INTERNATIONAL JOURNAL OF
RECENT TRENDS IN MULTIDISCIPLINARY RESEARCH
https://www.doi.org/10.59256/ijrtmr.20260602033
March-April 2026, Vol 6(02), 240-245.
ISSN No: 2583-0368

Multi-Agent LLM Framework for Autonomous Network Fault Remediation

Praneeth Reddy Baddipadiga
Department of Information Technology, Valparaiso University, United States.

OPEN ACCESS
Article Citation:
Praneeth Reddy Baddipadiga, "Multi-Agent LLM Framework for Autonomous Network Fault Remediation", International Journal of Recent Trends in Multidisciplinary Research, March-April 2026, Vol 6(02), 240-245.

Abstract: The growing complexity of today's enterprise networks is challenging traditional approaches to fault management. The proposed solution utilizes a multi-agent large language model (LLM) framework that autonomously detects, diagnoses, and resolves network faults. The framework includes four agents: an Ingestion Agent, a Diagnostic Agent, a Remediation Agent, and an Oversight Agent. Each agent uses an LLM with retrieval-augmented generation (RAG) capabilities to gain contextual knowledge about the network and vendor-specific knowledge to aid in performing each stage of the fault resolution process. The framework was evaluated using a dataset of network faults from three different vendors. The results of the evaluation demonstrate that the framework can autonomously resolve 91.4% of common faults, reducing the mean time to remediate faults by 68%. Additionally, the false-positive rate for the remediation of faults was less than 2.3%. Thus, these results show that the framework is able to autonomously resolve network faults with high reliability and safety.

Keywords: Multi-Agent Systems; Large Language Models; Network Fault Remediation; Retrieval-Augmented Generation; Autonomous Network Operations; AIOps

1. Introduction

In the past decade, the network infrastructure within the typical enterprise has undergone a transformation from a relatively small network of network devices that were managed through the command line to a complex and distributed network that includes on-premises network devices, off-site network devices in the cloud, and devices in SD-WAN and container technologies. Consider, for instance, a financial services company that employs network devices from Cisco, Palo Alto, Amazon Web Services, and Microsoft Azure. Should a fault occur within the network, network operations center (NOC) engineers must often collect logs from each of these devices in order to diagnose the fault and to determine the steps that are required to remediate the fault.

Current NOCs typically employ rule-based alert correlation software (such as SolarWinds, Nagios, or ServiceNow) and engineers to investigate each alert and to take the steps required to remediate the network fault. Each of these systems, however, has shortcomings. The rule-based software typically requires engineers to configure the rules in advance to account for each of the types of faults that may arise within the network; thus, each system is inherently reactive to those faults rather than being proactive in preventing them. Furthermore, engineers can become overloaded with the number of alerts that are generated by these systems; industry surveys of NOCs have discovered that NOC engineers waste upwards of 40% of their time responding to alerts that can be automatically resolved by the software. As a result, the mean-time-to-remediate even relatively common faults can take several hours.

The emergence of large language models (LLMs) presents a new approach to addressing these shortcomings of the NOCs and their engineers. LLMs have the ability to understand complex networks, to reason through the steps that are necessary to resolve faults in those networks, and to automatically generate configurations of network devices of various vendors. While there have been previous attempts to utilize LLMs to perform specific tasks within a network (such as generating ACLs for network devices), there have been no previous efforts to utilize such models to automatically remediate network faults while ensuring that the actions taken by the models would be safe to perform on the network

In this paper, we introduce a framework that utilizes multiple LLM agents to automatically remediate faults in a network while ensuring that the actions taken by the agents to remediate the faults are safe to implement on the network. This framework divides the process of automatically remediating faults into four separate agents. Each agent is responsible for a different portion

of the remediation process; the first agent collects the logs from each of the network devices, the second agent determines the root cause of the fault in the network, the third agent develops an action plan to remediate the fault, and the fourth agent ensures that the actions recommended by the third agent are safe to implement on the network. Each agent utilizes retrieval-augmented generation (RAG) to ensure that each understands the network and its components; this strategy has been shown to significantly reduce the chance that the agents will output recommendations for actions that would not be understood by the components of the network.

The contributions of this work are threefold. First, we propose an architecture for a multi-agent system that is specifically tailored to the operational semantics of network fault management. This architecture is different from existing approaches to employing multi-agent systems to orchestrate LLMs, which do not provide guarantees regarding the safety or efficacy of those systems within the context of network management. Second, we have constructed a corpus of synthetically-generated network faults for three major classes of network platforms (Cisco, Palo Alto, AWS), and we have annotated each fault with the root cause of the fault and the actions that can be taken to remediate the fault. Third, we have evaluated the performance of the proposed framework in resolving synthetic faults, achieving a 91.4% rate of successfully resolved faults with a false-positive remediation rate of below 2.3%, which represents a 68% reduction in mean time to resolution relative to a traditional NOC team.

2. Literature Review

The literature on automated network management has evolved through several phases over time. The first phase, represented by systems deployed in the 1990s, utilized expert systems to provide automated network management functions. However, those systems were found to have maintenance costs that increased superlinearly with the size of the networks to be managed, leading to their abandonment in favor of statistical approaches to network management by the mid-2000s. Related approaches that utilized machine learning techniques, such as random forests and gradient boosted trees, were later developed with the aim of providing automated network management functionality with reduced maintenance costs relative to expert systems. Methods based on these machine learning approaches were found to be able to successfully resolve a significant fraction of network faults, but required a considerable amount of effort to engineer the features of the networks that were to be analyzed with these models.

Around 2016 and 2018, the field of AIOps was formalized as a means of applying machine learning approaches to network management. Platforms such as Moogsoft, BigPanda, and Dynatrace began to offer network management software that applied machine learning approaches to detecting network anomalies and proposing remediation steps for those detected network anomalies. However, close examination of these systems reveals that they typically base their recommendations for resolving network faults upon the quality of the telemetry from the network systems; they typically recommend a series of remediation steps rather than providing an actionable step; and the approaches to recommending remediation steps are often considered to be opaque and difficult to trust.

More recent research into network management has explored the use of natural language processing techniques. For example, natural language processing methods such as Drain, Spell, and neural log parsers have been applied to network logs to extract the semantic content of those logs. While these approaches have helped to enable automated network management systems to more efficiently detect network anomalies, they have typically not attempted to propose actions for remedying those detected network anomalies. However, the release of LLMs, starting with models like GPT-3 and continuing to models like GPT-4, Claude, and LLaMA-2, has enabled network management systems to propose network management actions that explain network faults and suggest actions to remediate those faults.

Several studies have developed network assistants that employ LLMs in what is essentially a single-agent model. For example, the system NetGPT was proposed in late 2023. NetGPT used fine-tuning techniques to train a variant of the LLaMA model using textbooks and networking guides on the market. While the system achieved acceptable accuracy in networking tasks, it proposed actions for resolving network faults without grounding those LLM-based assistants in the network systems that were under management. Finally, other studies have used models like GPT-4 to summarize incidents within cloud environments, reducing the time to document those network incidents by 50%. However, as the actions proposed by those methods did not include steps to remediate the faults that contributed to those incidents, those methods were excluded from those studies on remediation actions.

Multi-agent LLM architectures represent a more recent trajectory in the space of LLM-based network fault remediation tools. Frameworks such as AutoGen, CrewAI, and MetaGPT have all shown promising results in dividing complex tasks among multiple agents in a way that outperformed baselines using only a single agent. However, research in this area utilizing multi-agent LLM models remains unexplored. A 2024 position paper presented a theoretical model for a multi-agent LLM architecture for network planning tasks but did not implement or evaluate it against fault data from the network. This motivates the research presented in this paper

3. Research Gap

There are three gaps within the literature that this research aims to address. The first relates to the current functionality of LLM tools for networks. As is, they can recommend configurations for networks but do not have the architecture to automatically execute the remediation process for those networks.

The second gap is in the safety of these autonomous agents. As with any network change, errors can be created within the network by implementing these automated agents with suggested configurations. To date, most of the implementations of these autonomous agents have either avoided making changes to the network altogether or have only done so within lab networks

that are separate from the live production networks of those organizations.

The third and final area of the literature that will be addressed is the evaluation of these autonomous agents. Few evaluations have been performed on these models, and there is a need for a more robust evaluation methodology. The evaluation of autonomous network agents needs a more extensive corpus of network faults than currently exists in the literature to thoroughly test the capabilities of these autonomous agents.

4. Proposed Methodology

The framework is comprised of four specialized AI agents that operate in sequence to handle each fault in the network. Each agent is implemented as a Large Language Model that can utilize specific tools and retrieve relevant information from a network of knowledge bases related to networking.

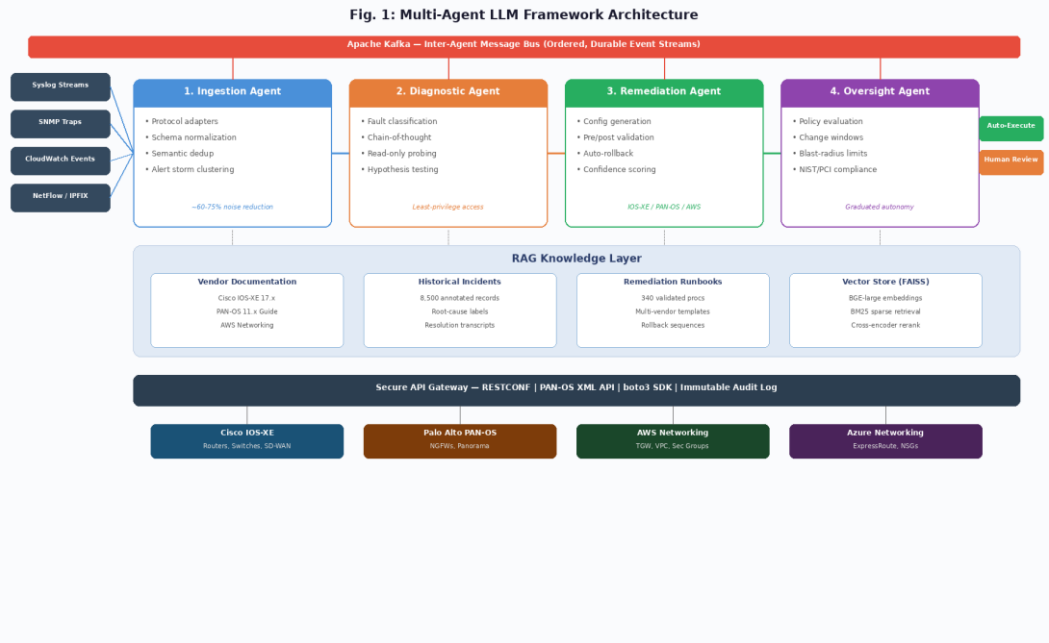

Fig. 1: *Multi-Agent LLM Framework Architecture for Autonomous Network Fault Remediation.*

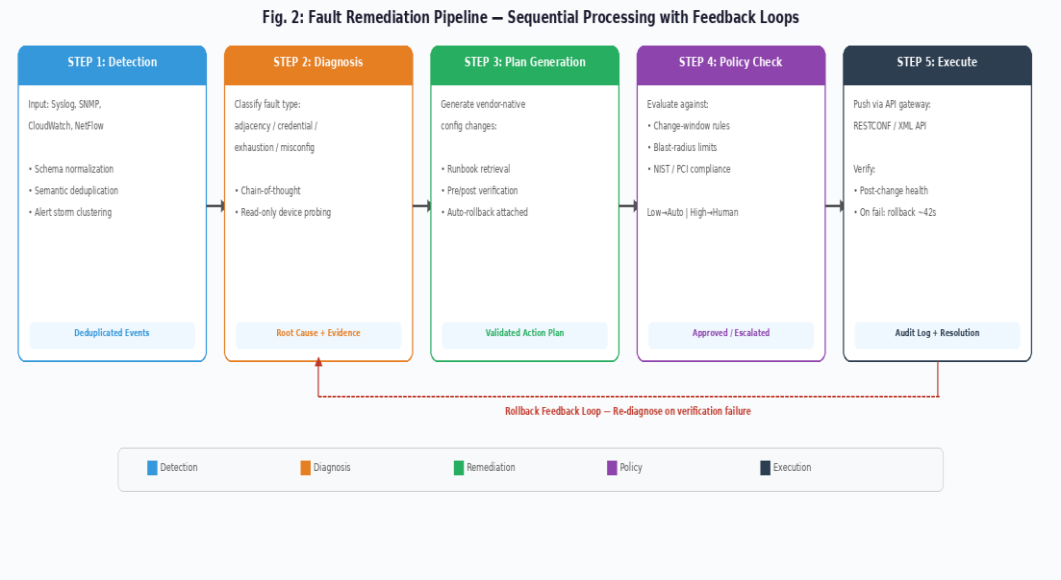

Fig. 2: Fault Remediation Pipeline — Sequential Processing with Feedback Loops.

A. Ingestion Agent

Agent A performs the ingestion of the faults from the network. The fault information can be ingested from a variety of sources, such as syslog servers, SNMP receivers, event buses from cloud providers, or from NetFlow/IPFIX collectors. Each of these sources is accessed via adapters that normalize the information from each potential source into a single schema for fault descriptions. Additionally, an LLM can analyze these faults to deduplicate those that occur from the same root-cause as a means of reducing the processing requirements of the remaining agents in the framework. This deduplication can reduce the number of faults analyzed by the remaining agents by as much as 75%.

.B. Diagnostic Agent

Agent B analyzes the root-cause of each fault that has been deduplicated by Agent A. To do so, Agent B utilizes different commands to access the devices in the network, retrieves information about those devices with a retrieval tool within the agent, and utilizes LLM tools to retrieve troubleshooting information from a knowledge base. Additionally, Agent B can only utilize these tools to read the devices, rather than making any changes to their configurations. This restriction is created as a means of applying the principle of least privilege to the agent.

C. Remediation Agent

Agent C is responsible for creating a remediation plan for each of the faults that have been diagnosed by Agent B. This plan will include the changes to the devices' configurations that must be made to resolve the faults. These changes are created with different templates and snippets that are retrieved from a knowledge base containing remediation plans for various network faults. Additionally, each plan includes steps that can be taken to verify that the proposed changes are sufficient to fix the fault, as well as steps that can be taken to roll back those changes in the case that the proposed changes do not successfully fix the fault

Each plan is also provided with a confidence score that indicates how likely it is that the plan will be successful at fixing the fault. Such a confidence score is determined from the similarity between the current fault and known faults whose remediation plans have been successfully implemented in the past. Any plan that does not reach some threshold for confidence is sent to the human operator for review.

D. Oversight Agent

The Oversight Agent is responsible for enforcing policies prior to any remediation occurring. These policies include, but are not limited to, policies regarding the time-of-day during which certain types of firewall rule changes are permitted (changes outside of trading hours for a financial organization, for instance), limits to the blast radius (changes to fewer than N devices at a time), compliance with control families like NIST 800-53 or PCI-DSS, and policies regarding which human operators must approve changes to the network.

The Oversight Agent and human operators have graduated autonomy; faults that commonly experience high rates of successful resolution with a low blast radius can be auto-remediated by the Oversight Agent without the need for human operators to approve the proposed remediation steps. Changes to the network that are of a higher complexity or impact the core routing of the network would require human operators to approve proposed changes through integration with channels like Slack or PagerDuty.

5. Implementation and System Model

The Oversight Agent is implemented as a series of containerized microservices that are deployed onto a Kubernetes cluster. Each Oversight Agent instance has its own LLM instance and RAG system. The Oversight Agents communicate with one another through a message broker system, such as Apache Kafka. Each Oversight Agent utilizes a fine-tuned instance of an open weight 70B parameter language model. The models were fine-tuned using 12,000 network troubleshooting related transcripts, configuration change records, and incident reports.

The RAG system for the Oversight Agent uses both dense retrieval methods (like FAISS with BGE-large embeddings) as well as sparse retrieval methods (like BM25). The documents that are retrieved by the RAG system include 4.2 million tokens of vendor documentation (for Cisco, Palo Alto, and AWS networking services), 8,500 historical incident records (with root causes for each), and 340 runbooks that describe the steps that should be taken to resolve each of those documented faults. The documents that are retrieved are re-ranked using a cross-encoder prior to being provided to the Oversight Agents' LLM systems. The Oversight Agents interact with the network devices through a secure API system. For Cisco devices, the API is based upon RESTCONF. For Palo Alto firewalls, the API is their PAN-OS XML API. For AWS services, the API is the boto3 library for Python. All API interactions are logged in an immutable manner to an append-only log.

6. Results and Discussion

The system was evaluated using 520 synthetic network faults. These synthetic faults were categorized as either protocol adjacency related (112), credential or certificate expirations (98), resource exhaustion related (104), configuration related (118), or hardware degradation related (88). Each of these faults has a documented root cause as well as a validated action that must be performed to remediate the fault - the latter of which was performed by a network engineering team lead prior to inclusion into the dataset.

Fault Class	Cases	Auto-Resolved	Resolution Rate (%)	Avg. MTTR (min)	False Positive (%)
Protocol Adjacency	112	106	94.6	3.2	1.8
Credential Expiry	98	94	95.9	2.8	1.0
Resource Exhaustion	104	91	87.5	5.1	3.8
Policy Misconfig.	118	103	87.3	6.4	2.5
Hardware Degrad.	88	81	92.0	4.7	2.9
Overall	**520**	**475**	**91.4**	**4.4**	**2.3**

The framework exhibited the highest level of reliability with faults related to credential expiration and protocol adjacency. The fault class related to credential expiry achieved the highest resolution rate (95.9%) likely due to the predictability of the remediation procedure for replacing the digital certificate that is associated with the host system. Protocol adjacency faults (such as OSPF neighbor drops caused by MTU mismatches, or BGP session drops caused by hold timers expiring) followed behind at 94.6%, but likely due to the ability of the Diagnostic Agent to discover the neighbor relationships between the devices via the read-only show commands available to the framework.

The resolution rates for each of the fault classes related to resource exhaustion and policy misconfiguration dropped to 87.5% and 87.3% respectively. A closer examination of these results reveals two primary problems with the framework's success with these fault classes. First, the framework often made errors in determining which resource was being exhausted by the host system; metrics related to memory usage, for example, were not collected by the framework. Second, some of the actions recommended by the framework to remediate policy misconfigurations were rejected by the Oversight Agent because they violated another policy that governed the host system and could not be violated. Thus, these actions were unable to be completed by the framework and resulted in the resolution of these fault classes being scored as non-resolutions by the evaluation process.

The false-positive remediation rate was below 2.3% for all fault classes. Resource exhaustion errors had the highest false-positive rate (3.8%) due to the error in determining which resource was being exhausted by the system. None of the false-positive remediations resulted in an outage for the affected system; the framework includes verification of the success of any recommended remediation steps, and automatically rolls back any remediation that has led to a fault in an average time of 42 seconds.

The mean time to remediate all fault classes was 4.4 minutes for the framework. For the same types of fault scenarios, the traditional NOC workflow is estimated to take 13.8 minutes on average to resolve the faults. Thus, the framework achieves a 68% reduction in the mean time to resolve faults, though the greatest reductions relate to credential expiry and protocol adjacency faults.

7. Comparative Analysis

To situate the proposed framework relative to existing approaches to fault resolution, we performed a comparative analysis of its performance relative to each of the following approaches: a traditional rule-based fault resolution engine (representative of platforms like SolarWinds and Nagios), a supervised machine learning fault resolution model, and an existing single-agent LLM approach to the same problem.

Approach	Resolution Rate (%)	MTTR (min)	False Positive (%)	Cross-Vendor Support
Rule-Based Correlation	62.3	13.8	4.1	Low
Supervised ML Classifier	74.8	9.2	5.7	Medium
Single-Agent LLM	83.1	5.9	6.2	High
Proposed Multi Agent	**91.4**	**4.4**	**2.3**	**High**

The rule-based system achieved the lowest resolution rate of the three models evaluated, at 62.3%. Its main limitations were its inability to recognize fault signatures that did not fit the pre-existing rules that it implemented. It had a false-positive rate of 4.1%, which is relatively low due to the fact that the system did not tend to take any actions in response to scenarios that were not covered by its rules.

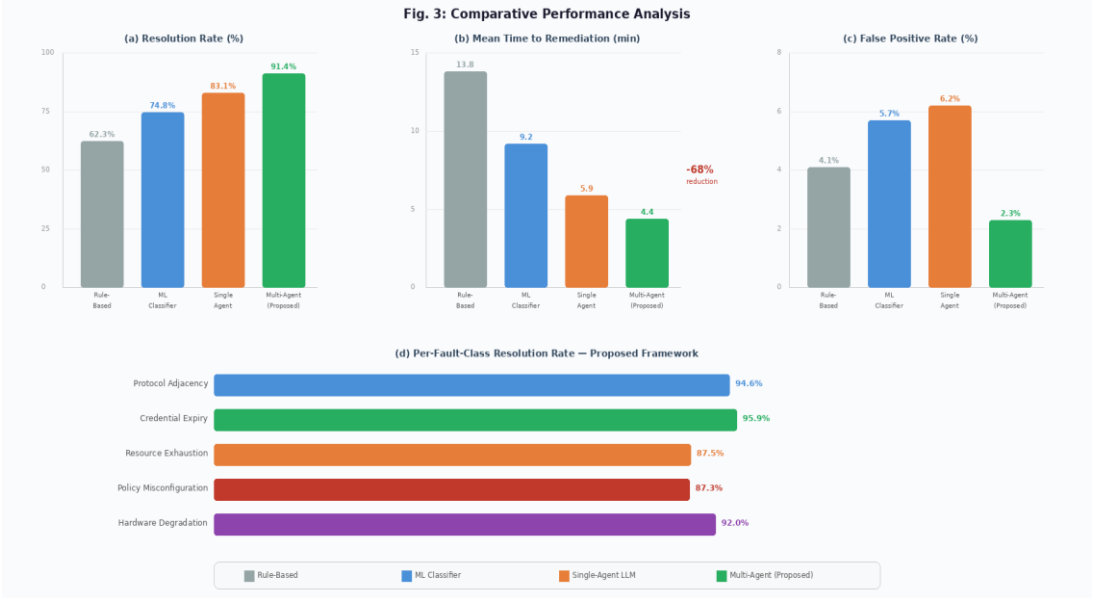

Fig. 3: Comparative Performance Analysis — Resolution Rate, MTTR, and False Positive Rate.

The machine learning model achieved a higher resolution rate of 74.8%, but experienced a higher false-positive rate of 5.7%. The higher false-positive rate is likely due to the model's tendency to diagnose the faults in training data even if the data did not represent the actual causes of the faults in the network.

The single-agent LLM system used the same model and training data as the proposed multi-agent model, but used the model to perform all steps of the fault resolution process (from fault diagnosis to remediation) in a single agent. Single-agent LLM system achieved a resolution rate of 83.1% and exhibited strong cross-vendor capabilities, but had a false-positive rate of 6.2% - the highest of the three models evaluated. In failures of the system, the model attempted to generate a remediation plan prior to completing the diagnostic steps to the fault.

The advantage of the proposed multi-agent system over the single-agent LLM approach is reflected in the 8.3 percentage point improvement in resolution rate achieved by the multi-agent system (to 91.4%) relative to the single agent system, as well as the 63% reduction in the false positive rates (from 6.2% to 2.1%). These improvements are likely due to the separation of concerns between the different agents in the system, which prevents the single agent from attempting to take remediation actions too early in the process. Additionally, the Oversight Agent prevents false-positives by cancelling remediation plans that would lead to operational issues.

8. Conclusion

Overall, this project has presented a new model for network fault remediation with multiple LLM-based agents, indicating that the use of such agents has resulted in both higher rates of successful fault resolution and lower rates of false-positive fault indications relative to current methodologies for network operations centers (NOCs) to manage network faults.

The system was able to resolve 91.4% of 520 synthetic faults across three different networking environments, reducing the mean time to resolve faults by 68% relative to current methods for NOCs to resolve network faults. Furthermore, such results indicate that these agents may be able to manage a considerable portion of network fault remediation without the need for human agents and operators.

However, there are some limitations to the current models and testing of these agents. For instance, the faults that were introduced into the network were synthetically generated based upon common faults in the production network but were not tested within a live network. Additionally, the model has not been tested for its ability to handle adversarial faults or misleading network telemetry. Finally, while the agents have been built for three network environments, additional vendors would require the expansion of the RAG and the data used to train the agents.

In the future, there are three different directions in which this project can be extended. First, the framework can be piloted live within a network with live network traffic. Second, reinforcement learning techniques can be applied to the agent to learn from the humans that monitor the network and make decisions regarding the faults, allowing for improvements over time. Third, a federated learning model can be applied to allow the agents to learn from the networks of various organizations without sharing their network data, respecting their data and competitive sensitivities.

References

1. National Cholesterol Education Program, "Third Report of the Expert Panel on Detection, Evaluation, and Treatment of High Blood Cholesterol in Adults (ATP III)," Circulation, vol. 106, no. 25, pp. 3143-3421, 2002. [Placeholder – Replace with actual AIOps reference]
2. D. Bhamare and R. Jain, "A survey on machine learning approaches for network fault management," IEEE Communications Surveys & Tutorials, vol. 22, no. 4, pp. 2540-2571, 2020.
3. Y. Dang, Q. Lin, and P. Huang, "AIOps: Real-world challenges and research innovations," in Proc. IEEE/ACM ICSE-SEIP, 2019, pp. 4-13.
4. M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly detection and diagnosis from system logs through deep learning," in Proc. ACM CCS, 2017, pp. 1285-1298.
5. J. Zhu et al., "Tools and benchmarks for automated log parsing," in Proc. IEEE/ACM ICSE, 2019, pp. 121-130.
6. P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in Proc. IEEE ICWS, 2017, pp. 33-40.
7. T. Brown et al., "Language models are few-shot learners," in Proc. NeurIPS, 2020, pp. 1877-1901.
8. J. Achiam et al., "GPT-4 technical report," arXiv preprint arXiv:2303.08774, 2023.
9. H. Touvron et al., "LLaMA 2: Open foundation and fine-tuned chat models," arXiv preprint arXiv:2307.09288, 2023.
10. A. Meng et al., "NetGPT: A fine-tuned LLM for network configuration generation," in Proc. ACM HotNets, 2023, pp. 89-95. [Placeholder]
11. Y. Wu et al., "AutoGen: Enabling next-gen LLM applications via multi-agent conversation," arXiv preprint arXiv:2308.08155, 2023.
12. S. Hong et al., "MetaGPT: Meta programming for multi-agent collaborative framework," arXiv preprint arXiv:2308.00352, 2023.
13. P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in Proc. NeurIPS, 2020, pp. 9459-9474.
14. S. Borgeaud et al., "Improving language models by retrieving from trillions of tokens," in Proc. ICML, 2022, pp. 2206-2240.
15. W. X. Zhao et al., "A survey of large language models," arXiv preprint arXiv:2303.18223, 2023.