for code generation. Although the work of Ahmed et al.[2] is closely related through its use of LLMs for text generation, our emphasis remains on code generation. Moreover, unlike the Andromeda [29], which provides an overview of the Ansible Galaxy ecosystem, our dataset is specifically tailored for training LLMs with generated and scraped Ansible playbooks along with their associated prompts. To our knowledge, KubePlaybook is the starting point for K8s-based Ansible playbook benchmark dataset crafted using LLMs. Aimed at auto-remediation microservices, it sets a new precedent for employing LLM-generated datasets in practical applications. Table 3 outlines the distinctions between the current cutting-edge research initiatives and our methodology. This includes the employment of LLMs for the generation of the Ansible playbook, the automation of code scraping techniques, the detailed generation of report descriptions, and the careful crafting involved in prompt engineering.

Table 3: Comparison between state-of-the-art research and our approach.

References	LLMs usage	Scraped repositories	Ansible playbook	NL prompts
Paper [2]	✗	✓	✗	✓
Paper [5]	✗	✓	✗	✗
Paper [6]	✗	✓	✗	✗
Paper [13]	✓	✗	✗	✗
Paper [15]	✓	✓	✗	✗
Paper [25]	✓	✗	✗	✗
Paper [29]	✗	✓	✓	✗
Our Dataset	✓	✓	✓	✓

6 CONCLUSION & FUTURE DIRECTIONS

In conclusion, while the application of AI, especially LLMs, in automating IT operations holds promise for self-healing mechanisms, challenges persist in applying these advancements to IT-centric languages like YAML. The introduction of the KubePlaybook dataset, comprising 130 NL prompts and Ansible playbooks, represents a significant step towards addressing this gap. It facilitates the contextual learning of LLMs to generate Ansible-YAML scripts for automated remediation tasks, a pivotal development for advancing IT automation in cloud-native environments. It is poised to enhance incident response in the dynamic realm of IT operations. Future research may focus on augmenting playbook quality by incorporating more real-time fault data examples. The manual creation of corresponding NL prompts for each script, a meticulous and time-consuming process, motivates us to explore automatic prompt generation for auto-remediation tasks in microservices environments. Additionally, investigating the performance and adaptability of different LLMs for playbook generation presents a potential area of exploration.

REFERENCES

[1] 1steveww et al. 2023. Robot Shop is a sample microservice application. Retrieved Mar 6, 2023 from https://github.com/instana/robot-shop

[2] Toufique Ahmed, Supriyo Ghosh, Chetan Bansal, Thomas Zimmermann, Xuchao Zhang, and Saravan Rajmohan. 2023. Recommending Root-Cause and Mitigation Steps for Cloud Incidents using Large Language Models. *arXiv preprint arXiv:2301.03797* (2023).

[3] Meriem Azaiez and Walid Chainbi. 2016. A multi-agent system architecture for self-healing cloud infrastructure. In *Proceedings of the International Conference on Internet of things and Cloud Computing*. 1–6.

[4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[5] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

[6] Maëlick Claes and Mika V Mäntylä. 2020. 20-MAD: 20 years of issues and commits of Mozilla and Apache development. In *Proceedings of the 17th International Conference on Mining Software Repositories*. 503–507.

[7] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155* (2020).

[8] Ansible Galaxy. 2024. Ansible Galaxy. Retrieved Jan 30, 2024 from https://galaxy.ansible.com/ui/collections/

[9] GitHub. 2024. GitHub. Retrieved Jan 30, 2024 from https://github.com/

[10] Fabian Gloeckle, Baptiste Roziere, Amaury Hayat, and Gabriel Synnaeve. 2023. Temperature-scaled large language models for Lean proofstep prediction. In *The 3rd Workshop on Mathematical Reasoning and AI at NeurIPS'23*.

[11] Red Hat. 2023. QoTD. Retrieved Oct 11, 2023 from https://github.com/redhat-developer-demos/qotd.git

[12] Red Hat. 2023. Red Hat Ansible Automation Platform. Retrieved Nov 27, 2023 from https://www.redhat.com/en/technologies/management/ansible

[13] Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, et al. 2021. Measuring coding challenge competence with apps. *arXiv preprint arXiv:2105.09938* (2021).

[14] Eric Horton and Chris Parnin. 2022. Dozer: migrating shell commands to ansible modules via execution profiling and synthesis. In *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice*. 147–148.

[15] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. Codesearchnet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436* (2019).

[16] Jiajun Jiang, Weihai Lu, Junjie Chen, Qingwei Lin, Pu Zhao, Yu Kang, Hongyu Zhang, Yingfei Xiong, Feng Gao, Zhangwei Xu, et al. 2020. How to mitigate the incident? an effective troubleshooting guide recommendation technique for online service systems. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1410–1420.

[17] Yuxuan Jiang, Chaoyun Zhang, Shilin He, Zhihao Yang, Minghua Ma, Si Qin, Yu Kang, Yingnong Dang, Saravan Rajmohan, Qingwei Lin, et al. 2023. Xpert: Empowering Incident Management with Query Recommendations via Large Language Models. *arXiv preprint arXiv:2312.11988* (2023).

[18] Cornel Klein, Reiner Schmid, Christian Leuxner, Wassiou Sitou, and Bernd Spanfelner. 2008. A survey of context adaptation in autonomic computing. In *Fourth International Conference on Autonomic and Autonomous Systems (ICAS'08)*. IEEE, 106–111.

[19] Sarda Komal, Namrud Zakeya, Rouf Raphael, Ahuja Harit, Rasolroveicy Mohammadreza, Litoiu Marin, Shwartz Larisa, and Watts Ian. 2023. ADARMA Auto-Detection and Auto-Remediation of Microservice Anomalies by Leveraging Large Language Models. In *Proceedings of the 33rd Annual International Conference on Computer Science and Software Engineering*. 200–205.

[20] Heiko Koziolek and Nafise Eskandani. 2023. Lightweight Kubernetes Distributions: A Performance Comparison of MicroK8s, k3s, k0s, and Microshift. In *Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering*. 17–29.

[21] Kubectl. 2024. Kubectl. Retrieved Jan 30, 2024 from https://kubernetes.io/docs/reference/kubectl/

[22] Kubernetes. 2024. Kubernetes. Retrieved Jan 30, 2024 from https://kubernetes.io/

[23] Zeyan Li, Junjie Chen, Rui Jiao, Nengwen Zhao, Zhijun Wang, Shuwei Zhang, Yanjun Wu, Long Jiang, Leiqin Yan, Zikai Wang, et al. 2021. Practical root cause localization for microservice systems via trace analysis. In *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*. IEEE, 1–10.

[24] Fred Lin, Keyur Muzumdar, Nikolay Pavlovich Laptev, Mihai-Valentin Curelea, Seunghak Lee, and Sriram Sankar. 2020. Fast dimensional analysis for root cause investigation in a large-scale service environment. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 4, 2 (2020), 1–23.

[25] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, et al. 2021. Codexglue: A machine learning benchmark dataset for code understanding and generation. *arXiv preprint arXiv:2102.04664* (2021).

[26] Sehrish Malik, Moeen Ali Naqvi, and Leon Moonen. 2023. CHESS: A Framework for Evaluation of Self-adaptive Systems based on Chaos Engineering. *arXiv preprint arXiv:2303.07283* (2023).

[27] Pavel Masek, Martin Stusek, Jan Krejci, Krystof Zeman, Jiri Pokorny, and Marek Kudlacek. 2018. Unleashing full potential of ansible framework: University labs administration. In *2018 22nd conference of open innovations association (FRUCT)*. IEEE, 144–150.

[28] Nabor C Mendonça, Pooyan Jamshidi, David Garlan, and Claus Pahl. 2019. Developing self-adaptive microservice systems: Challenges and directions. *IEEE Software* 38, 2 (2019), 70–79.

[29] Ruben Opdebeeck, Ahmed Zerouali, and Coen De Roover. 2021. Andromeda: A dataset of Ansible Galaxy roles and their evolution. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 580–584.

[30] Saurabh Pujar, Luca Buratti, Xiaojie Guo, Nicolas Dupuis, Burn Lewis, Sahil Suneja, Atin Sood, Ganesh Nalawade, Matt Jones, Alessandro Morari, et al. 2023. Automated Code generation for Information Technology Tasks in YAML through Large Language Models. *arXiv preprint arXiv:2305.02783* (2023).

[31] Laria Reynolds and Kyle McDonell. 2021. Prompt programming for large language models: Beyond the few-shot paradigm. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–7.

[32] Katharine Sanderson. 2023. GPT-4 is here: what scientists think. *Nature* 615, 7954 (2023), 773.

[33] Komal Sarda. 2023. Leveraging Large Language Models for Auto-remediation in Microservices Architecture. In *2023 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*. IEEE, 16–18.

[34] Jacopo Soldani and Antonio Brogi. 2022. Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: A survey. *ACM Computing Surveys (CSUR)* 55, 3 (2022), 1–39.

[35] Roy Sterritt, Manish Parashar, Huaglory Tianfield, and Rainer Unland. 2005. A concise introduction to autonomic computing. *Advanced engineering informatics* 19, 3 (2005), 181–187.

[36] Kaiming Tao, Zachary A Osman, Philip L Tzou, Soo-Yon Rhee, Vineet Ahluwalia, and Robert W Shafer. 2024. GPT-4 Performance on Querying Scientific Publications: Reproducibility, Accuracy, and Impact of an Instruction Sheet. (2024).

[37] Catherine Tony, Markus Mutas, Nicolás E Díaz Ferreyra, and Riccardo Scandariato. 2023. LLMSecEval: A Dataset of Natural Language Prompts for Security Evaluations. *arXiv preprint arXiv:2303.09384* (2023).

[38] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[39] Hulya Vural, Murat Koyuncu, and Sinem Guney. 2017. A systematic literature review on microservices. In *Computational Science and Its Applications–ICCSA 2017: 17th International Conference, Trieste, Italy, July 3-6, 2017, Proceedings, Part VI 17*. Springer, 203–217.

[40] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903* (2022).

[41] Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C Schmidt. 2023. A prompt pattern catalog to enhance prompt engineering with chatgpt. *arXiv preprint arXiv:2302.11382* (2023).

[42] Frank F Xu, Uri Alon, Graham Neubig, and Vincent Josua Hellendoorn. 2022. A systematic evaluation of large language models of code. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*. 1–10.

[43] Tianyin Xu, Xinxin Jin, Peng Huang, Yuanyuan Zhou, Shan Lu, Long Jin, and Shankar Pasupathy. 2016. Early detection of configuration errors to reduce failure damage. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 619–634.