International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 05 May 2026
Page No: 3395-3401
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0438

Relix: A Comparative Latency Framework for REST and GRPC Using Redis Acceleration

G. Swapna[1], S. V. Punith Kumar[2], E.Susmitha[3]
[1,3]Assistant Professor, Department of Computer Science and Engineering, Rajiv Gandhi University of Knowledge Technologies RK Valley, Kadapa, Andhra Pradesh, India
[2]Student, Department of Computer Science and Engineering, Rajiv Gandhi University of Knowledge Technologies RK Valley, Kadapa, Andhra Pradesh, India
***Emails**: gswapna51@gmail.com[1], sangupunithkumar123@gmail.com[2], susmisuna@gmail.com[3]*

Abstract
Microservice-based architectures require efficient communication mechanisms to ensure high performance and scalability. REST, which relies on HTTP/1.1 and JSON, is widely adopted due to its simplicity but suffers from larger payload sizes and higher latency. In contrast, gRPC utilizes HTTP/2 and Protocol Buffers, enabling faster communication through binary serialization and persistent connections. This research presents a comparative latency analysis of REST and gRPC microservices enhanced with Redis caching. Both systems were implemented using identical business logic and evaluated under two scenarios: without caching (cold requests) and with caching (warm requests). Key performance metrics such as latency and payload size were measured and visualized. The experimental results demonstrate that gRPC consistently outperforms REST in terms of latency, stability, and payload efficiency. Redis caching significantly improves performance in both approaches, with gRPC showing greater benefits due to its compact data representation. The study concludes that gRPC is more suitable for high-performance microservice communication, especially in latency-sensitive applications.

Keywords *-Microservices, REST API, gRPC, Redis Caching, Latency Analysis, Protocol Buffers (Protobuf), JSON Serialization , Performance Optimization, Distributed Systems, API Communication*

1. Introduction

Modern software systems have transitioned from monolithic architectures to microservice-based designs to achieve better scalability, flexibility, and ease of deployment. In such architectures, efficient inter-service communication plays a critical role in determining overall system performance. Representational State Transfer (REST) has long been the dominant communication paradigm due to its simplicity, statelessness, and compatibility with web standards. However, REST relies on HTTP/1.1 and JSON serialization, which often introduce higher latency and larger payload sizes, making it less suitable for high-performance and real-time applications. To address these limitations, gRPC has emerged as a modern communication framework that utilizes HTTP/2 and Protocol Buffers (Protobuf). This combination enables faster data transmission, reduced payload size through binary serialization, multiplexed connections, and improved overall efficiency. As microservices continue to scale, the need to evaluate such communication protocols under real-world conditions becomes increasingly important. This research work is an extension of the previously published paper titled “Enhancing Microservice Performance: A Hybrid Approach Using Caching and Batching Techniques”. The earlier study focused on improving microservice performance by combining caching and batching strategies to reduce database load and enhance throughput. While significant improvements were achieved, the communication protocol itself remained an underlying factor influencing system latency and efficiency. Building upon that foundation, the present study shifts focus toward analyzing how different communication protocols—specifically REST and gRPC—impact performance when combined with caching mechanisms. Redis, an in-memory data store, is integrated into both architectures to evaluate its effect on reducing response time and database dependency. By

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 05 May 2026
Page No: 3395-3401
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0438

conducting controlled experiments on both cached (warm) and non-cached (cold) scenarios, this study aims to provide a comprehensive comparison of latency, payload size, and performance stability.

The primary objective of this research is to determine whether gRPC offers measurable advantages over REST in microservice communication, particularly in latency-sensitive environments. Additionally, the study seeks to understand how caching interacts with each protocol and whether the combination of efficient communication and caching can further enhance system performance [1-7]. Through empirical analysis and visualization, this work contributes to guiding developers and system architects in selecting the most appropriate communication strategy for modern distributed applications.

2. Research Aim

The primary aim of this research is to evaluate and compare the performance of REST and gRPC communication protocols in microservice architectures, with a focus on latency, payload efficiency, and the impact of Redis caching. The study aims to identify the most efficient protocol for high-performance, low-latency applications and to provide practical recommendations for optimizing microservice communication.

3. Research Objectives

- To implement REST and gRPC microservices with identical functionality for fair comparison.
- To integrate Redis caching into both REST and gRPC systems to optimize performance.
- To measure and compare latency for cached (warm) and non-cached (cold) requests.
- To analyze and compare payload sizes of JSON (REST) and Protobuf (gRPC).
- To evaluate the impact of caching on performance improvement in both protocols.
- To determine the most efficient and stable communication protocol for microservice architectures.

4. Research Questions

- How do REST and gRPC differ in terms of latency without caching?
- How does Redis caching improve the performance of REST and gRPC?
- Which communication protocol benefits more from caching?
- How do payload sizes differ between JSON and Protobuf, and how does this affect performance?
- Does gRPC consistently outperform REST in both cached and non-cached scenarios?
- Which protocol provides better stability and consistency under repeated requests?

5. Problem Statement

Microservice-based applications rely heavily on efficient inter-service communication to achieve high performance and scalability. REST, the most widely used communication approach, utilizes HTTP/1.1 and JSON serialization, which introduce significant overhead in terms of larger payload sizes, increased latency, and slower data processing. These limitations make REST less suitable for latency-sensitive and high-throughput systems. Although gRPC has emerged as a high-performance alternative using HTTP/2 and Protocol Buffers, its practical advantages over REST in real-world microservice environments are not always clearly quantified. Additionally, while caching mechanisms such as Redis are known to improve system performance by reducing database access time, their impact may vary depending on the communication protocol used. Therefore, the key problem addressed in this research is the lack of a comprehensive, experimental comparison between REST and gRPC in terms of latency, payload efficiency, and performance consistency, both with and without Redis caching. This creates uncertainty for developers and system architects in selecting the most suitable communication protocol for building efficient and scalable microservice-based applications [8-16].

6. Literature Review

Microservice architectures depend on efficient communication mechanisms to ensure scalability and performance. Various studies have explored different communication protocols and optimization techniques to address latency and data transfer challenges.According to Google Developers (2020), gRPC is a high-performance, open-source Remote Procedure Call (RPC) framework designed for

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 05 May 2026
Page No: 3395-3401
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0438

efficient communication between distributed systems. It leverages HTTP/2 for transport, enabling features such as multiplexing, header compression, and persistent connections. These features significantly reduce latency compared to traditional REST-based communication, especially in microservice environments where frequent inter-service communication is required. The efficiency of gRPC is further enhanced by Protocol Buffers, as described in the Google Protocol Buffers Developer Guide (2021). Protocol Buffers (Protobuf) provide a compact binary serialization format that is faster and more efficient than text-based formats like JSON. The guide highlights that Protobuf reduces payload size and improves serialization and deserialization speed, making it highly suitable for performance-critical systems. This advantage directly contributes to reduced network overhead and faster response times. A study by Kumar and Saha (2019) evaluated the performance of REST and gRPC in microservice architectures and concluded that gRPC consistently outperforms REST in terms of latency and throughput. The study attributes this improvement to binary serialization and the use of HTTP/2, which minimizes communication overhead. However, the authors also noted that REST remains popular due to its simplicity, flexibility, and widespread adoption.

Similarly, Zhang, Wang, and Chen (2020) conducted a comparative study of REST and gRPC APIs for cloud-native applications. Their findings indicate that gRPC provides lower latency and better performance stability, particularly under high-load conditions. The study also emphasizes that REST APIs, while easier to implement and debug, suffer from higher payload sizes and increased processing time due to JSON-based communication. In addition to protocol-level optimizations, performance enhancement techniques such as caching have been widely studied. The previously published work titled "Enhancing Microservice Performance: A Hybrid Approach Using Caching and Batching Techniques" serves as a foundational reference for this research. That study demonstrated that integrating caching and batching mechanisms significantly reduces database load and improves system throughput. However, it primarily focused on optimization techniques without deeply analyzing the role of communication protocols in performance improvement. Building upon these existing studies, the current research extends the analysis by combining protocol comparison (REST vs gRPC) with caching using Redis. While prior literature confirms the advantages of gRPC and the benefits of caching independently, there is limited work that evaluates their combined impact under identical experimental conditions. This research addresses that gap by providing a comprehensive comparison of latency, payload size, and performance consistency across both protocols with and without caching.

7. Research Methodology

This research adopts an experimental and quantitative approach to evaluate the performance of REST and gRPC communication protocols in a controlled microservice environment. Both protocols are implemented separately using identical business logic, database structure, and system configuration to ensure a fair and unbiased comparison.

The study is designed to compare the performance of REST and gRPC under two conditions:

- Without caching (cold requests) – data is retrieved directly from the database.
- With caching (warm requests) – data is served from Redis cache.

The methodology involves implementing two microservices (REST and gRPC), integrating Redis caching, executing repeated read operations, recording performance metrics, and analyzing the results. This structured experimental design ensures accurate evaluation of latency, efficiency, and payload optimization.

The system is developed using modern technologies suitable for microservice architecture:

- Backend: Spring Boot (REST), Java gRPC
- Cache Layer: Redis (in-memory data store)
- Database: MySQL
- ORM: Hibernate/JPA
- Frontend: React.js with Chart.js for visualization

These technologies provide a stable and efficient environment for measuring performance accurately.

The system consists of two parallel pipelines:

- REST Pipeline: Client → REST API → Redis Cache → MySQL → JSON response

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 05 May 2026
Page No: 3395-3401
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0438

- gRPC Pipeline: Client → gRPC Service →Redis Cache → MySQL → Protobuf response

Both pipelines share identical data models and business logic to ensure consistency. The architecture includes controller, service, repository, and cache layers, enabling systematic data flow and fair comparison. Experiments are conducted in a controlled local environment to eliminate external network variations:

- Processor: M4
- RAM: 16 GB
- Operating System: macOS
- Software: JDK 17, Spring Boot 3+, Redis Server, MySQL, React

This setup ensures that observed performance differences are due to protocol behavior rather than environmental factors.

The study focuses on the following key metrics:

- Latency (ms):
- Cold latency (database access)
- Warm latency (cache access)
- Payload Size (bytes):
- JSON size for REST
- Protobuf size for gRPC

Additional observations include cache hit/miss behavior and performance consistency across multiple executions.

The evaluation process follows a structured sequence:

- Populate sample data in the MySQL database.
- Execute REST API calls without caching and record latency and payload size.
- Execute REST API calls with Redis caching and measure warm latency.
- Execute gRPC calls without caching and record performance metrics.
- Execute gRPC calls with caching and measure warm performance.
- Store collected data for analysis, including latency and payload values.
- Visualize results using charts and graphs through a React-based dashboard.
- Analyze and compare results to determine performance differences.

This methodology provides a systematic and fair comparison of REST and gRPC by maintaining identical conditions and measuring key performance indicators. It enables accurate analysis of how communication protocols and caching mechanisms influence microservice performance.

Results and Discussion

A total of 20 observations were recorded for both REST and gRPC under each testing condition, namely cache-enabled (warm) and no-cache (cold) scenarios, to ensure consistency and reliability in performance evaluation.

8. Rest Performance Analysis

8.1.Cache-Enabled (Warm) Latency

The recorded latency values include initial warm-up overhead due to database initialization and cache population. Hence, the first two observations are excluded for steady-state analysis shown in Figure 1,2.

- Total observations: 20
- Warm-up values removed: 2
- Effective observations: 18

173.666, 15.794, 2.527, 1.413, 1.791, 2.221, 2.081, 2.428, 2.765, 1.503, 2.615, 1.116, 4.019, 2.614, 2.868, 1.448, 2.332, 2.669, 2.874, 1.094

Average (including warm-up) = 11.49ms

Average (warm only) = 2.24ms

- Minimum latency: 1.09 ms
- Maximum latency: 173.66 ms

8.2. No-Cache (Cold) Latency

All values represent direct database access without caching.

6.746, 7.159, 6.205, 7.038, 7.411, 4.122, 7.145, 7.933, 6.648, 5.020, 4.712, 6.555, 4.980, 3.999, 6.742, 5.639, 7.217, 7.075, 4.826, 7.318

Average latency=6.22ms

- Minimum latency: 3.99 ms
- Maximum latency: 7.93 ms

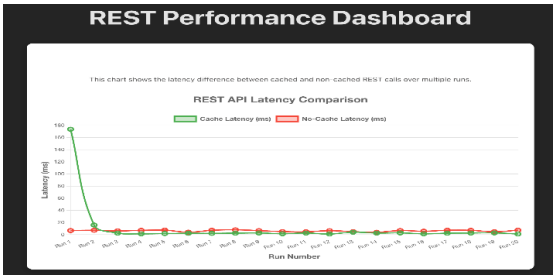

Figure 1 gRPC Performance Analysis

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 05 May 2026
Page No: 3395-3401
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0438

9. gRPC Performance Analysis

9.1.Cache-Enabled (Warm) Latency

Similar to REST, the first two values are excluded due to initialization overhead.

- Total observations: 20
- Warm-up values removed: 2
- Effective observations: 18

176.434, 27.785, 1.681, 1.740, 1.774, 1.629, 2.440, 1.593, 1.880, 0.986, 1.518, 1.277, 1.635, 1.762, 1.169, 1.628, 0.982, 1.721, 1.701, 1.670

Average (including warm-up) = 11.65ms Average (warm only) = 1.60ms

- Minimum latency: 0.98 ms
- Maximum latency: 176.43 ms

2.2 No-Cache (Cold) Latency

6.083, 6.144, 5.545, 4.335, 5.855, 5.531, 5.849, 5.729, 5.447, 5.135, 5.361, 4.464, 4.608, 4.808, 5.514, 5.046, 5.019, 5.545, 5.203, 5.236

Average latency=5.32ms

- Minimum latency: 4.33 ms
- Maximum latency: 6.14 ms

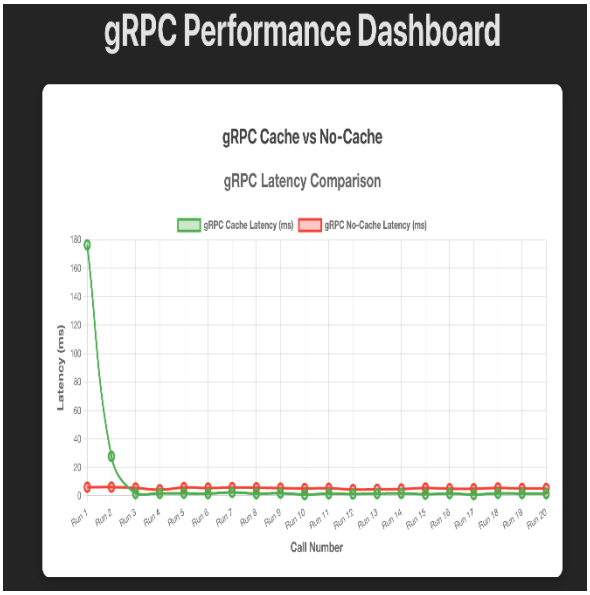

Figure 2 gRPC Performance Dashboard

10. Payload Size Comparison

- REST (JSON): 110 bytes
- gRPC (Protobuf): 80 bytes

Reduction=((110−80)/110))×100=27%

Thus, gRPC achieves approximately **27% smaller payload size** compared to REST.

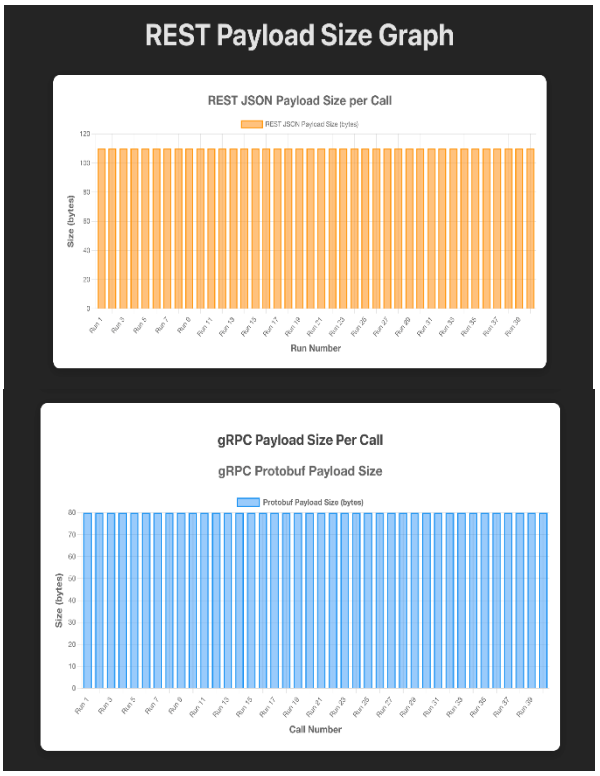

Figure 3 Payload size per call

11. Performance Improvement due to caching

REST Improvement

((6.22−2.24)/6.22) ×100=64%

gRPC Improvement

((5.32−1.60)/5.32) ×100=70%

12. Comparative Analysis

Table 1 Comparative Analysis

Metric	REST	gRPC
Cache Latency (Avg)	2.24 ms	1.60 ms
No-Cache Latency (Avg)	6.22 ms	5.32 ms
Payload Size	110 bytes	80 bytes
Cache Improvement	64%	70%

The results clearly indicate that:

- gRPC achieves lower latency in both cache and no-cache scenarios.
- gRPC provides better performance improvement with caching compared to REST.
- Smaller payload size in gRPC contributes to reduced transmission time and higher

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 05 May 2026
Page No: 3395-3401
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0438

efficiency.

- REST shows higher variability, whereas gRPC demonstrates more stable and consistent performance.

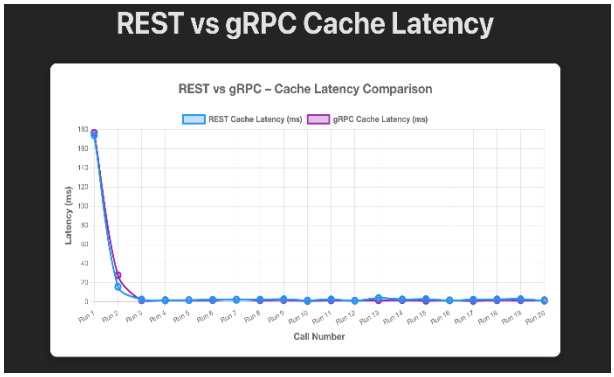

Figure 4 Cache Latency

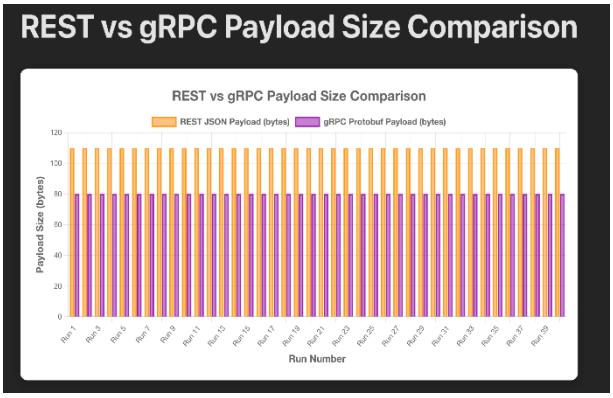

Figure 5 Payload Size Comparison

Conclusion

This research presented a comprehensive comparative analysis of REST and gRPC communication protocols in a microservice architecture, with a focus on latency, payload size, and the impact of Redis caching. A total of 20 observations were conducted for each scenario to ensure reliable and consistent evaluation.The experimental results clearly demonstrate that gRPC outperforms REST across all key performance metrics. In cache-enabled scenarios, gRPC achieved an average latency of approximately 1.60 ms, whereas REST recorded around 2.24 ms. Similarly, in no-cache conditions, gRPC maintained lower latency (~5.32 ms) compared to REST (~6.22 ms). This confirms that gRPC provides faster communication both with and without caching.

The study also revealed that Redis caching significantly improves performance for both protocols. REST showed an improvement of approximately 64%, while gRPC achieved around 70% improvement, indicating that gRPC benefits more effectively from caching mechanisms.In terms of payload efficiency, gRPC demonstrated a clear advantage with 27% smaller payload size (80 bytes) compared to REST (110 bytes). This reduction in data size contributes directly to faster transmission and lower processing overhead.Additionally, gRPC exhibited greater stability and consistency with lower latency variation (jitter), whereas REST showed relatively higher fluctuations. This makes gRPC more suitable for real-time and high-frequency communication in distributed systems.Overall, the findings of this study conclude that gRPC is a more efficient, faster, and reliable communication protocol than REST, especially in performance-critical microservice environments. While REST remains simple and widely adopted, gRPC is the preferred choice when low latency, optimized data transfer, and system scalability are of primary importance.

Future Scope

This study can be extended by evaluating REST and gRPC under high-concurrency conditions to analyze scalability and throughput. Future work may also include testing with security protocols such as HTTPS and TLS to measure their impact on performance. Deployment in containerized environments like Docker and Kubernetes can provide real-world insights. Additionally, analyzing CPU, memory, and network usage, as well as integrating advanced techniques like batching and asynchronous communication, can further enhance performance evaluation.

Acknowledgement

The authors thank Rajiv Gandhi University of Knowledge Technologies RK Valley for support.

Author Biographies

G. Swapna is a Lecturer in the Department of Computer Science and Engineering at RGUKT-RK Valley. With over nine years of teaching experience, she has developed a strong academic foundation and a commitment to student mentorship. Her primary research interests and areas of expertise include Software Engineering, Big Data, Computer

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 05 May 2026
Page No: 3395-3401
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0438

Networks, and Web Technologies. E. Susmitha is an Assistant Professor in the Department of Computer Science and Engineering at RGUKT-IIIT RK Valley and a Ph.D. scholar at JNTUA College of Engineering, Anantapur. With over nine years of teaching experience, she has built a distinguished academic record, including the publication of one book, eleven international journal papers, and one patent. In addition to presenting at an international conference, she has participated in more than 13 Faculty Development Programs (FDPs). A recipient of the 2016 Prathibha Award for her M.Tech, she has also successfully qualified for both UGC-NET and APSET. Her diverse research interests encompass Machine Learning, Artificial Intelligence, Big Data, mobile applications, web technologies, and the Internet of Things (IoT).

References

[1]. Fielding, R. T. (2000). Architectural Styles and the Design of Network-Based Software Architectures. University of California, Irvine.
[2]. Google Developers. (2020). gRPC: A High Performance, Open Source Universal RPC Framework. Retrieved from https://grpc.io
[3]. Google Protocol Buffers. (2021). Protocol Buffers: Developer Guide. Retrieved from https://developers.google.com/protocol-buffers
[4]. Microsoft Azure Architecture Center. (2020). REST vs gRPC for Microservices Communication. Retrieved from https://learn.microsoft.com/
[5]. Redis Labs. (2023). Redis Official Documentation. Retrieved from https://redis.io/docs/
[6]. Postman Engineering. (2021). Performance Comparison: JSON vs Protocol Buffers. Postman Engineering Blogs.
[7]. Kumar, A., & Saha, D. (2019). Performance Evaluation of REST and gRPC in Microservices Architecture. IEEE International Conference on Advanced Computing.
[8]. Apache Foundation. (2022). HTTP/2: Protocol Overview. Retrieved from https://http2.github.io/
[9]. Newman, S. (2015). Building Microservices: Designing Fine-Grained Systems. O'Reilly Media.
[10]. Fowler, M. (2019). Microservices and Distributed Systems. martinfowler.com
[11]. Oracle Java Documentation. (2023). Spring Boot and JPA Performance Guidelines. Oracle Corporation.
[12]. MySQL Documentation. (2023). MySQL Query Optimization and Caching. Retrieved from https://dev.mysql.com/doc/
[13]. Zhang, Q., Wang, L., & Chen, W. (2020). Comparative Study of REST and gRPC APIs for Cloud Native Applications. Journal of Cloud Computing.
[14]. Nginx Performance Blog. (2020). Impact of HTTP/2 on Microservice Latency. Nginx Inc.
[15]. Verma, R., & Sharma, S. (2021). High-Performance Serialization Techniques in Distributed Systems. International Journal of Computer Applications.
[16]. G. Swapna, E. Susmitha, N. Satyanandaram, and S. V. Punith Kumar, "Enhancing Microservice Performance: A Hybrid Approach Using Caching and Batching Techniques," International Research Journal on Advanced Engineering Hub (IRJAEH), vol. 4, no. 2, pp. 766–775, 2026.