Future Generation of Communication and Internet of Things (FGCIOT)

Journal homepage: http://fgciot.semnaniau.ac.ir/

PAPER TYPE (Research paper)

Analysis of Monolithic and Microservice Architectures Using Client-Side Inference and Statistical Modeling: A Case Study Approach

Hamidreza.Naseri*[1], Hoda Avazzadeh[1], Mehdi Ghasemi[2]

[1]*Department of Computer Engineering, BA.C., Islamic Azad University, Bandar Abbas, Iran.*

[2]*Department of Information Technology, Ministry of Educational System, Bandar Abbas, Iran*

Article Info

Article History:

Received 20 February 2025

Revised 22 March 2025

Accepted 25 May 2025

Keywords:

Client-Side Threat Analysis , Microservices Security, System Architecture Inference

*Corresponding Author's Email Address: Dr.Hamidreza.Naseri@gmail.com

Extended Abstract

In the evolving context of web-based system deployment, software architectural design—monolithic or microservice—is the deciding factor for its security posture. This paper presents a novel client-side solution for system architecture inference and evaluation of performance-related threats through browser-level resource monitoring and Gaussian Mixture Model (GMM) clustering of response times. By analyzing actual systems such as Digikala and Jonoob Iran, the work discovers unequivocal signs of architecture: high domain heterogeneity and modular latency clusters in microservices, and centralized domain usage with persistent latency in monolithic systems. The work not only confirms theoretical differences but also presents a lightweight, non-intrusive diagnostics infrastructure for architecturally classifying systems and detecting anomalies, with broad red teaming, DevOps monitoring, and security auditing implications. The approach enhances theoretical as well as practical solutions to architecture-sensitive threat analysis in access-controlled environments.

Introduction

In today's digitally connected and cloud-focused era, software system security has become a critical concern for developers, system architects, and security experts. As businesses increasingly rely on web-based services and deploy systems at scale, the underlying architectural model has a direct influence on the functionality and vulnerability of these systems. Two of the leading paradigms—monolithic architectures and microservices-based architectures—each offer alternative design philosophies with differing security implications. While monolithic systems are traditionally implemented as a single, tight-coupling unit, microservices divide functions among independently deployable and loosely coupled services. Both architectures possess unique security challenges and opportunities that must be well understood within the settings of real-world operational environments [1,2].

Monolithic architectures, although with an easier deployment and operation, are typically vulnerable due to their centralized setup. A successful attack on a single building block can reveal the entire structure. Microservices, however, enhance modularity, scalability, and maintainability but introduce a much-increased attack surface in that they depend on distributed services and inter-service communications [3,4]. Security controls in microservice environments must contend with network-level attacks, service authentication and authorization, and secure communication protocols between services—problems often less acute in monolithic setups [5,6].

The trend towards containerized microservices, cloud-native platforms, and serverless architectures complicates the security picture even further. Attackers can attack insecure APIs, misconfigured service meshes, or privilege-escalation vectors in orchestrated environments like Kubernetes or Docker [7,8]. Furthermore, the dynamic and decentralized nature of microservices makes perimeter-based security models ineffective, and thus, there is a move toward zero-trust

Doi:

frameworks [9]. In turn, monolithic systems remain susceptible to side-channel attacks, memory data leaks, and legacy bugs, especially if not refactored or actively maintained [10,11].

The main research issue of this study comes from the need to find out how architectural choices influence system vulnerabilities and threat spreading. Despite more recent work on microservices security[12,13], little attention has been directed toward comparing monolithic and microservice threat patterns in identical operation environments. Still fewer studies have looked into the use that attackers make of architectural knowledge, in terms of, for example, system decomposition topology, service entrance points, or communication graphs—to execute and plot attacks[14,15].

This paper discusses the following research questions:

•Is the targeted system based on a monolithic or microservice architecture?

By way of a detailed case study of operational software systems, this research aims to uncover how the structure of an application affects its vulnerability to security threats. The case study approach allows us to observe threats in action and investigate the real-world implications of theoretically outlined vulnerabilities reported in the literature[16,17].

This paper aims to critically evaluate the nature and extent of security threats in monolithic and microservices-based systems. Empirically, through reported incidents, we plan to establish a taxonomy of threats, evaluate the sufficiency of currently available mitigation controls, and provide practical recommendations to system architects, particularly for cloud-native and edge-deployed systems[18,19].

With the fast-paced evolution of web technology, software systems have experienced radical changes in architectures, modes of deployment, and security paradigms over the last few decades. As monolithic architecture models, as well as emerging microservice paradigms, are increasingly used worldwide, worries regarding novel problems of vulnerabilities, attack methods, and security solutions unique to architectural designs have emerged. While there is significant literature that explores conceptual threats, mitigation strategies, or threat defenses, relatively more work grounded on analyzing contrasts and similarities between threat scenarios between different architectures is still lacking.

The traditional work gives a complete picture of software architecture evolution from monolithic, tightly coupled systems to microservice-based, loosely coupled applications[1]. Microservices are undoubtedly beneficial in terms of modularity, scalability, and maintainability, but come with complex security challenges, especially in distributed systems. Above all, the increase in service-to-service communication increases the attack surface, offering more entry points for potential attackers.

Another study, in its empirical study of microservice security, found an array of common vulnerabilities like insecure APIs, inconsistent authentication mechanisms, and lack of centralized access control[4]. Through a systematic mapping study, it emphasized the importance of encryption, safe configuration management, and internal network segmentation—qualities that are not necessarily rigidly enforced in microservices architectures[12].

One of the most comprehensive multivocal security literature reviews for microservice-based systems, a breakdown of threats into three types—intra-service threats, inter-service communication risks, and external interface threats—is a multi-layered analysis of security[3]. Also, their research in 2019 graphically depicted the commonly used security mechanisms employed in microservices, such as token-based authentication (e.g., JWT), role-based access control (RBAC), and application-level firewalls[3].

One of the studies introduced a security orchestrator that orchestrates authentication, logging, and threat response in service-oriented systems[13]. Their work puts focus on how complex security orchestration is when services are extremely dynamic and independently deployable. Another study introduced a fine-grained access control model, with coarse-grained or static policies being insufficient for current service ecosystems[6].

The next paper studied a microservices-specific zero-trust parameterization model that departed from legacy perimeter security and focused instead on ongoing authentication of all service interactions and requests[9]. These are increasingly important techniques as the classic security perimeters get abstracted within cloud-native and container environments.

Despite their diminishing fame in cloud-native systems, monolithic architecture continues to pervade, most commonly in business and legacy systems. Another paper suggests, their centralized structure makes it convenient for access management[20]. However, it amplifies the impact of an attack—a vulnerability in any component of it may leave the system vulnerable in totality.

One of the papers discussed the problem of deep dependency chains in monolithic applications with cascading vulnerabilities and patch management issues[2]. Monolithic applications are particularly vulnerable to side-channel attacks such as Meltdown and Spectre, in which shared memory and speculative execution are targeted at the hardware level[10,11]. Such attacks are prone to monoliths due to tight integration and shared memory space.

The next paper also investigated the bitter reality of side-channel vulnerabilities in conventional architectures through their work on acoustic attacks against printers,

demonstrating how physical attributes of equipment can even offer sensitive information leakage[21].

One of the most important shortcomings of existing literature is that there is not much comparative and empirical analysis of threat patterns in monolithic and microservice architectures. While many works focus on security in one or the other paradigm, few address how architectural design itself affects threat surfaces, threat propagation, and attacker behavior [22].

In addition, most of the current body of research is either simulation-based or conceptual-based. Additionally, there is a scarcity of case study–based work that studies security vulnerabilities in operating environments where deployment constraints, configuration issues, and real-world attacker profiles are present. Such an empirical study is needed to bridge the gap between theoretical knowledge and real-world security practice.

The application of architectural knowledge to attacker strategies—how attackers may apply knowledge about service graphs, ingress points, or inter-process communication in system mapping and attacking—lacks enough research. Identification of these dynamics is critical for planning ahead of time.

A three-layered conceptual threat analysis framework for security threats in software architecture is suggested based on the literature reviewed:

1. Structural Layer: Captures the top-level architectural model (microservices or monolithic) and internal dependencies, service decomposition, and the number of microservices.

This layered architecture provides a basis for structured threat analysis in production-quality software systems. It also acts as a reference model to classify, compare, and neutralize threats under both architecture paradigms.

METHODS

This chapter outlines the procedure for designing, implementing, and evaluating a novel client-side approach to infer the architectural style of web-based systems indirectly. The primary objective of this approach is to estimate the classification of systems as Monolithic or Microservice-Based architectures based on the analysis of resources downloaded by the browser and system response patterns. The following sections describe in detail the data acquisition methods, preprocessing and refinement stages, and statistical computation based on cluster modeling through the application of programming and statistical software.

A. 2.2 Research Design

The research in this thesis is applied-analytic and conducted with an empirical case study. There are two levels of data analysis:

1. Inference of system structure through the analysis of external resources and the number of domains loaded to the client-side
2. Investigating server response behavior using statistical modeling to identify underlying patterns in the data.

These complementary techniques enable a multi-layered examination of system design and performance under real operating conditions.

B. 2.3 Tools and Data Collection Methods

Data were pulled using an open-source tool named Zbrowse, which is Node.js and a Headless Chromium engine-based. The tool simulates a browser without a user interface, captures all network requests and responses of the target page, and stores them as JSON files.

Here, each target site was monitored at every three-minute interval for 24 hours, and a total of 500 tracking sessions were created. Each session's outcome was stored as a JSON file with details such as the resource URL, request type, headers, request time, and origin server.

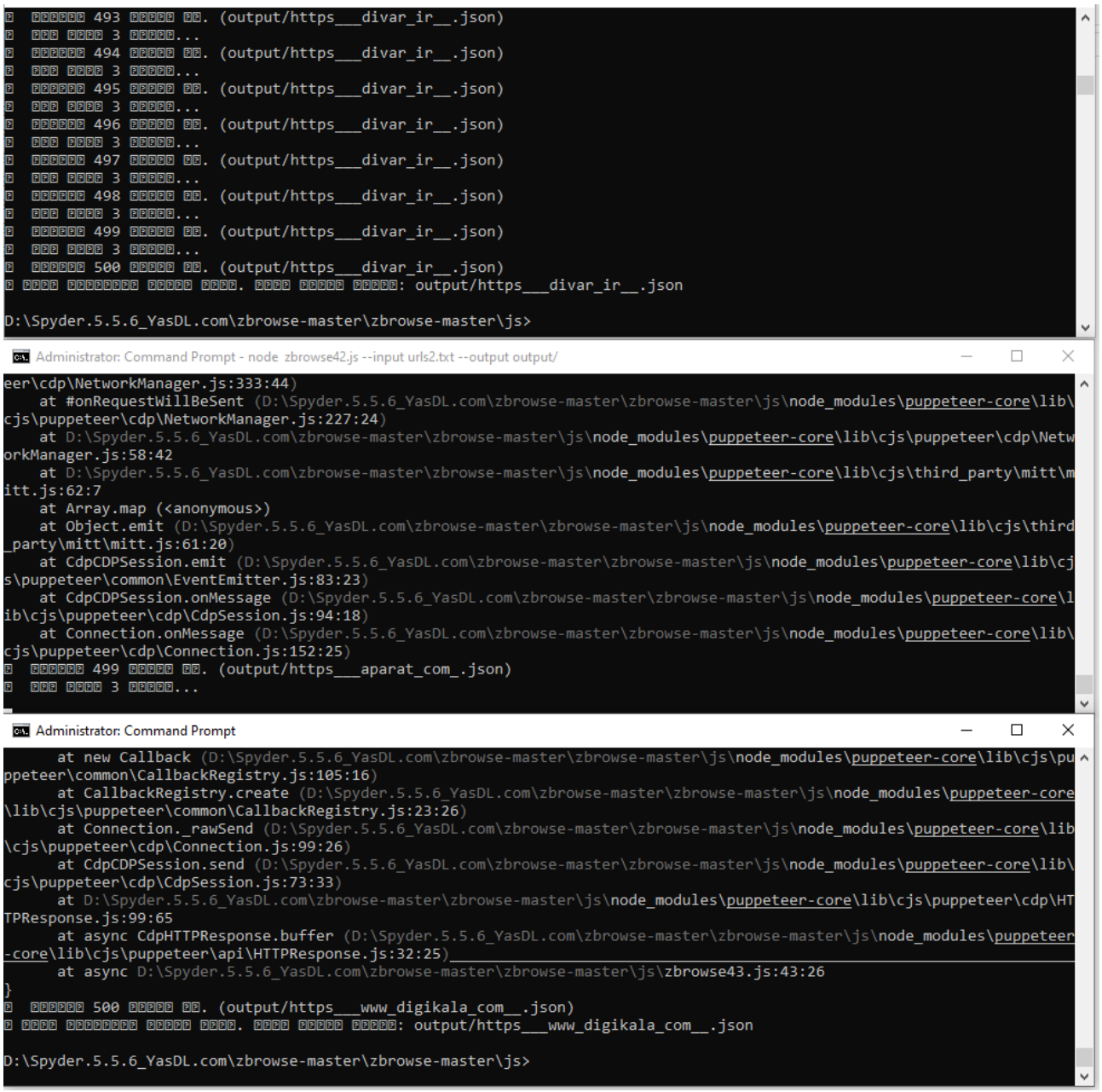

Fig 1- Console Output of Client-Side Inference and JSON File Generation in VyOS Architecture Detection Framework

C. 2.4 Data Preprocessing and Refinement

At the preprocessing stage, the JSON files obtained were worked on in Python with appropriate libraries. Python scripts were utilized to parse and get vital information such as resource domain, Server header, file type (JavaScript or CSS), and response origin.

During this stage, the external domain resources provided through Content Delivery Networks (CDNs) were not considered. Only resources that are directly related to the target site's infrastructure were retained. For instance, the Server header for a site like "Divar" was identified as "Sotoon" and displayed a specific cloud infrastructure, whereas "Jonoob Iran" type systems lacked a specific header and were categorized separately.

D. 2.5 Inferring System Architecture from the Client-Side Perspective Domain Count Heuristic for Architectural Inference

After extracting all resource domains from client-side observations, the number of unique domains was used as an approximate indicator of system modularity:

A high domain count may suggest a microservice-based or modular architecture, where services such as authentication, API endpoints, static resources, and third-party integrations are hosted on distinct subdomains or servers.

A low domain count, often close to one, may reflect a monolithic architecture, in which most or all resources are delivered from a single origin.

It is important to note that this approach is preliminary and heuristic. It does not replace detailed backend architectural analysis, but it provides a lightweight, scalable inference method for environments where server-side access is restricted or unavailable.

E. 2.6 Statistical Analysis of Response Time Using GMM

For the purpose of analyzing system response patterns, time data obtained in the aparat_output.csv file were utilized. The file was first read with the assistance of the pandas library in Python. Response time values (in milliseconds by default) were scaled to seconds and arranged as a two-dimensional array suitable for modeling.

Lastly, clustering was performed under the Gaussian Mixture Model (GMM). The model was trained for various numbers of clusters (from 1 to 9), and in each arrangement, two information criteria were calculated:

•Akaike Information Criterion (AIC)

•Bayesian Information Criterion (BIC)

The model with the smallest BIC value was selected as the optimal model. To visualize the results, Kernel Density Estimation (KDE) was used to approximate the probability distribution of the data. Density curves were plotted for the entire dataset and each identified cluster. If two or more clusters were found, the intersection point of the dominant distributions was recommended as a decision threshold. All statistical processing, visualization, and saving of outputs were done via Python scripting.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture

# خواندن داده
df = pd.read_csv('I:/scan/huda/uni/term2/system/zbrowse-master/zbrowse-master/js/digikala_output.csv')
response_times = df['response_time_ms'] / 1000.0
X = response_times.values.reshape(-1, 1)

# محاسبه BIC و AIC برای تعداد کلاسترهای مختلف
n_components_range = range(1, 10)  # تعداد کلاستر از 1 تا 9
bics = []
aics = []

for n_components in n_components_range:
    gmm = GaussianMixture(n_components=n_components, random_state=0)
    gmm.fit(X)
    bics.append(gmm.bic(X))
    aics.append(gmm.aic(X))

# رسم نمودار
plt.figure(figsize=(12, 6))

plt.plot(n_components_range, bics, label='BIC', marker='o')
plt.plot(n_components_range, aics, label='AIC', marker='s')
plt.xlabel('Number of Components (Clusters)')
plt.ylabel('Score')
plt.title('BIC and AIC Scores for GMM')
plt.legend()
plt.grid(True)
plt.show()
```

Fig 2- Codes of Cluster

```python
# خواندن داده‌ها
df = pd.read_csv('I:/scan/huda/uni/term2/system/zbrowse-master/zbrowse-master/js/digikala_output.csv')
response_times = df['response_time_ms'] / 1000.0  # به ثانیه تبدیل شود

# آماده سازی داده
X = response_times.values.reshape(-1, 1)

# پیدا کردن بهترین تعداد کلاستر با استفاده از BIC
lowest_bic = np.infty
bic = []
n_components_range = range(1, 6)  # از 1 تا 6 خوشه بررسی کن
for n_components in n_components_range:
    gmm = GaussianMixture(n_components=n_components, random_state=0)
    gmm.fit(X)
    bic_score = gmm.bic(X)
    bic.append(bic_score)
    if bic_score < lowest_bic:
        lowest_bic = bic_score
        best_gmm = gmm
print (bic)
# برچسب‌های نهایی کلاستر
labels = best_gmm.predict(X)

# رسم نمودار
plt.figure(figsize=(10, 6))

# رسم KDE کل داده‌ها
kde_total = gaussian_kde(response_times)
x_grid = np.linspace(response_times.min() - 0.01, response_times.max() + 0.01, 1000)
plt.plot(x_grid, kde_total(x_grid), label='KDE', color='black')

# رنگ‌بندی برای کلاسترها
colors = ['blue', 'orange', 'green', 'purple', 'brown', 'cyan']
for cluster in np.unique(labels):
    cluster_data = response_times[labels == cluster]
    kde = gaussian_kde(cluster_data)
    plt.plot(x_grid, kde(x_grid), label=f'Cluster {cluster+1}', color=colors[cluster % len(colors)], lw=2)

# اگر حداقل دو کلاستر وجود داشت، threshold بین دو کلاستر اصلی حساب کن
means = best_gmm.means_.flatten()
sorted_idx = np.argsort(means)
if len(sorted_idx) >= 2:
    mean1, mean2 = means[sorted_idx[0]], means[sorted_idx[1]]
    std1 = np.sqrt(best_gmm.covariances_[sorted_idx[0]]).flatten()[0]
    std2 = np.sqrt(best_gmm.covariances_[sorted_idx[1]]).flatten()[0]
    threshold = (mean1 * std2**2 - mean2 * std1**2 + std1 * std2 * np.sqrt((mean2 - mean1)**2 + 2*(std2**2 - std1**2)*np.log(std2/std1))) / (st
```

Fig 3- Python Script for Gaussian Mixture Model (GMM)-Based Clustering and Threshold Estimation of Response Times from Client-Side Log Data

F. 2.7 Advantages of the Proposed Approach

•Utilization of client-side observations to conclude system structure without source code or server access;
•Accurate response behavior modeling using GMM with the ability to uncover hidden properties;
•Application of AIC and BIC criteria for the optimal choice of clusters, minimizing human intervention;
•Capacity to visualize outcomes and determine behavioral thresholds for identifying delays or anomalies;
•A fully automated, code-based implementation on the Python platform.

This chapter introduced a new research methodology for indirectly investigating system architectures from the client-side perspective and statistically evaluating server response behaviors. Through the integration of data collection tools, Python coding, GMM modeling, and loaded domain analysis, an approximate classification of system architectures was achieved. The proposed methodology provides a light, fast, and non-intrusive means of evaluating web-based systems in real-world settings. The detailed results obtained through this methodology will be presented in part three.

G. 2.8.Comparison of Clustering Algorithms: GMM, K-means, and DBSCAN

To validate the decision to use the Gaussian Mixture Model (GMM) as the main clustering approach in this study, a comparison with two other well-known algorithms, specifically K-means and DBSCAN, was conducted. Each of these algorithms possesses some advantages and disadvantages depending on the characteristics of the data and the aims of clustering.

GMM provides soft clustering, which assigns a probability distribution over clusters to each data point. It is particularly useful when the clusters overlap or are of varying shapes and densities. GMM also supports elliptical-shaped clusters and can represent complex data patterns, making it a competitive choice to model heterogeneous server response behaviors. However, it has the assumption that the data are Gaussian distributed and is computationally more costly than less complex algorithms like K-means.

K-means, however, is a fast and efficient algorithm suitable for large datasets. It performs well when clusters are spherical, equal in size, and well separated. However, it makes hard assignments (each data point

belongs to one cluster) and is very sensitive to the initial placement of centroids and to outliers.

DBSCAN handles it differently by using a density-based approach to discover clusters of any shape without requiring the number of clusters as input. It is extremely robust to noise and can identify outliers explicitly. Nevertheless, the success of DBSCAN depends heavily on the selection of its parameters (ε and min_samples), and it does not perform well on datasets with differing densities or high dimensions.

Briefly, based on the properties of this study's dataset, namely the presence of overlapping response patterns and the aspiration for probabilistic interpretation, GMM was selected as the most appropriate algorithm. Its ability to model uncertainty and relaxed cluster forms is well adapted to the analytical goal of extracting faint behavioral trends in server performance.

Results and Discussion

A. 3.1.Clustering of Response Times and Statistical Model Selection

To analyze latent patterns and behavioral trends in the response times of the Tabnak server, a Gaussian Mixture Model (GMM) was applied to the observed server response time data (measured in seconds). The analysis provided insights into the distribution of response times, identification of underlying clusters, and the definition of a threshold for distinguishing between different system performance zones.

The second graph shows the evaluation of GMMs with varying numbers of components, ranging from 1 to 9 clusters. Two standard statistical metrics—Bayesian Information Criterion (BIC) and Akaike Information Criterion (AIC)—were employed to determine the optimal number of clusters. As expected, both BIC and AIC values generally decreased with the addition of more clusters, indicating an improved model fit. However, a distinct elbow point was observed at five clusters, beyond which the reduction in BIC and AIC values became marginal. This suggests diminishing returns in model performance and increased complexity. Therefore, the five-cluster model was selected as the optimal trade-off between model accuracy and simplicity.

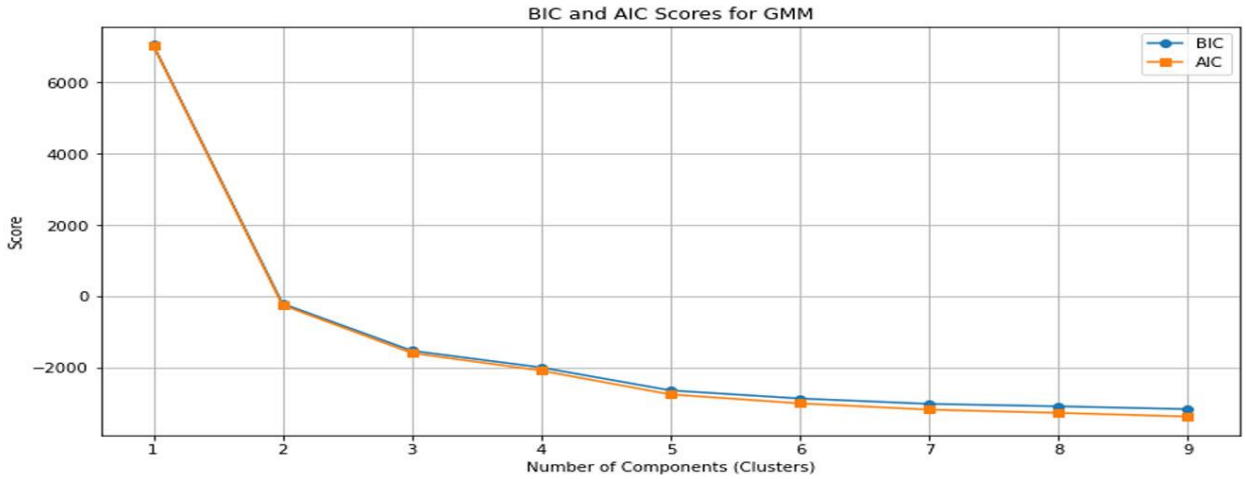

*Fig. 4.*BIC and AIC for GMM

B. 3.1.2. Distribution of Response Times

The first graph illustrates the Kernel Density Estimation (KDE) of response time distributions across the five clusters identified by the GMM for the Tabnak server. Each cluster is represented by a distinct color, signifying different server performance behaviors:

Clusters 1 and 2 (blue and orange) show the highest density in response times below one second, indicating the fastest and most efficient server behavior.

Cluster 3 (green) represents moderate latency, reflecting standard server response times under typical conditions.

Clusters 4 and 5 (purple and brown) correspond to slower response times with greater variance, signaling degraded performance or potential system issues.

A red vertical line in the plot indicates the optimal threshold, derived from the intersection of the two most dominant Gaussian components. This threshold serves as a practical boundary to differentiate between normal and abnormal server response behavior

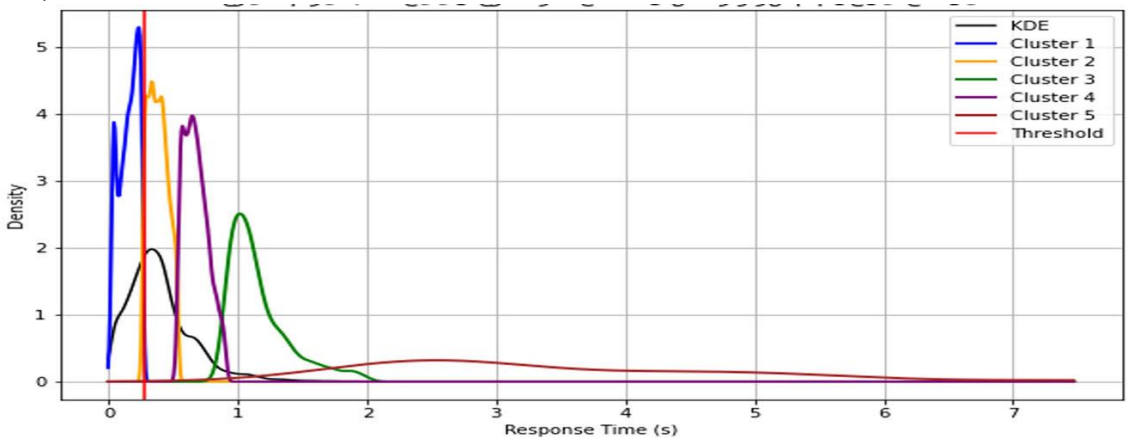

*Fig. 5.*Clustering of Response Times and Statistical Model Selection of Tabnak

C. 3.1.3. Interpretation and Application of Results

The results highlight significant variability in Tabnak's server response performance, validating the assumption that server behavior patterns can be inferred from client-side response data. The heterogeneity in response times may point to underlying architectural features—such as a modular (e.g., microservices-based) infrastructure—or differences in server load, configuration, or hosting environments. Moreover, the use of GMM, along with BIC and AIC for cluster validation, enabled a data-driven approach to determining optimal performance segmentation and establishing a statistically robust performance threshold.

D. 3.2. Response Time Analysis by GMM Clustering

This section presents the results of the statistical analysis of server response times for the Divar platform, using the Gaussian Mixture Model (GMM). The primary aim was to uncover latent behavioral trends in server responsiveness and determine the appropriate number of clusters that best represent these patterns. Two key visualizations were generated for this purpose:

A density plot displaying the clustered distribution of response times;

A model selection plot showing BIC and AIC values across different GMM configurations.

E. 3.2.1. Determining the Optimal Number of Clusters

The model selection plot compares the Bayesian Information Criterion (BIC) and the Akaike Information Criterion (AIC) for GMMs with cluster counts ranging from 1 to 9. Both metrics initially decrease sharply, reflecting a significant improvement in model fit as more clusters are introduced. However, after the sixth cluster, the rate of decline in both BIC and AIC becomes minimal, signaling that adding further clusters yields negligible gains while increasing model complexity. Based on this inflection point, the six-cluster model was selected as the optimal balance between accuracy and parsimony for analyzing Divar's server response times.

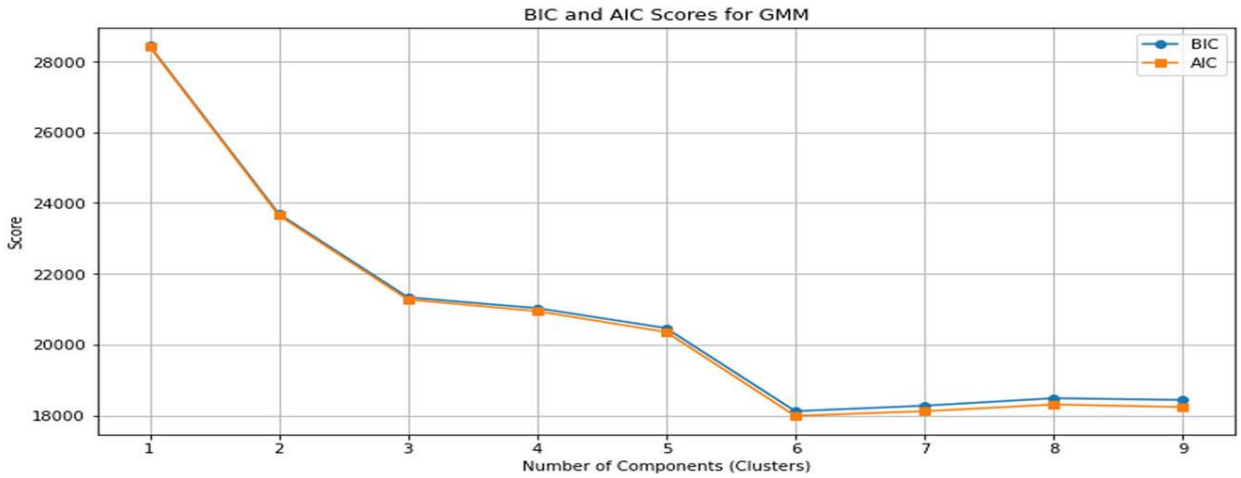

*Fig. 6.*BIC and AIC for GMM

F.3.2.2. Clustering Distribution of Response Times
The first plot displays the estimated Probability Density Function (PDF) of Divar's server response times using Kernel Density Estimation (KDE), overlaid with the six clusters identified by the GMM. Each cluster is color-coded to reflect distinct behavioral patterns:

Clusters 1 and 2 (e.g., orange and purple) represent extremely short response times (under one second), which signify optimal server performance.

Clusters 3 and 4 (e.g., blue and green) fall within moderate response ranges and likely correspond to standard system behavior with slight latency.

Clusters 5 and 6 (e.g., black and brown) exhibit extended response times, often exceeding 10 seconds, which may point to server overload, bottlenecks in modular services, or repeated request handling.

A red vertical line on the graph indicates the threshold derived from the intersection of the two most influential Gaussian components. This threshold can be used to distinguish between normal and abnormal response patterns for Divar's system performance.

G.3.2.3. Interpretation of Behavioral Patterns and PracticalApplications
The clear emergence of six distinct response time clusters suggests a high degree of heterogeneity in Divar's server performance. This variability may stem from factors such as a microservices-based infrastructure, asynchronous service calls, or the distribution of load across independent service modules. Notably, the presence of long-tail distributions in clusters 5 and 6 may signal issues related to scalability, inefficient resource allocation, or request retries under high load conditions.

Moreover, the GMM's ability to identify optimal clusters and define behavioral thresholds makes it a valuable tool for performance monitoring and anomaly detection. Its reliance on client-side response data and unsupervised learning enables a lightweight, scalable approach for real-time diagnostics and architectural insights, particularly relevant for complex platforms like Divar.

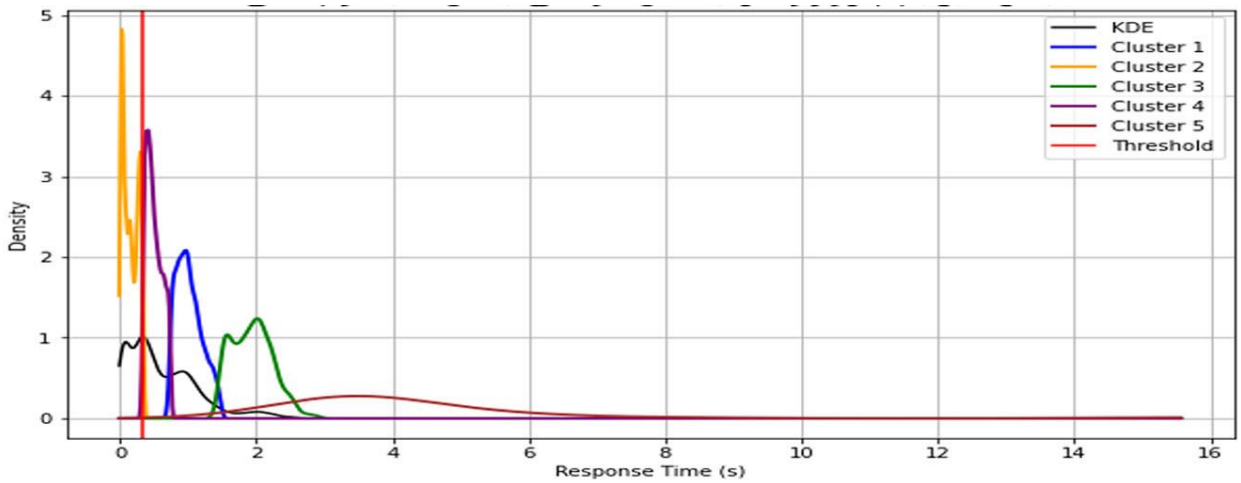

*Fig. 7.*Clustering of Response Times and Statistical Model Selection of Divar

H.3.3. Response Time Analysis Using the GMM Model

This section presents the statistical analysis of server response time data for the Aparat platform. The primary objective was to identify underlying behavioral patterns in response latency and determine the optimal number of clusters through the use of the Gaussian Mixture Model (GMM). Two key visualizations formed the basis of this analysis:

An estimated response time density plot with clearly separated clusters;

A model selection plot using standard statistical criteria.

I.3.3.1. Determining the Optimal Number of Clusters Using BIC and AIC

The second plot illustrates the values of the Bayesian Information Criterion (BIC) and the Akaike Information Criterion (AIC) for GMMs with components ranging from 1 to 9. As observed, both criteria drop sharply at the beginning, suggesting a better model fit with more clusters. However, this decline starts to level off around five clusters, indicating an "elbow point." This plateau suggests that increasing the number of clusters beyond five results in marginal gains but greater model complexity. Therefore, the five-component GMM was selected as the most balanced model, offering sufficient accuracy without unnecessary complexity in analyzing Aparat's server response data.

J.3.3.2. Temporal Distribution Analysis Based on Clusters

The first plot shows the Probability Density Function (PDF) of Aparat's response times, estimated through Kernel Density Estimation (KDE), overlaid with the five clusters identified by the GMM. Each cluster is represented in a distinct color to highlight different response behaviors:

A red vertical line in the plot signifies the threshold between normal and abnormal behavior, determined by the intersection of the two most prominent Gaussian components. This threshold may serve as a practical reference point in monitoring Aparat's system performance.

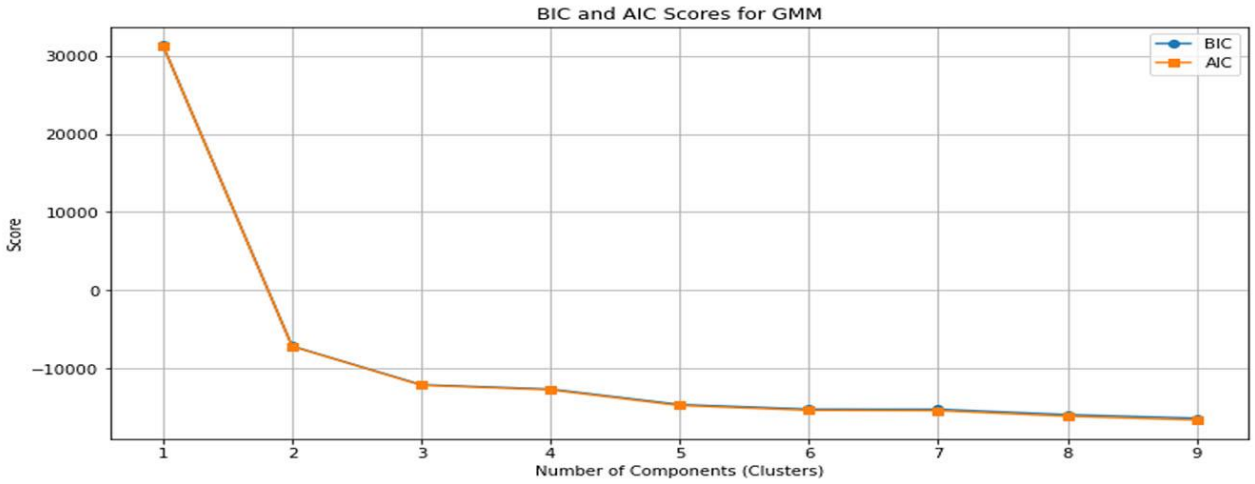

Fig. 8:.BIC and AIC for GMM

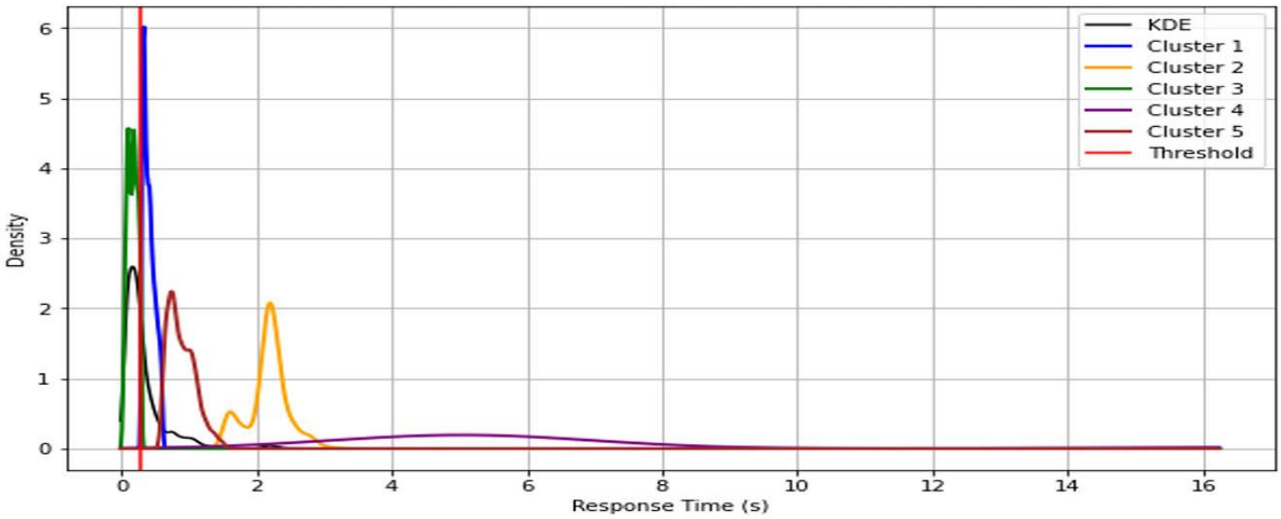

Fig. 9:.Clustering of Response Times and Statistical Model Selection of Apart

K.3.3.3. Ultimate Interpretation and Practical Applications

The identification of multiple clusters in Aparat's response time data confirms the presence of heterogeneous behavioral patterns. This variation may result from underlying architectural factors, such as the use of microservices, load balancing mechanisms, or differences across content delivery modules.

By leveraging the GMM along with BIC and AIC for statistical validation, this analysis enabled a data-driven approach to uncovering and categorizing system behaviors. The resulting model can be applied in real-world settings for performance monitoring, early anomaly detection, and client-side inference of system architecture. Overall, this methodology offers a scalable, lightweight analytical framework suitable for ongoing assessment of complex web platforms like Aparat.

L.3.4. Response Time Analysis of the "Jonoob Iran" System with the GMM Model

Here, the outcome of the response time analysis of the Jonoob Iran system using the Gaussian Mixture Model (GMM) and model selection criteria, i.e., Bayesian Information Criterion (BIC) and Akaike Information Criterion (AIC), is given. The purpose was to discover hidden patterns in the responsiveness of the system and also the optimal number of clusters in the data.

M.3.4.1. Optimal Cluster Number Based on BIC and AIC

The second figure illustrates the values of the Bayesian Information Criterion (BIC) and Akaike Information

Criterion (AIC) for Gaussian Mixture Models (GMMs) with 1 to 9 components. As the number of clusters increases, both metrics initially decrease sharply, indicating improved model fit. However, after the second cluster, the rate of decline becomes more gradual, eventually stabilizing around the fifth cluster. This trend suggests that further increasing the number of clusters beyond five results in marginal improvement while adding unnecessary complexity. Therefore, the five-cluster configuration was chosen as optimal, representing the best trade-off between model accuracy and simplicity in analyzing the response behavior of the Jonob Iran server.

N. 3.4.2. Response Behavior and Clustered Distribution Analysis

The first figure presents the Kernel Density Estimation (KDE) of the response time distribution, overlaid with the five clusters identified by the GMM:

Clusters 1 and 2 (blue and orange) represent very short response times (under 2 seconds), corresponding to efficient and stable system performance for the majority of requests.

Cluster 3 (purple) falls within a medium response time range, approximately between 4 and 6 seconds, reflecting moderate system delays likely due to transient load or routine processing overhead.

Cluster 4 (brown) contains high-latency responses exceeding 6 seconds, which may be indicative of system congestion, processing bottlenecks, or inefficiencies within specific modules.

Cluster 5 (green) extends into a long-tail distribution, with response times reaching up to 25 seconds. This cluster most likely captures outliers, anomalies, or failures such as request queuing, server overload, or internal system errors.

A red vertical line is drawn on the KDE plot to indicate the analytical threshold derived from the intersection of the two most dominant Gaussian components. This threshold serves as a practical boundary to distinguish between normal and abnormal system behavior in Jonob Iran's web infrastructure.

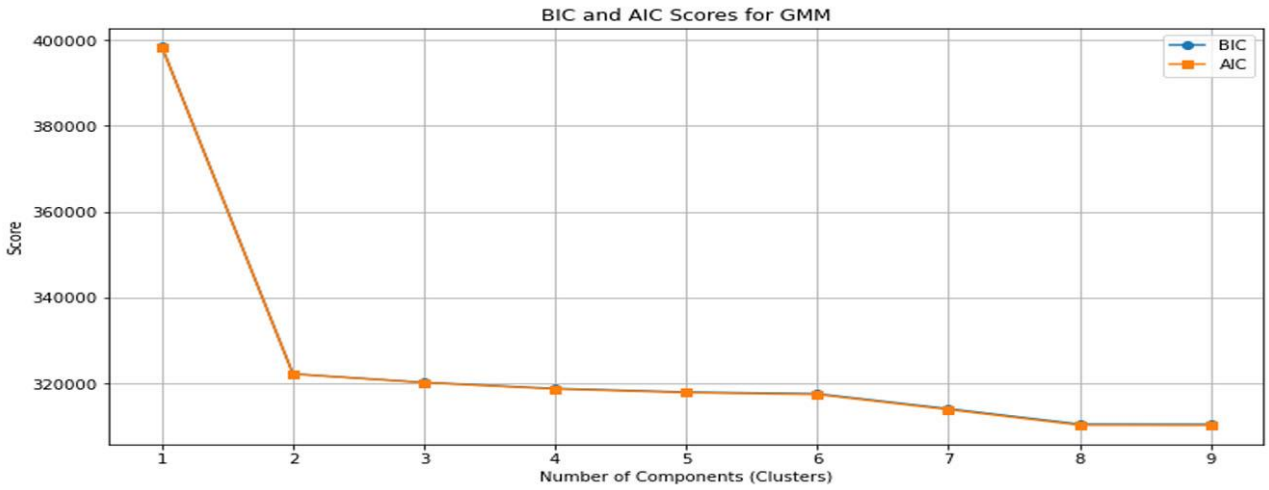

*Fig. 10:.*BIC and AIC for GMM

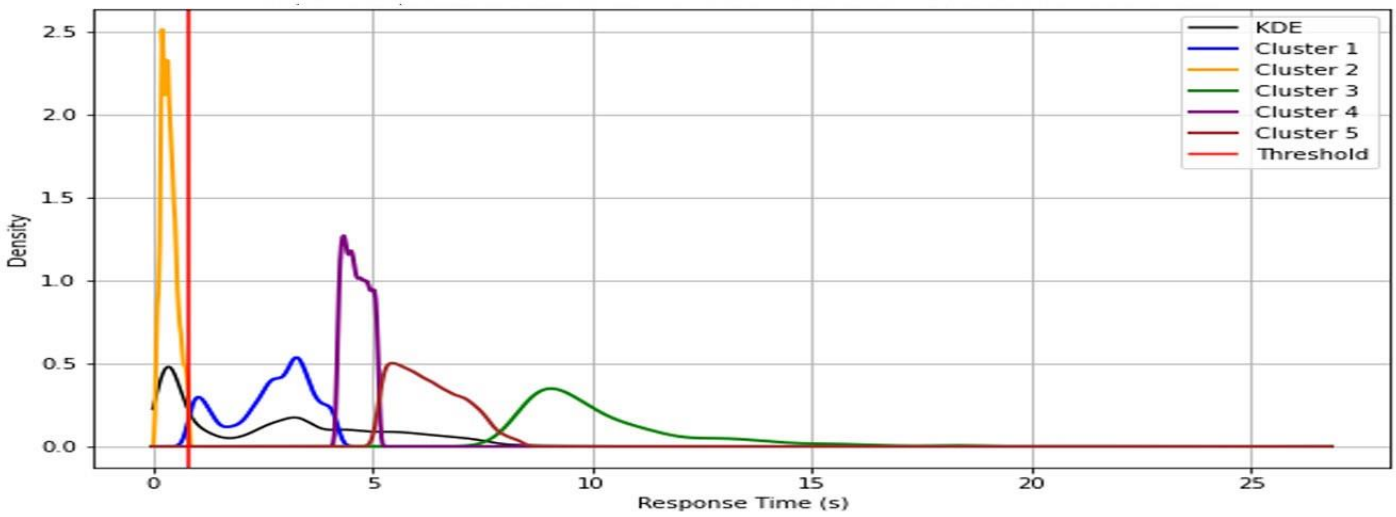

Fig. 11:.Clustering of Response Times and Statistical Model Selection of Apart

O. 3.4.3. Final Interpretation and Implications

The clustering analysis confirms the existence of multiple distinct behavioral patterns in Jonob Iran's server response times. In particular, the presence of a long-tailed fifth cluster suggests underlying issues such as scalability limitations, lack of asynchronous request handling, or architectural inefficiencies. The observed cluster structure points to considerable systemic complexity, which may be consistent with a monolithic or tightly coupled architecture, as opposed to modular or microservice-based designs.

In summary, applying the GMM model allowed for a nuanced and data-driven segmentation of response time behaviors. Supported by BIC and AIC metrics, the model facilitated accurate cluster selection and helped establish performance thresholds. These findings have practical applications in system performance monitoring, anomaly detection, and architectural diagnostics based on client-side web analytics, particularly for systems like Jonob Iran.

P. 3.5. Statistical Analysis of the Response Times of the "Digikala" System Based on the GMM Model

The current section gives the statistical analysis of the response times of the Digikala system based on the Gaussian Mixture Model (GMM). The main purpose of this analysis was to uncover latent behavioral clusters in the data and to set the boundaries separating normal from anomalous system behavior based on statistical measures.

Q. 3.5.1. Finding the Best Number of Clusters

The second plot shows the values of two key statistical estimates—Bayesian Information Criterion (BIC) and Akaike Information Criterion (AIC)—for GMM specifications from 1 to 9 clusters. As the number of clusters increases, both estimates shrink significantly, and the decreasing rate diminishes and stabilizes after the sixth cluster. This suggests that a model with five or six clusters has the best trade-off between accuracy and parsimony. Here, the five-cluster model was used as it resulted in a major reduction of the information criteria without increasing complexity too much.

.5.2. Response Time Density Distribution Analysis

The initial figure displays the Kernel Density Estimation (KDE) of the distribution of response times, superimposed on the clusters identified by the GMM:

Clusters 1 and 2 (blue and green) are concentrated in a very low response time range (below 1.5 seconds), showing fast and good performance for the majority of the requests.

Cluster 3 (purple) spans the range of 2 to 5 seconds, illustrating responses with median latencies.

Cluster 4 (orange) is spread out more widely and includes higher response times, going up to about 15 seconds.

Cluster 5 (red) has a low-density configuration with an elongated tail over 20 seconds, possibly due to anomalous activity or high degrees of usage.

The figure's vertical red line defines the juncture point between two distributions of prevalence, serving as a boundary for discriminating between normal and potentially abnormal system operational behavior.

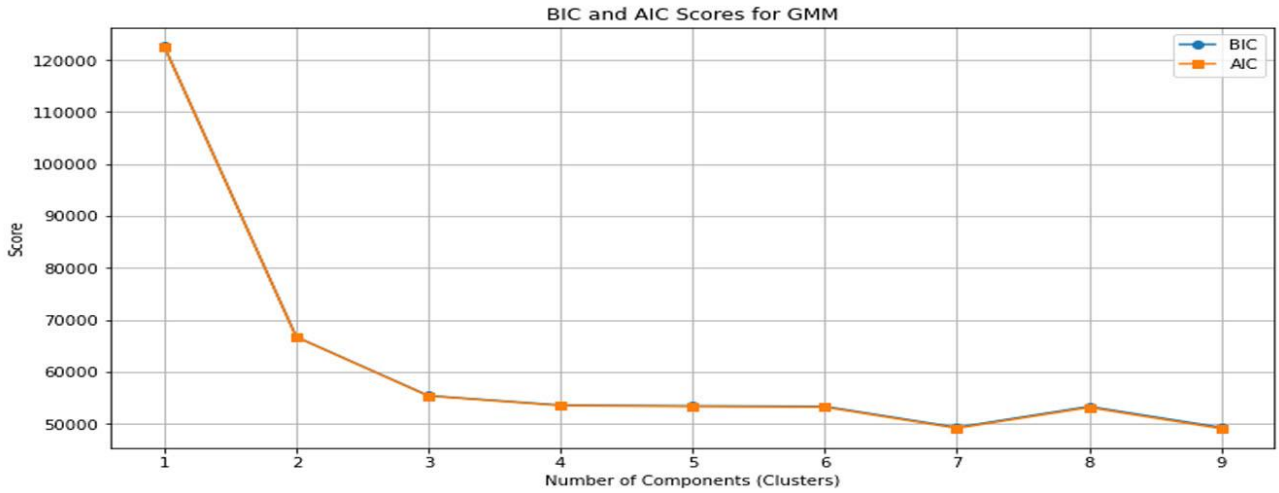

*Fig. 12.*BIC and AIC for GMM

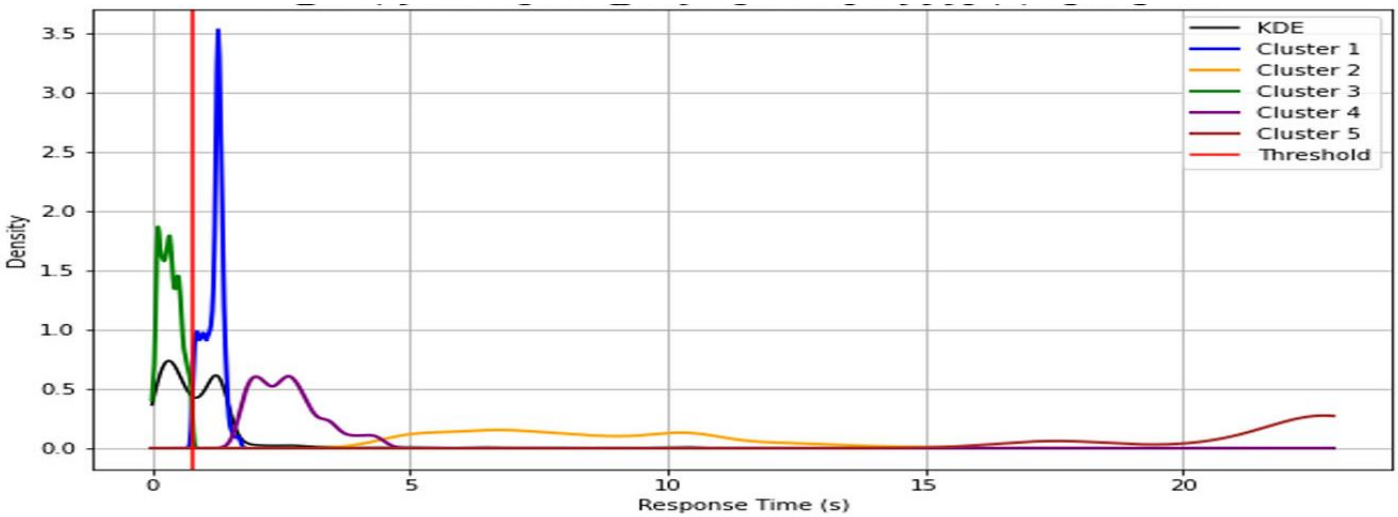

*Fig. 13.*Clustering of Response Times and Statistical Model Selection of Apart

DISCUSSION

The initial research question sought to identify whether the noted systems were running under monolithic or microservices-based architectural patterns using client-side behavioral data. Such a categorization was facilitated by two indirect indicators, namely the number of distinct domains loaded during runtime and the temporal breakdown of response times, represented using Gaussian Mixture Models (GMM). While domain count offers a useful first-pass approximation, exceptions may exist, for instance, monolithic systems using multiple CDNs or microservice systems behind a unified reverse proxy.

Empirical observations revealed different architectural structures. Digikala-type systems exhibited characteristic features of microservice architectures, i.e., numerous unique domains, modular and asynchronous request behavior, and a few low-latency response clusters. Systems such as Jonoob Iran, however, revealed characteristic features of a monolithic structure, i.e., a single central domain usage, less component interaction, and higher latency clusters typical of linear processing or bottlenecks.

These results are very congruent with a study[1], which has characterized microservice-based structures as assemblies of loosely coupled, autonomously deployable services that naturally increase scalability and modularity. Similarly, another study[4] empirically demonstrated that microservices would have more domain fragmentation and dynamic composition of services, which effects were observed in Digikala's domain footprint and grouped response profiles.

From a performance point of view, the low-latency clusters seen in Digikala confirm a study[12] finding that

microservices, if properly designed, can react quickly to localized requests due to service isolation and stateless processing. However, the presence of long-tail latency clusters in the same system is a sign of problems highlighted in a study[3], including inter-service communication overhead and complex dependency chains. These nuances further support the need for robust orchestration mechanisms, as proposed by a study[13], which advocated for dynamic authentication and access control between microservice boundaries.

Conversely, Jonoob Iran's architecture aligns with a study[2], which noted that monolithic systems tend to confine all services into a single deployable unit with fewer external dependencies. This provides simpler routing logic but greater exposure to internal bottlenecks. The system's extended, uniform latency clusters reflect the issues of some studies[10,11], which linked centralized architectures with degraded performance under speculative execution attacks (e.g., Spectre and Meltdown). Such systems usually employ shared memory spaces and lack microservice isolation benefits, making them vulnerable to a broader category of systemic threats.

Interestingly, the absence of server header signatures in Jonoob Iran, in contrast to infrastructure-named systems like Divar (e.g., "Sotoon"), provides evidence for a study[21] argument that architectural opacity, which is prevalent in legacy monolithic systems, both impedes defensive analysis and serves as a basis for side-channel reconnaissance by adversaries.

Aside from confirming existing theoretical frameworks, this work adds to the literature by introducing a client-side, non-intrusive inference method without source code or internal log access. This is distinct compared to existing simulation-heavy or backend-centric research[22], and it offers practical application to red teams, security auditors, and researchers operating in restricted-access environments.

In summary, the classification results reaffirm historical system behavior and threat exposure differences between microservice-based and monolithic systems. Using the integration of architectural heuristics and statistical analysis, this work introduces a new, lightweight architecture-aware vulnerability analysis framework for real-world environments.

A. 4.1 Comparative Analysis of System Architecture, Response Behavior, and Security Attributes

For a better understanding of the architectural and security differences among the test platforms, this section provides a comparative explanation of five sites: Tabnak, Digikala, CafeBazaar, Torob, and Jonoob Iran. Comparison is made based on key indicators such as inferred type of architecture, average response time, loaded domains, number of response clusters identified through GMM, HTTP server headers, and external resources percentage.

The results indicate that pages such as Digikala, Torob, and CafeBazaar, which exhibit architectural characteristics of microservices, should have more variability in response times and more variance in domain requests. These are characteristic of the distributed, loosely coupled nature of microservice architecture. Tabnak, as a sample, exhibits more regular response behavior and little domain variability, the signature of a monolithic system. However, its centralized character places it at risk of more comprehensive security vulnerabilities in the event of an organized attack.

Similarly, the Jonoob Iran system with low domain diversity, consistent response behavior, and no visible HTTP server headers has been classified as a monolithic architecture. These findings are also in alignment with previous results by Almeida et al. (2017) and Kocher et al. (2019), who identified that monolithic systems involving centralized infrastructure and memory are predominantly vulnerable to side-channel attacks and information leakage.

This comparative assessment not only brings out the differences in architecture between the systems but also provides insightful observations on their security weakness, threat posture, and the growing imperative to embrace multi-layered defense strategies and zero-trust frameworks in modern web-based scenarios.

Conclusion

This study sought to evaluate security threats in monolithic and microservice architectures with an empirical, client-side analysis approach. The central objective was to develop and experiment with a lightweight technique for indirectly inferring architectural models and detecting performance-centric behavioral patterns based on browser-level resource monitoring and response time clustering.

By meticulous case studies of existing systems such as Digikala and Jonoob Iran, the research explained that architectural variance can be coherently concluded using non-intrusive indicators such as the number of different domains accessed and temporal behavior explained in terms of the Gaussian Mixture Model (GMM). The study confirmed that systems with higher domain diversity, modular latency profiles, and distributed behaviors were more predictive of microservice architectures. In contrast, centralized domain dependency and longer, homogeneous latency patterns predicted monolithic architectures.

Secondarily, the availability of model selection criteria such as AIC and BIC allowed objective characterization of behavioral clusters that, when decoded together with architectural theory, provided strong evidence for system modularity, scalability, and susceptibility to performance degradation or threat propagation.

5.1 Theoretical Implications

This study contributes to the theoretical literature on architectural security by bridging an essential gap between abstract threat modeling and empirical data. While previous research[1,3,12] has introduced conceptual threats in microservices and monolithic systems, little empirical validation of architectural inference models in real operational environments without backend access has existed.

By enabling the presentation of a client-side, statistical, and architecture-aware inference model, the research contributes to the analytical toolkit available for software architecture analysis. It also makes the three-layered threat analysis conceptual model stronger, as indicated by the literature review. The ability to infer architectural patterns from empirically visible client-side behavior contributes to a corpus of research in "black-box" security analysis and architecture fingerprinting.

5.2 Practical Implications

In practice, the results of this work have significant implications for security professionals, system architects, and auditors:

For pen testers and red teams, the approach delineated facilitates hidden reconnaissance and architectural fingerprinting of target systems without requiring high privilege or server-side permission.

For DevOps operators and system designers, the work presents quantifiable indicators—response clustering and domain diversity—that can be monitored in order to reason about architectural health and identify outliers in performance.

For defenders and security operations centers (SOCs), understanding how architecture is actualized in traffic and response patterns can inform the development of anomaly detection systems that are architecture-aware.

Also, the employment of open-source applications such as Zbrowse and Python during automated implementation makes it highly replicable and transferable for organizations looking to enhance their security posture using lightweight architectural diagnosis.

5.3 Additional Research Recommendations

While this research provides a compelling method, it also inspires a number of avenues for future research:
Expanded Dataset Domain: Expanding the dataset domain to include more domains of systems in more industries (e.g., financial, healthcare, IoT) would increase its generality and support sector-specific threat modeling.
Integration with Real-time Threat Intelligence: This can be integrated with real-time threat intelligence feeds in future research to match observed vulnerabilities or attacks with deduced architecture.
Machine Learning-Based Classification: The application of supervised or semi-supervised learning methods can enhance classification accuracy, particularly by incorporating additional client-side features such as script origin, request frequency, and network entropy.
Server-Side Validation: Although this study employed only client-side data, supporting evidence from backend telemetry (where available) would validate the accuracy of the inference method.
Zero-Trust and API-Centric Architectures: With the growing importance of zero-trust architectures and API-first systems, the proposed method can be further applied to infer API distribution and trust boundaries in cloud-native systems.
Lastly, this work not only sheds light on the apparent dissimilarities between monolithic and microservice architectures but also presents a new, non-intrusive system profiling methodology of theoretical and practical interest in the rapidly evolving realm of software security.

Conflict of Interest

“All authors declared that there are no conflicts of interest”.

References

[1] Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: Yesterday, today, and tomorrow. Present Ulterior Software Engineering, 195–216. https://doi.org/10.1007/978-3-319-67425-4_12

[2] Almeida, W. H. C., de Aguiar Monteiro, L., Hazin, R. R., de Lima, A. C., & Ferraz, F. S. (2017). Survey on microservice architecture - security, privacy and standardization on cloud computing environment. ICSEA, 302–307.

[3] Pereira-Vale, A., Fernandez, E. B., Monge, R., Astudillo, H., & Marquez, G. (2021). Security in microservice-based systems: A multivocal literature review. Computer Science Review, 40, 100400. https://doi.org/10.1016/j.cosrev.2021.100400

[4] Alshuqayran, N., Ali, N., & Evans, R. (2018). Towards microservice architecture recovery: An empirical study. In 2018 IEEE International Conference on Software Architecture (ICSA) (pp. 47–56). https://doi.org/10.1109/ICSA.2018.00013

[5] Pereira-Vale, A., Marquez, G., Astudillo, H., & Fernandez, E. B. (2019). Security mechanisms used in microservices-based systems: A systematic mapping. In 2019 Latin American Computing Conference (CLEI) (pp. 1–10). https://doi.org/10.1109/CLEI47609.2019.235088

[6] Nehme, A., Jesus, V., Mahbub, K., & Abdallah, A. (2019). Fine-grained access control for microservices. In Foundations and Practice of Security (pp. 193–208). https://doi.org/10.1007/978-3-030-29959-0_10

[7] Yu, D., Jin, Y., Zhang, Y., & Zheng, X. (2018). A survey on security issues in services communication of microservices-enabled fog applications. Concurrency and Computation: Practice and Experience.

[8] Li, X., Chen, Y., & Lin, Z. (2019). Towards automated inter-service authorization for microservice applications. In Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos.

[9] Zaheer, Z., Chang, H., Mukherjee, S., & Van der Merwe, J. (2019). EzTrust: Network-independent zero-trust perimeterization for microservices. In Proceedings of the ACM Symposium on SDN Research (pp. 25–36). https://doi.org/10.1145/3314148.3314357

[10] Kocher, P., Horn, J., Fogh, A., Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., Mangard, S., Prescher, T., Schwarz, M., & Yarom, Y. (2019). Spectre attacks: Exploiting speculative execution. In 2019 IEEE Symposium on Security and Privacy (SP) (pp. 1–19). https://doi.org/10.1109/SP.2019.00002

[11] Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Fogh, A., Horn, J., Mangard, S., Kocher, P., Genkin, D., Yarom, Y., & Hamburg, M. (2018). Meltdown: Reading kernel memory from user space. In USENIX Security Symposium (pp. 973–990). https://www.usenix.org/conference/usenixsecurity18/presentation/lipp

[12] Hannousse, A., & Yahiouche, S. (2020). Securing microservices and microservice architectures: A systematic mapping study. Computer Science Review, 38, 100303. https://doi.org/10.1016/j.cosrev.2020.100303

[13] Banati, A., Kail, E., Karoczkai, K., & Kozlovszky, M. (2018). Authentication and authorization orchestrator for microservice-based software architectures. In 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO) (pp. 1204–1209). https://doi.org/10.23919/MIPRO.2018.8400160

[14] Granchelli, G., Cardarelli, M., Di Francesco, P., Malavolta, I., Iovino, L., & Salle, D. (2017). Microart: A software architecture recovery tool for maintaining microservice-based systems. In 2017 IEEE International Conference on Software Architecture Workshops (ICSAW).

[15] Ahmadvand, M., & Ibrahim, A. (2016). Requirements reconciliation for scalable and secure microservice (de)composition. In 2016 IEEE 24th International Requirements Engineering Conference Workshops (REW).

[16] Ravichandiran, R., Bannazadeh, H., & Leon-Garcia, A. (2018). Anomaly detection using resource behaviour analysis for autoscaling systems. In Proceedings of the 4th IEEE Conference on Network Softwarization and Workshops (NetSoft).

[17] Sun, Y., Nanda, S., & Jaeger, T. (2015). Security-as-a-service for microservices-based cloud applications. In 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom).

[18] Pahl, M.-O., & Donini, L. (2018). Securing IoT microservices with certificates. In NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium.

[19] Pahl, M.-O., & Aubet, F.-X. (2018). All eyes on you: Distributed multi-dimensional IoT microservice anomaly detection. In 2018 14th International Conference on Network and Service Management (CNSM).

[20] Sheridan, E. (2019). Microservices security: It gets worse before it gets better. WhiteHat Security. https://www.whitehatsec.com/blog/microservices-security/

[21] Backes, M., Dürmuth, M., Gerling, S., Pinkal, M., & Sporleder, C. (2010). Acoustic side-channel attacks on printers. In USENIX Security Symposium (pp. 307–322). https://www.usenix.org/legacy/events/sec10/tech/full_papers/Backes.pdf

[22] Ahmadvand, M., & Pretschner, A. (2018). Integrity protection against insiders in microservice-based infrastructures: From threats to a security framework. In Software Technologies: Applications and Foundations (pp. 19–34). https://doi.org/10.1007/978-3-030-04771-9_2