Pragmatic Cybersecurity
https://www.sciltp.com/journals/pc

Article

Response-Level Identification of Cloud API Misconfigurations Using Large Language Models †

Akshay Krishna and Farzana Zahid *

School of Computing and Mathematical Sciences, University of Waikato, Hamilton 3216, New Zealand
* Correspondence: farzana.zahid@waikato.ac.nz

† Extended from a non-archival presentation titled: Response-Level Identification of Cloud API Misconfigurations Using Large Language Models. In Proceedings of the 31st Australasian Conference on Information Security and Privacy, Perth, WA, Australia, 6–9 July 2026.

How To Cite: Krishna, A.; Zahid, F. Response-Level Identification of Cloud API Misconfigurations Using Large Language Models. *Pragmatic Cybersecurity* **2026**, *1*(2), 12. https://doi.org/10.53941/pc.2026.100012

Received: 21 July 2026
Revised: 13 August 2026
Accepted: 14 Augutst 2026
Published: 20 Augutst 2026

Abstract: Application Programming Interfaces (APIs) are a set of rules that enable communication, data exchange, and automated interactions between applications and services. With the rapid advancement of cloud computing, APIs have evolved from simple data-access interfaces into critical components for managing, configuring, and orchestrating cloud resources. Most modern cloud platforms rely on RESTful APIs for provisioning cloud resources, applying configurations, and maintaining services. As a result, APIs misconfigurations have become a critical cloud security threat that can lead to sensitive data exposure, unauthorized access, or operational disruptions. Identifying these misconfigurations is challenging because traditional rule-based and static analysis methods often fail to capture complex, context-dependent configuration issues and system behaviors. In this study, we investigate the use of Large Language Models (LLMs) to detect security misconfigurations directly from cloud API response data. By treating API responses as representations of a system's configuration state, we assess whether LLMs can effectively identify potential security risks. We evaluate five LLMs using a unified zero-shot prompting approach and compare their performance with and without Retrieval-Augmented Generation (RAG) to understand the impact of external knowledge on misconfiguration detection. The study not only focuses on each model's ability to identify configuration components and detect misconfigurations, but also evaluates their capability to accurately determine the number of misconfigurations and generate clear, actionable security explanations. Our preliminary results show that Meta Llama Instruct combined with RAG achieves the reliable performance for identifying security misconfigurations in cloud API responses. This study provides new insights into the practicality of LLM-driven API cloud security analysis and paves the way for future research.

Keywords: cloud computing; API response; large language models (LLMs); misconfiguration; JSON

1. Introduction

Modern cloud computing platforms heavily rely on Application Programming Interfaces (APIs), in particular REST APIs, where communication between diverse applications is structured around standardized APIs requests and responses model [1,2]. Generally, a client application sends an API request to a server-side service in order to obtain an access or manage a specific resource. The server processes the request, performs the authentication and authorisation checks, and returns an API response to the client application in a specific text-based format, typically in JavaScript notation—JavaScript Object Notation (JSON), for further processing. This standardised communication model is the foundation for the scalability, automation, and interoperability of modern cloud environments.

In addition to the requested data, API response also provides metadata that includes configuration attributes,

Copyright: © 2026 by the authors. This is an open access article under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).
Publisher's Note: Scilight stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

permission bindings, identity associations, network settings, and policy enforcement details [3]. In many cases, the API response implicitly reveals whether security constraints are properly enforced. API responses, therefore, function as real-time representations of the effective configuration and security posture of cloud-based resources. While this design facilitates automation, it also introduces significant security risks in cloud environment. API responses in JSON is relatively easy to learn and to troubleshoot; if the resource is publicly accessible, overly permissive, or is using outdated permissions, this information can be revealed in the API response (even if it is not explicitly identified during the phase of API request), making it difficult to manage and secure cloud resources [2,4].

Existing approaches (Table 1) for detecting cloud misconfigurations are based on model-based API analysis, anomaly detection of IAM policies, static pre-deployment configuration checks, rule-based compliance checking, and template validation [5,6]. These approaches are effective in identifying known violations of predefined security benchmarks, but they often fail to detect contextual misconfigurations, and complex interactions between roles, permissions, and services [7]. The security audits are often time intensive and require skilled personnel; given the growth in both size and complexity of cloud environments, these approaches can lead to misconfigurations on the user end, compromising the cloud security. This raises a pressing research issue of developing an intelligent, and reliable mechanism capable of reasoning over cloud REST API responses to identify security misconfigurations beyond predefined rule sets [8].

Large Language Models (LLMs) have recently demonstrated strong capabilities in applying security-related analysis [9–11]. LLMs can understand the contextual relationships between the data within the configuration [9,12], and provide contextual explanations for the risks identified. LLM zero-shot reasoning ability enables them to interpret previously unseen configuration structures without task-specific fine-tuning. These characteristics make LLMs promising candidates for passive cloud security assessment based solely on API response data [9,13]. However, the use of LLMs in security-related contexts introduces additional challenges. Although LLMs may correctly identify misconfigurations, they may also generate hallucinated justifications (unsupported conclusions). Such unreliability poses significant risks because incorrect reasoning may overlook security issues or incorrectly classify benign configurations as security threats. This reliability gap highlights the need for structured evaluation frameworks and mechanisms that enhance trustworthiness in LLM-driven security analysis. Therefore, there is a need for a method that will provide a reliable and structured means of identifying configuration problems in a security context based on an analysis of cloud REST API responses while reducing the likelihood of producing incorrect interpretations.

It has been observed that several studies (Table 1) have explored the use of LLMs for securing cloud environment [14–19]. However, these studies largely focus on vulnerability descriptions, security documentation, or general threat analysis. Comparatively, despite the critical role of API responses in conveying configuration and maintaining security state of cloud resources, the use of LLMs to analyse cloud REST API responses as standalone artifacts is overlooked in the literature. As a result, we still lack a comprehensive understanding of LLM performance in cloud-focused cybersecurity applications. Furthermore, systematic comparative evaluations of multiple LLMs under controlled experimental settings remain limited, particularly with respect to assessing the reasoning capabilities of different LLMs.

Retrieval-Augmented Generation (RAG) has emerged as a promising technique to support the reasoning abilities of LLMs [20,21]. RAG allows LLMs to retrieve trusted security guidelines during analysis, reducing hallucinations and improving reliability in identifying API misconfigurations. However, none of the existing works investigate how LLMs interpret configuration information contained in API responses and reason about potential misconfigurations, both with and without RAG.

This paper addresses these research gaps by investigating whether LLMs can directly identify security misconfigurations from cloud REST API response data with a zero-shot prompting approach. The study adopts a passive evaluation methodology and ensures controlled experimentation without altering system state. To the best of our knowledge and based on our survey of existing literature (Table 1), this is the first study to systematically examine security weaknesses observable within the structure and content of API responses, rather than limiting analysis to vulnerabilities in API requests.

To achieve this aim, a dataset of 100 REST API JSON responses is constructed, consisting of both secure and intentionally misconfigured storage buckets and virtual instances derived from real-world and synthetic cloud environments. Multiple LLMs, such as Meta Llama 3 8B Instruct (with and without RAG) [22], ChatGPT 5.0 [23], and Gemini 2.5 [24], DeepSeek 3.1 [25] and OpenAI GPT OSS 20B [26] are evaluated under identical prompting conditions, both as standalone systems and in combination with RAG, to enable fair and systematic comparison. The evaluation framework examines detection capability, reasoning consistency, misconfiguration categorisation, and the impact of retrieval grounding on hallucination reduction. Thus, the *purpose of this work* is to empirically assess the feasibility of employing LLMs as configuration reasoning agents within cloud security workflows, rather than merely as generative language systems.

Table 1. Comparative analysis of existing studies using LLMs for cloud misconfiguration detection.

Refer	What They Did	Cloud Service	Resources Used	Accuracy Calculated	LLMs Used
[5]	Analysed cloud services REST APIs request, checked whether API interactions violated predefined security properties, used systematic exploration and formal checking of API behaviour	Microsoft Azure, Office 365 cloud service	Stateful REST API fuzzer, OpenAPI specifications	No	No
[6]	Studied AWS IAM policies, Used anomaly detection to identify misconfigured or risky permissions, focused on permission structure and usage patterns	AWS	AWS IAM policy, IAM policy datasets, AWS CLI, Neo4j	Precision, Recall, F1-score, ROCAUC	No
[14]	Synthesizes cutting-edge research and application of LLMs, FL, and RL/DRL for securing IoT and cloud environments	Generic IoT cloud ecosystems	Edge IIoT, IIoTsets, CIC IoT2023	No	No (theoret-ical discussion).
[15]	Conducted real-world experiments on AWS, GCP, Azure, IBM, and Oracle cloud platforms using LLMs to manage resource allocation and predict usage patterns using GPT 3	AWS, GCP, Microsoft Azure, IBM Cloud, Oracle Cloud	EC2 Instances (T2, M4, C5), Kaggle datasets	mean absolute error, Root mean square error	Used GPT-3 Architecture
[16]	Developed LLM Cloud Hunter to automatically extract core security indicators those candidates for SIEM integration	AWS	20 real-world cloud-related OSCTI reports, CloudTrail logs, sigma rules, Splunk	Precision, recall, F1, syntact-ic correctness of generated Sigma rules.	Used OpenAI GPT-4o (pretrained LLM).
[17]	Explore deploying Llama-2 on different AWS instances to understand speed vs cost vs resources.	AWS	SAGEMAKER service, EC2 instance	No	Llama-2 LLM
[18]	Developed SlsDetector, the first LLM-driven static analysis tool tailored for misconfiguration detection in serverless applications, focusing on AWS SAM configuration files	No provider mentioned	Cloud traffic logs, CICIDS2017, UNSW-NB15 datasets	precision, recall, F1-score	ChatGPT-4o and Gemini 1.5 Pro
[19]	Developed SARGE, a scalable system that leverages LLMs for zero-shot cloud misconfiguration detection across different providers, supporting both JSON and YAML configuration formats	AWS, Azure and GCP.	Real, synthetic configuration, Openstack misconfiguration scenarios	No	Gemini LLM
Our Work	Analysed JSON samples, used uniform zero shot prompting across all models, evaluated both LLMs and LLM with RAG.	Google Cloud Platform	REST API Responses in JSON format	Accuracy of Identifying Mis-configuration is calculated.	Meta Llama Instruct, ChatGPT 5.0, Gemini, Deepseek, OpenAI OSS

The *contributions of this work* are as follows:

- Introduction of LLM-based identification of cloud API misconfigurations at the response level (Section 2).
- Collection of API JSONs from real-time as well as synthetic data respectively from storage bucket and virtual instances (Section 3).
- Performed a comparative analysis of multiple LLMs (Section 3).
- Empirically evaluate the impact of RAG on detection reliability and hallucination reduction (Section 3).

2. Proposed Methodology

This section presents the proposed methodology (Figure 1) used to process and analyse cloud API responses with different LLM configurations, and to compare their effectiveness in detecting misconfigurations.

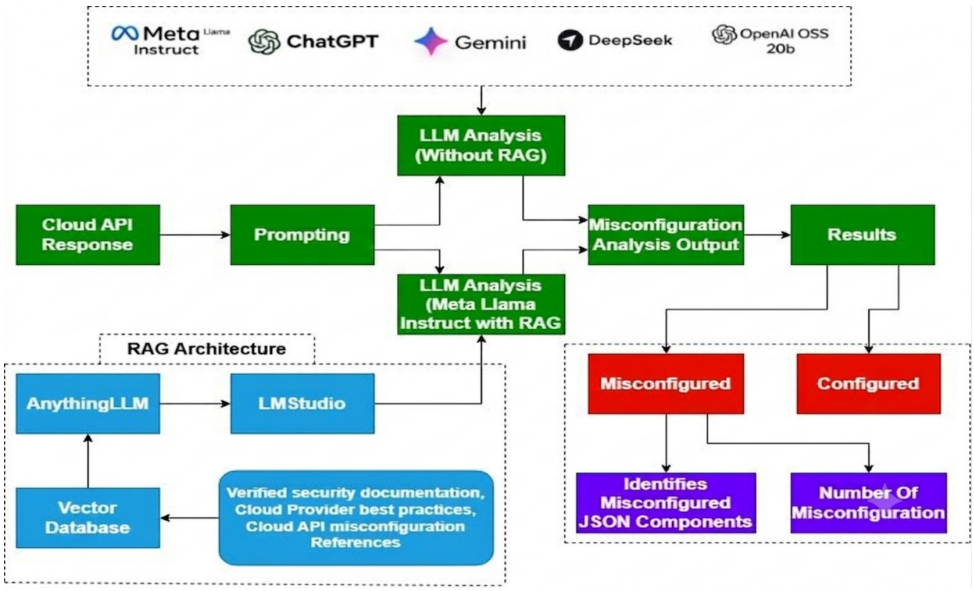


Figure 1. LLMs-based framework for identification of cloud API misconfigurations at response-level.

The proposed method establishes a structured approach for identifying cloud API misconfigurations by analysing the responses returned from REST APIs using LLM-based reasoning.

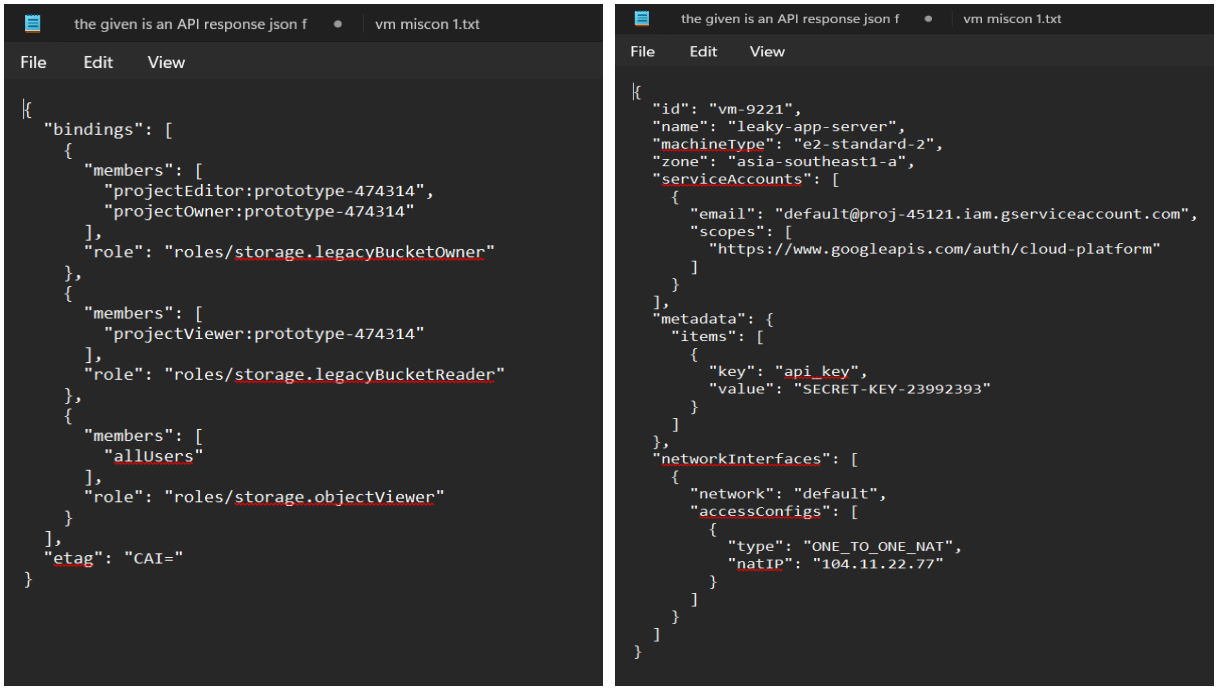
Using the reasoning capabilities of multiple LLMs, this research examines whether a given response from cloud services indicates a secure configuration and identifies and quantifies any potential misconfigurations that may introduce security risks.

2.1. Cloud API Response-Input Data

The process of identifying misconfigurations starts with the collection of cloud REST API responses in JSON format from deployed cloud environments. These responses include the actual runtime configuration and access control states of cloud resources, including permissions, roles, identity bindings, policy settings, and network attributes. In the proposed methodology, the collected cloud API responses are used as input data to LLMs for misconfiguration analysis. The structured JSON format enables the model to reason over configuration attributes, detect risky permission settings, and identify implicit security weaknesses. Figure 2 presents examples of API responses corresponding to a misconfigured storage bucket and a virtual instance.

Storage buckets are the primary containers used to store data objects in cloud environments. All data stored in Cloud Storage must reside within a bucket, and access to these objects is managed through permission and policy configurations [27]. The response, in Figure 2a, contains attributes such as bucket name, access control lists, IAM permissions, and visibility status. These attributes are critical because misconfigured access policies or public exposure configurations may significantly increase the risk of unauthorized access and data leakage.

Similarly, Google Cloud virtual instances are created through Google Compute Engine and represent virtual machines that execute applications and services on cloud infrastructure [13]. These instances provide configurable computing resources including CPUs, memory, storage, and network connectivity. The structured fields (Figure 2b) describe instance identity, machine type, zone, service accounts, metadata entries, and network interfaces. These attributes define both the operational characteristics and the security posture of the instance, and are considered critical elements in misconfiguration analysis.



(a) Misconfigured Storage Bucket **(b)** Misconfigured Virtual Machine

Figure 2. Sample API response of misconfigured storage bucket and virtual instance created in GCP.

2.2. Prompting

Prompting is utilised to guide the LLMs in evaluating the API responses, identifying potential misconfigurations, and providing justification for their results [28]. This prompting phase constitutes the second step of the proposed methodology and plays a critical role in evaluating the effectiveness and robustness of LLM-based misconfiguration detection.

In this study, zero-shot prompting was carefully designed to guide the LLM in evaluating cloud API responses to identify potential misconfigurations and justify its results. In this approach, the model receives no task-specific examples and relies solely on its internal reasoning [29]. Each LLM receives only the raw JSON API responses from the cloud to determine whether the configuration is secure or misconfigured, without prior indication of the correct label.

To ensure consistency across all models within the zero-shot setting, a standardised prompt is utilised. Although multiple prompt formulations were initially explored, the one with the best empirical performance was selected. The prompt does not provide any examples, labels, or task-specific guidance, but instead it guides the model to reason through the task using its existing knowledge and capabilities. The prompt first asks the model to identify the number of components present in each JSON response to ensure an understanding of the structure before further analysis. It then requests an assessment of whether the configuration is secure or misconfigured. If misconfigurations are identified, the number and their type are determined manually for validation purposes. The detailed prompts and results are discussed in Section 3.

2.3. LLM Analysis without RAG

Initially, the analysis is conducted using LLMs without integrating the RAG technique. In this case, the models are entirely dependent on their pre-trained knowledge to evaluate cloud API responses and can generalize well across different configurations; however, this setting also introduces limitations of incomplete interpretations, false conclusions, and unfounded assertions. Without access to up-to-date documentation or reference materials, the models may produce responses that are plausible but not necessarily accurate or consistent with verified security standards or documentation.

Moreover, cloud environments are dynamic in nature where APIs and security guidelines are evolving rapidly. Without RAG, LLMs cannot incorporate this evolving knowledge, which increases the likelihood of outdated or inconsistent assessments [20,21]. Hallucinated responses/outputs that seem correct but can reduce the reliability of the analysis and limit its practical applicability for security analysis or automated fixes [19]. In addition to this, complex configurations among multiple components, interdependent permissions, or misconfigurations that require

cross-referencing multiple sources can lead to misinterpretation without external context. Overall, LLMs without RAG can provide a fast, high-level evaluation, but their accuracy in detecting misconfigurations is uncertain.

2.4. LLM Analysis with RAG

To overcome the above-mentioned limitations, we focused on applying retrieval augmentation to the Meta Llama Instruct model. In this setup, the model is provided with external security information from trusted sources, which is retrieved during analysis. RAG helps address issues of incomplete or inaccurate interpretations by allowing the model to access relevant knowledge such as cloud security guidelines, official documentation, and best practice standards. By retrieving relevant and contextual information during evaluation, the model can make inferences based on verified references instead of solely relying on pre-trained knowledge.

In our study, this process is implemented using AnythingLLM [28] and LM Studio [30], which enable the model to access approved documentation and resources while analysing API responses. As a result, the model provides more accurate and reliable assessments of cloud configuration security, reducing the risk of false or misleading outputs. With RAG integration, the model can effectively perform analyses aligned with real-world standards and practices and handle complex configurations and interdependent settings. Overall, the integration of RAG with the Meta Llama Instruct model improves both the trustworthiness and practical applicability of LLM-driven security evaluations, making it a stronger tool for cloud API misconfiguration detection.

2.5. RAG Configuration

The RAG configuration describes the pipeline used for building the RAG module for Meta Llama Instruct. This section explains the role of vector database, AnythingLLM (v1.16.0), LM Studio (v0.4.21), and how they work together to provide the accurate and context-aware analysis of cloud API responses.

The Vector Database was created by collecting trusted cloud security documentation and best practice guidelines such as OWASP API security Risks [31], official documentation from Google Cloud Security [32], AWS Cloud security [33], Azure cloud security [34], NIST [35], CIS Cloud Platform benchmarks [36], document from NSA [8], CIS benchmarks for API Security [37] and some publicly available security configuration guidelines. Other resources, such as vendor-specific configuration recommendations, and threat intelligence reports, were also included to make sure that the database reflects up-to-date security practices. This documentation was uploaded to the knowledge module where these sources are segmented and stored as smaller sections. These smaller sections are called chunks in which the LLM can easily access according to the details provided from the user prompts. The segmentation also supports contextual prioritisation, and improves efficiency and accuracy during analysis by enabling the model to retrieve the most relevant sections first. These chunks are stored in a database called a Vector Database.

The vector database supports the semantic search of relevant information according to the content of the API response as well as the analysis. This part helps to efficiently match the configuration elements with their relevant security information. The database effectively acts as a dynamic knowledge base for RAG that can grow with the passage of time, allowing continuous updating as cloud security standards evolve.

AnythingLLM [28] acts as the retrieval interface that connects the vector database and language model. It is responsible for transforming documents into embeddings, managing search queries, and selecting the most relevant chunks for the model in real time. It also includes features to filter outdated or low-confidence sources, improving the trustworthiness of retrieved information. This makes the RAG-enabled Meta Llama Instruct model able to access external knowledge during analysis.

LM Studio [30] acts as a local host for the execution of the Meta Llama Instruct model. It allows seamless interaction with the language model, manages the retrieval pipeline, and provides a controlled environment for experimentation with and without RAG. LM Studio also provides logging and monitoring capabilities for retrieval operations, which enables evaluation of the RAG pipeline's effectiveness. In addition, it supports the integration of custom prompts, fine-tuning experiments, and performance benchmarking across different cloud configurations, making it a flexible platform for research and practical deployment.

2.6. Misconfiguration Analysis Output and Results Component

This phase presents the direct output produced by the LLMs, showing the model's response for each individual JSON sample after analysis. Each output reflects how the LLM interprets the configuration details and identifies potential misconfigurations based on the prompt. Please note that these outputs do not represent a cumulative or overall assessment of all API responses, but rather the analysis of single API response. The outputs consist of the detected misconfigurations, associated configuration components, and, where applicable, explanations or recommendations. Examining these outputs helps assess how effectively the LLMs identify security issues, explain

their decisions, and map configuration elements to relevant security standards. This approach also allows for a detailed comparison of model performance across different JSON samples, providing insights into the effectiveness, reliability, and limitations in handling diverse cloud configuration environments.

The aggregated results component then collects outputs from all analysed API responses to provide a comprehensive overview of models' performance. Two primary result measures are derived as shown in Figure 1: (i) identification of configuration components as secure or misconfigured, and (ii) if misconfigured then the number and type of misconfigurations. The latter is determined manually to ensure accuracy, as automated extraction can be prone to errors due to variations in API response structures, ambiguous outputs, or inconsistencies in model explanations. Manual validation allows precise classification of misconfigurations and ensures that both quantitative and qualitative analyses are reliable. Unlike prior studies, this work explicitly validates LLM outputs, as the literature to date has not addressed the verification of LLM-generated security assessments. The response from all the LLMs were taken by the common user prompt (See details in Section 3), ensuring consistency across all LLMs evaluated. By collecting and organizing these responses, the results component enables a systematic comparison of model performance, highlighting differences in accuracy, completeness, and reasoning across the evaluated LLMs.

Overall, this aggregated view supports both quantitative and qualitative analysis. Quantitatively, it allows measurement of metrics such as detection accuracy, and false rates. Qualitatively, it provides insight into the models' reasoning, explanation quality, and ability to link misconfigurations to relevant security standards. This dual approach ensures a thorough understanding of each model's strengths and limitations and highlights areas where retrieval augmentation or other enhancements can improve performance. Overall, the results component not only serves as the foundation for model comparison but also informs practical evaluation of LLM-driven cloud API security analysis, demonstrating how individual outputs contribute to broader insights about model reliability and applicability in real-world scenarios.

3. Experiments and Results

This section discusses the details of the experiments with the selected prompts, the LLMs' outputs, and the analysis methodology. The experiment was conducted using five different LLM models, including Meta Llama Instruct, ChatGPT, OpenAI OSS, Gemini, and DeepSeek, with each model evaluated on the same set of REST API response samples. The experiment was structured so that the prompting strategy remained identical across all models to ensure fairness and consistency in evaluation.

In the case of the retrieval-augmented configuration, Meta Llama Instruct was deployed locally using LM Studio as the runtime environment, while AnythingLLM acted as the orchestration mechanism to facilitate the RAG. A vector database was created and populated with verified security documents (Section 2). During the analysis, relevant information was retrieved from the vector database and provided to the model together with the API response. Please note that RAG was not used with models other than Meta Llama due to open-source availability and cost constraints. For the preliminary experiments, we considered both configured and misconfigured API responses from storage buckets and virtual instances resources, ensuring balanced evaluation. Quantitative results are presented with tables for easy model-to-model comparison, and qualitative notes bring attention to the differences in misconfigurations and the clarity of the models' decision making.

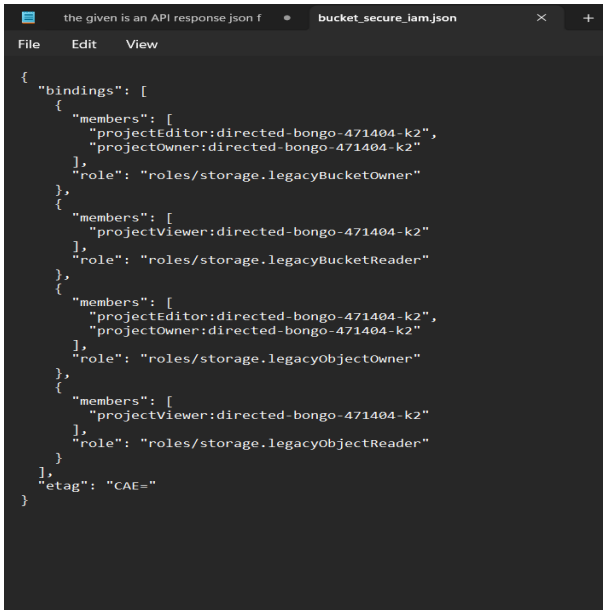
3.1. JSON API Responses and Prompting Strategy

For this study, 100 JSON samples were selected, consisting of 51 configured and 49 misconfigured responses. Among them, 59 samples correspond to storage buckets and 41 correspond to virtual machine instances. The samples were divided into two sets (JSON 1–50 and JSON 51–100) for all the evaluation to maintain structured experimentation. As an example, Figure 3a,b illustrates the sample REST API responses retrieved from the secured Google Cloud storage bucket and virtual instance, respectively.

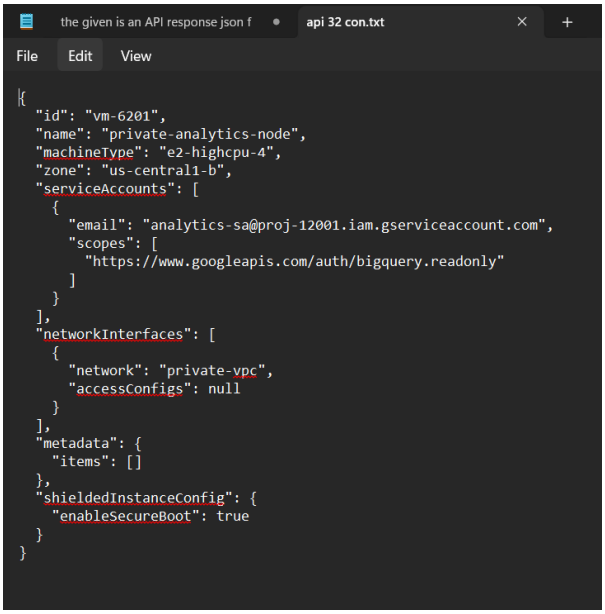
Similarly, Figure 3c,d presents the JSON representations of misconfigured cloud resources, where excessive permissions or public access settings are present. Both resources include metadata related to service accounts, network interfaces, firewall settings, and access permissions.

For each API response, a consistent analysis process was applied. First, the LLM was prompted to identify the individual configuration components present within the JSON response. After that, each LLM was asked to verify whether any components were misconfigured. Finally, if misconfigurations were identified, the LLM was required to explicitly describe the nature of those misconfigurations. A misconfiguration occurs when there is excessive access, leakage of information, or compromised access control, depending on what is contained in the API response. For each response, we tracked which configuration elements were identified as misconfigured by the model. In cases where misconfigurations were detected, the output of the model was further evaluated based on how accurately it

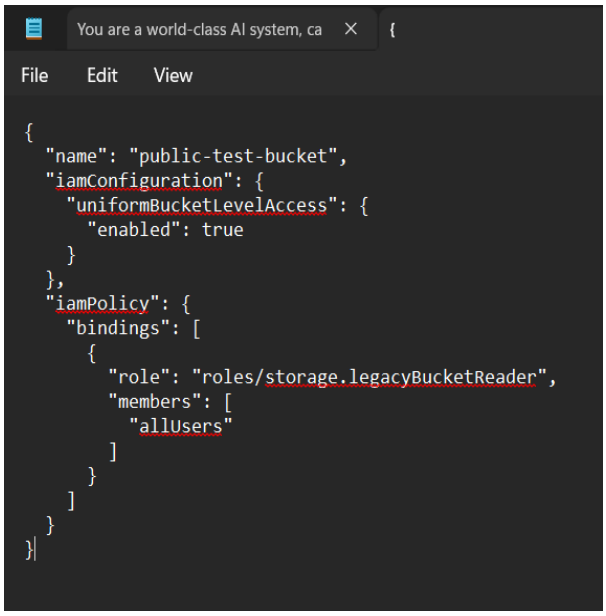
identified the number of misconfigurations and how well it explained their nature. The explanations generated were then used to carry out qualitative analysis and cross-model comparison. The quantitative summaries were derived from the evaluation results, while detailed per-sample findings and misconfiguration records were documented. This approach enabled systematic comparison across models and ensured traceability between each API response and its corresponding analysis outcome.



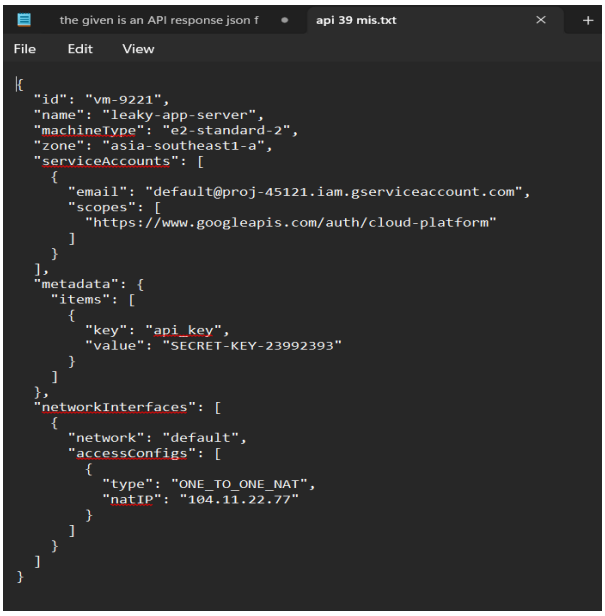
(a) Configured storage bucket JSON



(b) Configured VM JSON



(c) Misconfigured storage bucket JSON



(d) Misconfigured VM JSON

Figure 3. Configured and misconfigured JSON from storage bucket and virtual instance in GCP.

The prompting approach consisted of two components: a system prompt and a user prompt, each serving a distinct role in the API response analysis process. The system prompt was used only within AnythingLLM for the Meta Llama Instruct model with RAG. The prompt is:

> "*You are a world class AI system, capable of complex reasoning and reflection. You are given a cloud REST API response in JSON format that describe the configuration state of cloud resource. Analyze the response strictly based on the information provided in vector chunks here content. Do not assume missing information and do not invent configuration details.*"

We explored multiple prompt formulations, and the final design was selected to minimise hallucinations and

ensure that outputs were grounded solely in the API response content. We did not introduce any task-specific examples or labels to maintain the strict zero-shot setting. The Cloud REST API response was provided along with specific and structured instructions embedded in the user prompt. The user prompt specified the analysis steps to be carried out by the model. The model had to identify all security-related configuration components in the API response and determine whether each of the identified components was configured securely or misconfigured. The model had to communicate the identified components that were in a state of misconfiguration along with the corresponding rationale. Furthermore, the user prompt specified the structure of the output, which was intended to aid in comparing the results of all the models.

3.2. Experimental Results-Analysis of Model Performance on Cloud API Responses

Identifying configuration elements is a fundamental step in the analysis process, as incomplete identification of components can lead to inaccurate detection of configuration errors.

To assess whether these components are configured or misconfigured, as an illustration, the JSON in Figure 4 was provided to Llama with RAG. Please note: due to page limit, we report results only for Meta Llama 3 8B Instruct, both with and without RAG.


```
{
 "name": "public-test-bucket",
 "iamConfiguration": {
  "uniformBucketLevelAccess": {
   "enabled": true
  }
 },
 "iamPolicy": {
  "bindings": [
   {
    "role": "roles/storage.legacyBucketReader",
    "members": [
     "allUsers"
    ]
   }
  ]
 }
}
```


Figure 4. A misconfigured API response from storage bucket used for analysis.

The response generated by *Llama with RAG* is presented below:

```
("<thinking>

To identify the number of components and potential misconfigurations in the given
    JSON, I will analyze each part of the configuration.

1. **Name**: This is a string representing the name of the cloud resource (e.g.,
    "public-test-bucket").
2. **iamConfiguration**: This object contains settings related to IAM (Identity and
    Access Management) for the bucket.
3. **iamPolicy**: This object contains a policy document for the bucket, which
    defines access controls and permissions.

Based on this analysis, I have identified the following components:

1. Name
2. iamConfiguration
3. uniformBucketLevelAccess
Now, let's examine each component for potential misconfigurations.

**Misconfigured Components:**
1. **iamPolicy**: The binding associated with
    `roles/storage.legacyBucketReader`grants access to "allUsers", which is a
    public identifier that allows anyone to access the bucket. This might be
    intended for a publicly accessible resource, but it's essential to review and
    ensure that such sensitive resources are properly secured.

**Potential Misconfiguration:**
```

```
1. **Overly permissive access**: The binding associated with

`roles/storage.legacyBucketReader` grants access to "allUsers", which might allow
    unauthorized users to access the bucket.

</thinking>
<output>
There are 3 main components in the JSON and one of them is misconfigured:

* iamPolicy: The binding associated with roles/storage.legacyBucketReader grants
    access to "allUsers", which is a public identifier that allows anyone to access
    the
bucket. %)
```

It has been observed that Meta Llama Instruct with RAG has successfully analysed the JSON with. The prompt doesn't state that the given JSON is misconfigured, but the LLM correctly identified the misconfigured component and mentioned what the misconfiguration is. The model systematically decomposes the JSON, evaluates each component, and correctly identifies the IAM policy as the source of the issue, specifically the assignment of roles/storage.legacy BucketReader to allUsers, which results in overly permissive access. With the integration of external security knowledge through RAG, the assessment becomes more consistent and well-justified. The model cites retrieved sources (Figure 5) and lists references from the vector database, improving transparency and explainability in the security analysis.

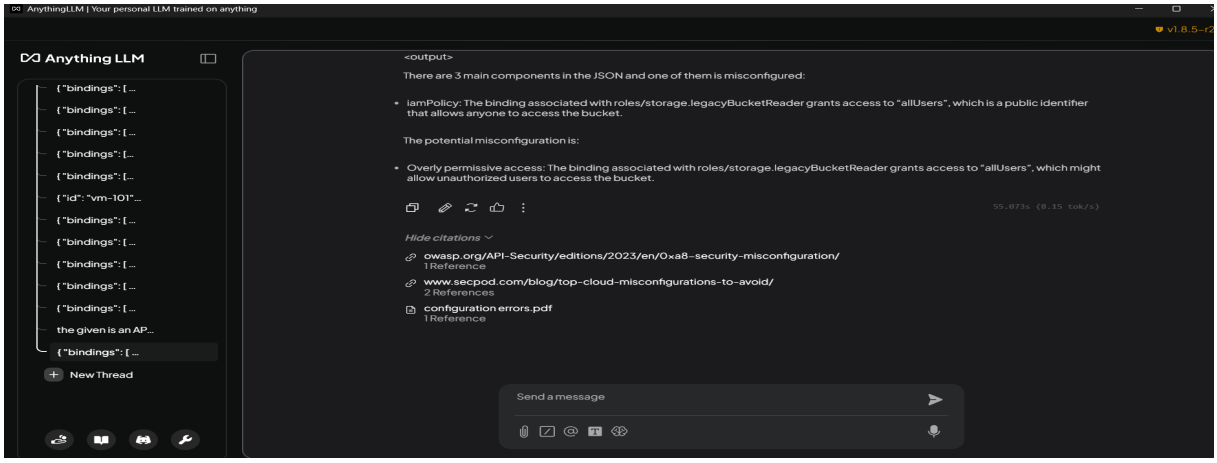


Figure 5. RAG output.

On the other hand, the Meta Llama Instruct, without the support of retrieval augmentation, misinterprets the structure of the JSON and identifies the missing members field as the main misconfiguration as shown below in the JSON analysis. However, this field is already present in the original response. It therefore fails to identify the main security risk with the response being publicly available. This response shows the increased likelihood of structural misinterpretation.

```
### JSON Analysis
The given JSON response contains **4 components:**

1. `name`
2. `iamConfiguration`
3. `iamPolicy`
4. `bindings`

### Misconfiguration Details

Upon analyzing the JSON response, it appears that there is a **misconfiguration**
    in the `bindings` component.

**Misconfiguration:** missing `members` field within the `bindings` component
```

```
The `bindings` component contains a nested `bindings` object, but the `members`
    field is missing. This is considered a misconfiguration.

The `bindings` component should have a nested `members` field, like this:

```json
"bindings": [
  {
    "role": "roles/storage.legacyBucketReader",
    "members": [
      "allUsers"
    ]
  }
]
```
This will ensure that the correct `members` field is present in the `bindings`
    component.
```

The answer provided by *ChatGPT* accurately recognises the implication of allowing allUsers to have access to the bucket, which would allow all users to access the bucket. However, the answer does not provide a lot of structural analysis or guidance on mitigating the situation. *Gemini* evaluates the resource structure, emphasizing the significance of uniform access to buckets and the relationship to IAM policy management. Rather than focusing on the direct issue of public exposure, the model emphasizes the possibility of inconsistencies in configuration deployment and practices. It also provides recommendations related to permission management within the organization. Similarly, the *OpenAI OSS* model provides the JSON components in a structured manner and correctly identifies the public access binding as the main misconfiguration issue. It correctly identifies the legacy role and suggests replacing it with a more restricted permission. In addition, it provides a corrected configuration example along with remediation steps. However, minor syntax errors not present in the input indicate some degree of hallucination. The process followed by *DeepSeek* is methodical, where it analyses each component of the JSON in a step-by-step fashion. It checks whether uniform bucket-level access is a safe configuration, confirming the main source of risk to be the IAM policy. It also correctly identifies the use of allUsers to allow public access, along with the possible implications for data security.

3.3. Discussion

All the models were able to identify all configuration components in the 100 API response sample set, namely resources, access roles, permissions, and policy bindings. However, there were some differences in the number of items identified by the models and the level of consistency in identifying the items in the response samples. Of the models tested, ChatGPT, Meta Llama 3 8B Instruct with RAG, DeepSeek, and OpenAI OSS were able to correctly identify the misconfiguration in the public access issue by recognizing the assignment to allUsers. DeepSeek and Llama with RAG were the best in providing consistent reasoning, where the configuration elements were clearly related to security implications. Gemini, on the other hand, focused more on the governance and deployment aspects, while Llama without RAG was not able to identify the principal exposure risk.

Regarding the number of misconfigurations identified, all models were able to detect explicit misconfiguration, such as publicly accessible storage buckets or overly permissive role assignments. However, when evaluated against samples including diverse configuration types and increasing levels of complexity, performance differences of models became more evident. While all models handled straightforward cases reliably, their effectiveness varied when analysing more complex and less explicit configurations, highlighting differences in reasoning depth and contextual understanding. As an illustration, first 15 samples have been shown in Figure 6.

In terms of configuration explanations, we observed variation in the level of detail provided by the models. Some models presented brief and precise explanations, while others produced longer responses that did not always clearly link the misconfiguration to its potential security impact. Models that were more context-aware were able to justify why a misconfiguration violated fundamental security principles, such as least privilege or minimal public exposure. The details of misconfigurations identified by all models for each JSON, including the first 10 samples, are shown in Table 2.

Table 2. Details of identified misconfiguration—First 10 JSON samples.

	Llama with RAG	**Llama without RAG**	**ChatGPT**	**Gemini**	**Deepseek**	**OpenAI OSS**
JSON1	Alluser grants access to read object data	Role Inconsistency, Role Mismatch	Alluser grants access to read object data	Secured	Alluser grants access to read object data, overlapping legacy roles	Public read access to every object
JSON2	Allauthenticated user gives access to anyone, the condition given will always be true	Condition Expression Is Always True, Role Assignment Might Not Be Restrictive Enough	Overly permissive access to allAuthenticatedUsers	Grants read access to all Google-authenticated users, Condition is labeled ”temporary” but is valid until year 2100.	The combination of effectively permanent access to AllAuthenticatedUsers	Public read for all authenticated users, a non-expiring condition
JSON3	Configured	Configured	Configured	Overlapping the Legacy roles	Overlapping the Legacy roles	Over-permissive logging service account
JSON4	Alluser makes public data exposure	Alluser makes public data exposure, Potential Duplicate Permissions, Role Inconsistency	Alluser makes public data exposure	Alluser makes public data exposure	Alluser makes public data exposure	Alluser makes public data exposure
JSON5	Alluser makes public data exposure, overlapping IAM roles	Alluser makes public data exposure, overlapping IAM roles	Alluser makes public data exposure	Alluser makes public data exposure	Alluser makes public data exposure, overlapping IAM roles	Alluser makes public data exposure
JSON6	Secured	Overpermissive IAM roles, duplicate roles	Secured	Overpermissive IAM roles	Overpermissive IAM roles	Secured
JSON7	Alluser makes high risk public access, overpermissive IAM roles	Alluser makes high risk public access, Project Owner not Assigned with Legacy Bucket Owner Role	Alluser makes high risk public access	Alluser makes high risk public access	Alluser makes high risk public access, overpermissive IAM roles	Public Object Read (allUser)
JSON8	All IP protocols allowed, All IP addresses are allowed, firewall direction is set to INGRESS	All IP protocols allowed, All IP addresses are allowed, firewall direction is set to INGRESS	Unrestricted Ingress from Anywhere, Allow All Protocols, Logging Disabled, Default Network Usage	Firewall direction is set to INGRESS	Firewall direction is set to INGRESS, Allow all protocols, Allow all IP addresses, rule is active and enforced, priority set to 1000	Open Ingress Rule Allowing All Traffic
JSON9	Secured	Duplicated roles	Secured	Doesn’t include modern IAM Practices	Overlapping Permission	Use Of Legacy IAMroles
JSON10	Configured	Configured	Configured	Configured	Configured	Configured

	Llama with RAG	Llama no Rag	ChatGPT	Gemini	Deepseek	OpenAI OSS	Actual Misconfiguration
Json1	1	2	1	0	2	1	1
Json2	2	2	1	2	1	2	2
Json3	0	0	0	1	1	1	0
Json4	1	3	1	1	1	1	1
Json5	2	2	1	1	2	1	1
Json6	0	2	0	1	1	0	0
Json7	2	2	1	1	2	1	1
Json8	3	3	4	1	5	2	3
Json9	0	1	0	1	1	1	0
Json10	0	0	0	0	0	0	0
Json11	0	0	0	0	0	0	0
Json12	0	0	0	0	0	1	0
Json13	0	0	0	0	0	0	0
Json14	0	0	0	0	0	0	0
Json15	0	1	0	1	1	2	0

Figure 6. Set 1: Number of misconfigurations identified in first 15 JSONs.

Misconfiguration accuracy was calculated manually using the standard formula (1) based on true positives, false positives and false negatives and information based on number of misconfiguration identified (Actual Misconfiguration in column in Figure 6). The accuracy for each LLM in identifying the misconfigurations is shown in the Table 3.

$$\text{Accuracy (AC)} = TP * 100/(TP + FP + FN) \quad (1)$$

Meta Llama 3 (8B Instruct) with RAG achieved the highest accuracy at 92.18% (TP = 118, FP = 5, FN = 5), demonstrating the benefit of incorporating RAG. Among models without RAG, ChatGPT 5.0 performed best at 89.23% (TP = 116, FP = 7, FN = 7), followed by Gemini 2.5 at 87.12% (TP = 115, FP = 9, FN = 8). Meta Llama without RAG and OpenAI GPT OSS 20B reached 83.72%, while DeepSeek 3.1 had the lowest accuracy at 82.48%. Overall, the integration of RAG has improved the model's ability to correctly identify misconfigurations and reduce false rates.

Overall, for our study, the ground-truth misconfigurations were defined using established cloud security best practices and configuration rules and were verified during dataset creation. To reduce bias, the annotations and evaluation results were reviewed by multiple researchers using predefined criteria and consistency checks. The prompts were also designed to provide task instructions and did not include labeled examples or feedback. Therefore, they were not intended to introduce implicit supervision.

Table 3. Comparative analysis of Meta Llama Instruct with RAG and other LLMs.

Llama with RAG	Llama no RAG	ChatGPT	Gemini	Deepseek	OpenAI OSS
TP = 118	TP = 108	TP = 116	TP = 115	TP = 113	TP = 108
FP = 5	FP = 11	FP = 7	FP = 9	FP = 14	FP = 6
FN = 5	FN = 15	FN = 7	FN = 8	FN = 10	FN = 15
Acc = 92.18%	Acc = 83.72%	Acc = 89.23%	Acc = 87.12%	Acc = 82.48%	Acc = 83.72%

3.4. Comparative Analysis with Relevant Existing Study

In addition to the comparative analysis of our work with existing tools (Table 3), we have also performed the analysis with the most relevant study in the literature. This research paper [19] proposes a system (SARGE) that integrates Google Gemini, RAG, and agent-based modeling to analyze cloud configuration files such as Terraform, JSON, and YAML templates. Moreover, the study [19] used pre-deployment configuration files for analysis, while the our study relies only on the evaluation of the model's reasoning capability on live responses from the REST API without using configuration templates or datasets for evaluation or fine-tuning the model. The depth of evaluation also differs significantly between both studies. In this research [19], proof of concept validation is conducted on a limited set of configuration scenarios, whereas in the current research, extensive comparative experimentation is conducted on 100 JSON API responses with multiple LLMs. In addition, quantitative measures, including True Positives, False Positives, False Negatives, and RAG vs. non-RAG, are evaluated in detail.

3.5. Limitations of Our Preliminary Study

As in this study, RAG was applied only to Meta Llama because of its open-source availability, the observed performance differences cannot be only attributed to either the model architecture or the RAG framework itself. In

the future, we are planning to apply RAG to other comparable models to provide more comprehensive evaluation.

Another limitation of this study is the relatively small dataset size, consisting of 100 JSON samples, including synthetically generated data. Due to the lack of publicly available datasets, the dataset was created to support controlled experimentation; however, it may not fully capture the complexity, diversity, and scale of real-world cloud environments. Consequently, the generalizability of the findings may be limited. Future work should focus on evaluating and validating the proposed approach using larger and more diverse real-world datasets.

Furthermore, the evaluation in this study focuses primarily on accuracy-related metrics and does not include a comprehensive analysis of robustness, such as the system's response to adversarial inputs or malformed API responses, as these aspects were outside the scope of this paper. Future research should include comprehensive robustness assessments to better understand the system's reliability in challenging and dynamic real-world environments.

4. Conclusions and Future Directions

The goal of this research was to determine whether LLMs are able to identify security misconfigurations directly from the responses of a cloud-based RESTful API, rather than examining the requests, logs, or policies themselves. In this study, we proposed and implemented an architecture which supports the structured prompting, multi-model accuracy comparison and the analysis of reasoning ability when implemented with RAG. It was found that the Meta Llama Instruct with retrieval augmentation outperformed all other LLMs, indicating the value of reasoning based on external sources of security information to produce consistent reasoning about complex cloud-based configurations.

From a practical standpoint, the results of the current study reinforce the possibility of using LLMs to help inform cloud security governance and auditing. Rather than replacing current security practices, LLMs can be used as a secondary level of evaluation to help security professionals better identify areas of potential risk. To begin with, the focus of the study was limited to storage bucket and virtual machine APIs. While these services are critical components of the cloud infrastructure, the current cloud computing systems provide access to a range of services, including identity and access management systems, container orchestration services, network services, and application environments. Secondly, the lack of publicly accessible datasets specifically designed for the analysis of cloud API responses led to the utilization of artificially generated configurations in the current study. Although artificial data allows researchers to conduct experiments, it may not reflect the full range of variations and complexities observed in real-world cloud infrastructures. The creation and availability of publicly accessible benchmark datasets for the analysis of cloud API responses could facilitate a deeper evaluation and facilitate comparative analysis across different studies. Another important area of research is scalability. As the infrastructure of the cloud increases in size and complexity, a considerable volume of API response data may have to be analyzed in near-real-time. Therefore, future research should focus on the implications of using a large language model-based approach to API response data analysis, including the associated cost and resource management implications.

Author Contributions

A.K.: Conceptualization, methodology, data curation, investigation, software, visualization, writing—original draft preparation, validation; F.Z.: Conceptualization, methodology, supervision, validation, writing—reviewing and editing. All authors have read and agreed to the published version of the manuscript.

Funding

This research received no external funding.

Institutional Review Board Statement

Not applicable.

Informed Consent Statement

Not applicable.

Data Availability Statement

Not applicable.

Conflicts of Interest

The authors declare no conflict of interest.

Use of AI and AI-Assisted Technologies

During the preparation of this work, the authors used Microsoft Copilot mainly for grammar correction and occasionally for improving readability. However, the content, ideas, and conclusions presented are entirely the work of the authors, reflecting their original research and insights.

References

1. Google Cloud APIs. Available online: https://docs.cloud.google.com/apis/docs/overview (accessed on 15 September 2025).
2. Patil, G. Introduction to APIs. In *Django REST APIs Demystified: Simplifying API Development with Django*; Apress: Berkeley, CA, USA, 2025; pp. 1–4.
3. Powell, P.; Smalley, I. What Is a REST API (RESTful API)? Available online: https://www.ibm.com/think/topics/rest-apis (accessed on 24 August 2025).
4. Erickson, J. What Is JSON? Available online: https://www.oracle.com/nz/database/what-is-json/ (accessed on 12 October 2025).
5. Atlidakis, V.; Godefroid, P.; Polishchuk, M. Checking Security Properties of Cloud Service REST APIs. In Proceedings of the 2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST), Porto, Portugal, 24–28 October 2020; pp. 387–397. https://doi.org/10.1109/icst46399.2020.00046.
6. van Ede, T.; Khasuntsev, N.; Steen, B.; et al. Detecting Anomalous Misconfigurations in AWS Identity and Access Management Policies. In Proceedings of the 2022 on Cloud Computing Security Workshop, Los Angeles, CA, USA, 7 November 2022; pp. 63–74.
7. Xu, W.; Huang, L.; Fox, A.; et al. Detecting Large-Scale System Problems by Mining Console Logs. In Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles, Big Sky, MT, USA, 11–14 October 2009; pp. 117–132.
8. National Security Agency. Available online: https://media.defense.gov/2020/Jan/22/2002237484/-1/-1/0/CSI-MITIGATING-CLOUD-VULNERABILITIES_20200121.PDF (accessed on 1 January 2026).
9. Xu, H.; Wang, S.; Li, N.; et al. Large Language Models for Cyber Security: A Systematic Literature Review. *ACM Trans. Softw. Eng. Methodol.* **2025**. https://doi.org/10.1145/3769676.
10. Hasanov, I.; Virtanen, S.; Hakkala, A. Application of Large Language Models in Cybersecurity: A Systematic Literature Review. *IEEE Access* **2024**, *12*, 176751–176778. https://doi.org/10.1109/access.2024.3505983.
11. Raiaan, M.A.K.; Mukta, M.S.H.; Fatema, K.; et al. A Review on Large Language Models: Architectures, Applications, Taxonomies, Open Issues and Challenges. *IEEE Access* **2024**, *12*, 26839–26874. https://doi.org/10.1109/access.2024.3365742.
12. Palo Alto Networks. What Is LLM (Large Language Model) Security? Available online: https://www.paloaltonetworks.com/cyberpedia/what-is-llm-security (accessed on 17 September 2025).
13. Bergeman, D. What Is Zero-Shot Learning? Available online: https://www.ibm.com/think/topics/zero-shot-learning (accessed on 31 August 2025).
14. Ghanem, M.C. Advancing IoT and Cloud Security through LLMs, Federated Learning, and Reinforcement Learning. In Proceedings of the 7th IEEE Conference on Cloud and Internet of Things, Montreal, QC, Canada, 29–31 October 2024; pp. 1–27.
15. Li, H.; Wang, S.X.; Shang, F.; et al. Applications of Large Language Models in Cloud Computing: An Empirical Study Using Real-World Data. *Int. J. Innov. Res. Comput. Sci. Technol.* **2024**, *12*, 59–69. https://doi.org/10.55524/ijircst.2024.12.4.10.
16. Schwartz, Y.; Ben-Shimol, L.; Mimran, D.; et al. LLMCloudHunter: Harnessing LLMs for Automated Extraction of Detection Rules from Cloud-Based CTI. In Proceedings of the ACM Web Conference 2025, Sydney, NSW, Australia, 28 April–2 May 2025; pp. 1922–1941.
17. Kodali, R.K.; Upreti, Y.P.; Boppana, L. Large Language Models in AWS. In Proceedings of the 2024 1st International Conference on Robotics, Engineering, Science, and Technology (RESTCON), Pattaya, Thailand, 16–18 February 2024; pp. 112–117. https://doi.org/10.1109/restcon60981.2024.10463557.
18. Wen, J.; Chen, Z.; Zhu, Z.; et al. LLM-Based Misconfiguration Detection for AWS Serverless Computing. *ACM Trans. Softw. Eng. Methodol.* **2026**, *35*, 1–28. https://doi.org/10.1145/3745766.
19. Goyal, M.K.; Chaturvedi, R. Detecting Cloud Misconfigurations with RAG and Intelligent Agents: A Natural Language Understanding Approach. *J. Electr. Syst.* **2025**, *20*, 2558–2570. https://doi.org/10.52783/jes.7882.
20. Lewis, P.; Perez, E.; Piktus, A.; et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In Proceedings of the Advances in Neural Information Processing Systems 33 (NeurIPS 2020), Virtual Event, 6–12 December 2020; Volume 33, pp. 9459–9474. https://doi.org/10.48550/arxiv.2005.11401.
21. Delgado, A.; Saitis, C.; Benetos, E.; et al. Deep Conditional Representation Learning for Drum Sample Retrieval by Vocalisation. *arXiv* **2022**, arXiv:2204.04651.
22. Meta Platforms, Inc. Meta LLaMA Documentation. Available online: https://www.llama.com/docs/overview/ (accessed on 30 August 2025).
23. OpenAI Platform Documentation. Available online: https://platform.openai.com/docs/overview (accessed on 31 August 2025).

24. Gemini API & Model Docs. Available online: https://ai.google.dev/gemini-api/docs (accessed on 31 August 2025).
25. DeepSeek. DeepSeek API Documentation. Available online: https://api-docs.deepseek.com/ (accessed on 30 August 2025).
26. OpenAI Model & Deployment Documentation. Available online: https://platform.openai.com/docs/models/gpt-oss-20b (accessed on 25 August 2025).
27. About Cloud Storage Buckets. Available online: https://docs.cloud.google.com/storage/docs/buckets (accessed on 1 January 2026).
28. AnythingLLM Documentation. Available online: https://docs.anythingllm.com/ (accessed on 25 August 2025).
29. Gadesha, V. Prompt Engineering Techniques. Available online: https://www.ibm.com/think/topics/prompt-engineering-techniques (accessed on 21 October 2025).
30. LM Studio Documentation. Available online: https://lmstudio.ai/docs/app (accessed on 25 August 2025).
31. OWASP Top 10 API Security Risks—2023. Available online: https://owasp.org/API-Security/editions/2023/en/0x11-t10/ (accessed on 2 September 2025).
32. Google Cloud. Google Cloud Security Best Practices Center. Available online: https://cloud.google.com/security/best-practices (accessed on 9 August 2025).
33. Amazon Web Services. AWS Security Documentation. Available online: https://docs.aws.amazon.com/security/ (accessed on 9 August 2025).
34. Azure Security Documentation. Available online: https://learn.microsoft.com/en-us/azure/security/ (accessed on 9 August 2025).
35. Chandramouli, R.; Butcher, Z. *Guidelines for API Protection for Cloud-Native Systems*; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2025. https://doi.org/10.6028/nist.sp.800-228.
36. CIS Google Cloud Platform Foundation Benchmark. Available online: https://www.cisecurity.org/insights/blog/cis-benchmarks-april-2024-update#CISGoogleCloudPlatformFoundationBenchmarkv3.0.0 (accessed on 8 August 2025).
37. CIS API Security Guide. Available online: https://learn.cisecurity.org/cis-api-security-guide (accessed on 9 August 2026).