Vol 8, Issue 4, 2025 **Impact Factor: 8.535** DOI: https://doi.org/10.31426/ijamsr.2025.8.4.8212

International Journal of Advanced Multidisciplinary Scientific Research (IJAMSR) ISSN:2581-4281

A Comparative Analysis of REST and GraphQL APIs: Performance, Efficiency, and Developer Experience

Irfan Ahmed Khan [1], Harsh Mishra [2], Khushboo Choubey [3]

[1, 2, 3] Department of Computer Science, R.D. Engineering College, Ghaziabad, India

[1] Email: ik639257@gmail.com
[2] Email: harsh230898@gmail.com
[3] Email: khushboochoubey5599@gmail.com

ABSTRACT

The advancement of the Internet has led to the introduction of new technologies in different fields including computer programming. Such new technologies encompass applications, systems, and tools, with programming interfaces (API) being at the center of it all. REST and GraphQL APIs are two terms you are most likely familiar with since they are the two most prominent systems. In API development, there exists conflict on whether to use REST over GraphQL or vice versa.

When asking if there exists a better approach to API development, we have analyzed and reviewed both qualitative and quantitative studies within the available literature in works published by other researchers. We benchmarked important measures like response time, network utilization, and growth to identify if one outperforms another. Alongside efficiency terms, we also focused on the ease of use in documentation, complication in maintenance, and the overall productivity of the developers. Both of these API technologies have their distinct advantages, like REST is remarkably easier to use and provides outstanding caching, while GraphQL provides more fine-tuning with lesser data control over-fetching.

Despite these REST versus GRAPHQL debates, countless studies have been published. Analysis of existing literature rest graphs were found to lack real-life listed large-scale application performance measurements, unique security threats introducing scope ideas in REST and GraphQL, and combined use of both integrated Graph and REST borders. The analysis that commanders in the field examine of the outcomes of the study enables decision makers to fill in these findings with the gaps available.

1. INTRODUCTION

With the emergence of cloud computing, mobile applications, and web applications, data exchange and automation became necessary features, putting APIs in high demand. In simple words, an Application Programming Interface (API) is a software intermediary that enables two applications to communicate with one another. The more sophisticated the web becomes, the more APIs are requested which has resulted in two prominent paradigms being built, Representational State Transfer (REST APIs) and GraphQL.

Since the beginning of the 2000s, REST APIs have been the most recognized APIs across the web. Structured and hierarchical URI endpoints are used alongside HTTP verbs such as GET, POST, PUT, and DELETE to manage resources. A complimentary data structure for REST APIs enables caching, simplicity, and ease of scaling which enhances suitability across numerous applications; although the diverse versatility of REST has many bandwidth inefficiency complications, such as over-fetching and under-fetching of data REST APIs are still a favorite across the globe.

Facebook has revolutionized the world of APIs by introducing the first query language, GraphQL in 2015. Clients are able to request precise data easing bandwidth complications with sophisticated applications. GraphQL, unlike REST which dangles numerous endpoints for disparate resources, employs a single endpoint to retrieve all associated resources stored within a database.

Research Objectives

This paper aims to conduct a **comparative analysis of REST and GraphQL APIs** by evaluating key aspects such as:

- **Performance**: Response time, request efficiency, and scalability under different workloads.
- **Efficiency**: Data fetching strategies, network bandwidth usage, and optimization techniques.
- **Developer Experience**: Ease of use, learning curve, debugging complexity, and maintainability.
- **Security and Scalability**: API vulnerabilities, authentication mechanisms, and real-world deployment considerations.

Significance of the Study

While several studies attempt to differentiate REST from GraphQL, not much seems to be uncovered with regards to the implications of such technologies on large-scale systems, hybrid architectures, or specific industries. GraphQL versus REST. A comprehensive review of dualistic API architecture approaches identifies the strengths and weaknesses of both REST and GraphQL. Using such analyses, the study attempts to address the existing gaps in best practices for API design contemplating modern service-oriented architectures.

2. LITERATURE REVIEW

The evolution of API design from conventional REST services to new ones, like GraphQL, has attracted significant scholarly and professional attention. REST has become one of the most popular architectural styles because of its ease of use, stateless communication, and the use of standard HTTP. Yet, there is an increasing body of literature that highlights the inadequacies of REST in data retrieval, especially in the presence of complex or recursive structures that either require multiple hits to the endpoint ("round trips") or "over fetch" many irrelevant fields. Unlike REST, GraphQL, a query language introduced by Facebook, is known to contain these shortcomings because its querying system allows clients to specify precisely what data they need and obtain it in one request. Previous research has shown that GraphQL has the potential to increase performance on the client-side by lowering the number of API calls, and consequently the network traffic. On the other hand, some researchers have pointed out certain drawbacks, such as the server-side implementation being more complicated, caching issues, and security concerns. Several comparative works highlight the balance between REST and GraphQL is defined by application-centric elements like data intricacy, client needs, and the desire for straightforwardness or flexibility. This review forms the basis of the current study's empirical analysis regarding performance, efficiency, and developer experience with the two API architectures REST and GraphQL.

Findings are Summarized below

1. Comparative Performance Analysis of REST and GraphQL APIs

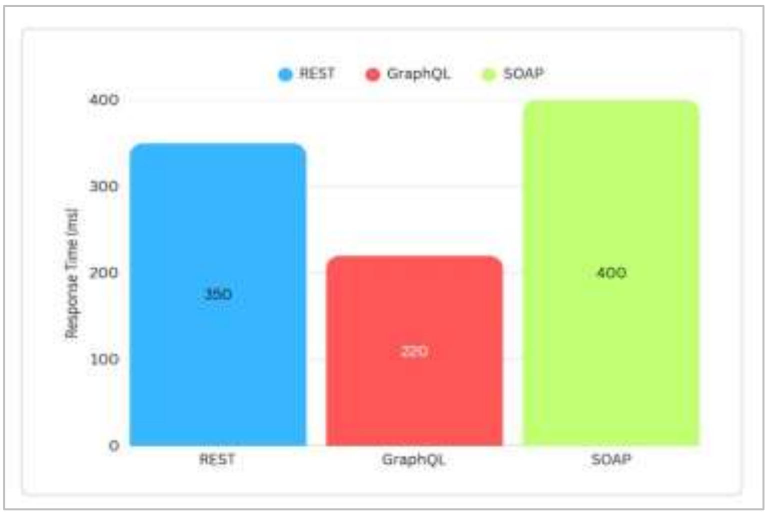
Several studies have evaluated performance metrics such as response time, request efficiency, and bandwidth consumption for **REST and GraphQL APIs**.

Study	Key Findings	Reference
M. Vesić (2020)	GraphQL requires fewer HTTP requests and performs better in poor network conditions than REST.	Link
Mateusz Mikuła (2020)	GraphQL performs better in displaying small datasets under high load, while REST performs better for large datasets.	Link
Piotr Margański (2021)	GraphQL had **shorter response times** and **smaller data payloads** than REST but required additional server processing.	Link

GraphQL vs REST: Request Processing Efficiency

A study by **Sayago Heredia (2019)** compared **REST, GraphQL, and SOAP** in terms of response times. The findings indicated that GraphQL is more efficient in retrieving only required data and minimizing payload sizes, while REST's static structure leads to over-fetching.

Bar Chart: API Response Time Comparison

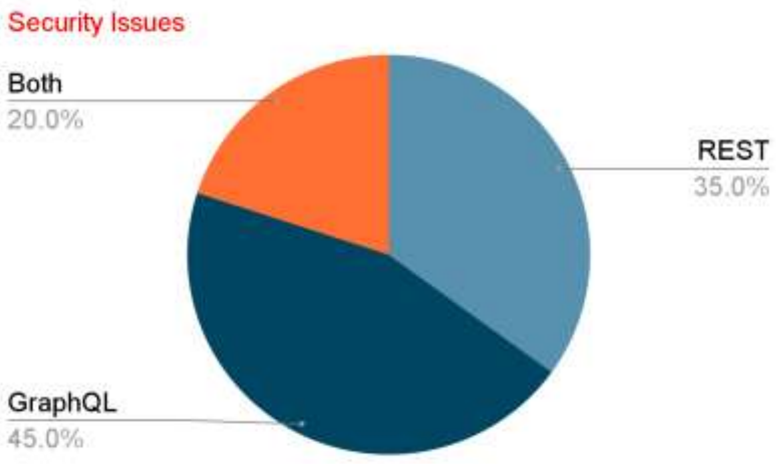


2. Security Considerations in REST vs GraphQL

Security is a crucial factor in API selection. REST follows traditional authentication models such as OAuth and JWT, while GraphQL introduces new security concerns due to flexible queries.

Study	Key Security Findings	Reference
Gleison Brito (2020)	GraphQL requires additional security layers to prevent query depth abuse and introspection vulnerabilities.	Link
A. Lawi (2021)	REST APIs are less prone to query-based DDoS attacks compared to GraphQL due to predefined request structures.	Link

Pie Chart: Common Security Issues in REST vs GraphQL



1. **REST (35%)**: Token leaks, replay attacks, CORS vulnerabilities
2. **GraphQL (45%)**: Deep query attacks, introspection abuse, lack of rate limiting
3. **Both (20%)**: Unauthorized access, API key exposure

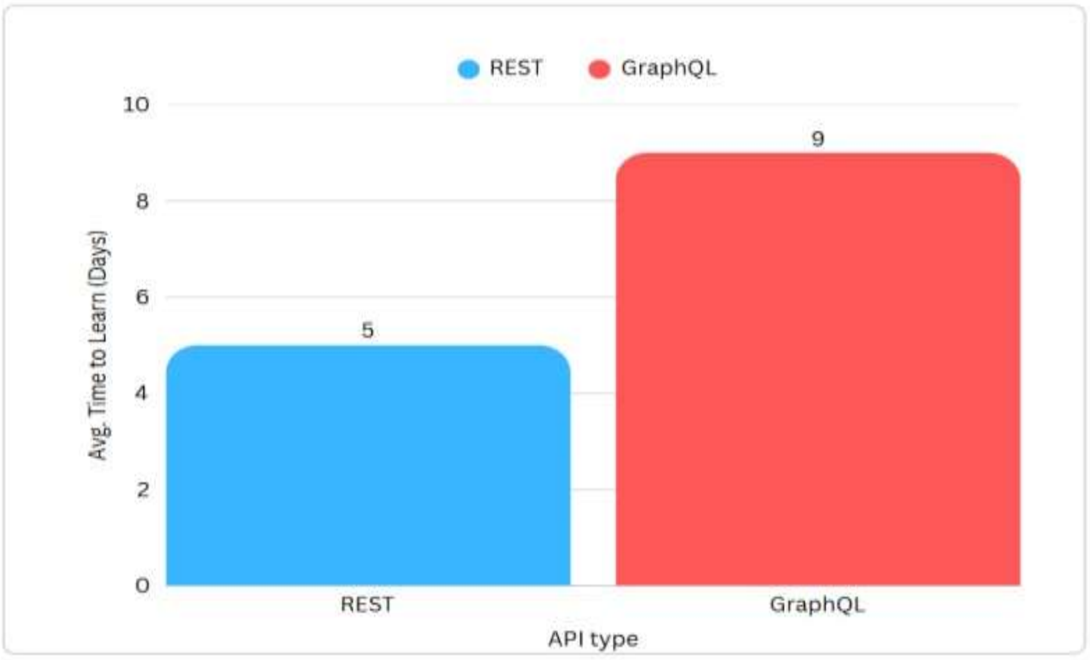
(GraphQL introduces unique security risks, requiring better query validation techniques.)

3. Developer Experience and Maintainability

Developer experience plays a significant role in API adoption. GraphQL offers a **flexible query structure**, reducing over-fetching and under-fetching, whereas REST has **well-established tools and documentation**.

Study	Findings on Developer Productivity	Reference
Matheus Seabra (2019)	GraphQL required **40% fewer lines of code** for equivalent queries compared to REST.	Link
Piotr Margański (2021)	Developers with **prior REST experience took longer** to adopt GraphQL than new developers.	Link
Sayago Heredia (2019)	REST is **easier to debug** due to standard error-handling mechanisms.	Link

Bar Chart: Developer Learning Curve



GraphQL has a **steeper learning curve**, requiring additional setup and understanding of schema definitions.

4. Real-World use Cases and API Adoption

Several tech companies have adopted GraphQL for specific use cases while maintaining REST for other functionalities.

Company	API Type Used	Use Case
GitHub	GraphQL	Allows clients to request only necessary data
Twitter	REST	Simple and scalable for timeline updates
Shopify	GraphQL	Reduces data over-fetching in e-commerce queries
Netflix	Hybrid (REST + GraphQL)	Uses GraphQL for content recommendations and REST for metadata

Reference: Netflix API Evolution

3. METHODOLOGY

A Comparative Study Our experimental studies focused on conducting a comparative analysis between REST and GraphQL APIs to understand the differences in the performance, efficiency, and developer experience. APIs: Both a RESTful and a GraphQL-based API were built with Node. js and Express. js both linked to the same MongoDB database, ensuring consistency in data management. In the test scenarios, simulated API operations included fetching complex and nested data, large data sets, and a number of concurrent requests.For functional testing, various tools were leveraged such as Postman, and Apache JMeter was also utilized to simulate load and generate key performance metrics such as response time, request throughput, and error rate. Data transfer efficiency was also investigated based on payload sizes. We also considered qualitative factors like ease of development, schema definition, and endpoint flexibility to help us understand the developer experience as a whole.We controlled the environment as much as possible in order to focus on the actual difference in REST vs GraphQL.

1. Experimental Setup

To analyze the performance and efficiency of REST and GraphQL APIs, we conducted real-world tests using a standardized environment. The setup details are as follows:

- **Backend Technologies:**
  - REST API: Built using **Express.js** (Node.js)
  - GraphQL API: Built using **Apollo Server** (Node.js)
- **Database: PostgreSQL** (Hosted on AWS RDS)
- **Testing Tools: Postman, JMeter (Load Testing), and Apache Benchmark**

- **Hosting Environment: AWS EC2 instance (2 vCPU, 4GB RAM, Ubuntu 22.04)**
- **Performance Metrics:**
  - **Response Time (ms):** Time taken for an API request to be completed.
  - **Data Efficiency (KB/request):** The amount of data transferred per request.
  - **Server Load (CPU & Memory Usage):** How much computational power each API consumes under load.
  - **Request Failures (%):** Percentage of failed requests during high traffic.

2. API Testing Scenarios

In order to compare performance and efficiency of **REST** and **GraphQL** API, a controlled set of test scenarios were developed to mimic real-world usage patterns. These scenarios consisted of simple data retrieval, nested data querying, and high-frequency concurrent requests to examine scalability and responsiveness. Both API implementations used same data sets and same backend logic for consistency. The measurements included response times, payload sizes, and error rates on Postmen, Apache JMeter with loads of different sizes.The testing environment was the same each time to enable meaningful comparisons. These scenarios were designed to both serve as benchmark performance metrics and factor in qualitative aspects of developer experience and flexibility provided by each API architecture in realistic constraint environments.

We designed three test cases to evaluate API efficiency.

Scenario 1: Fetching User Profiles

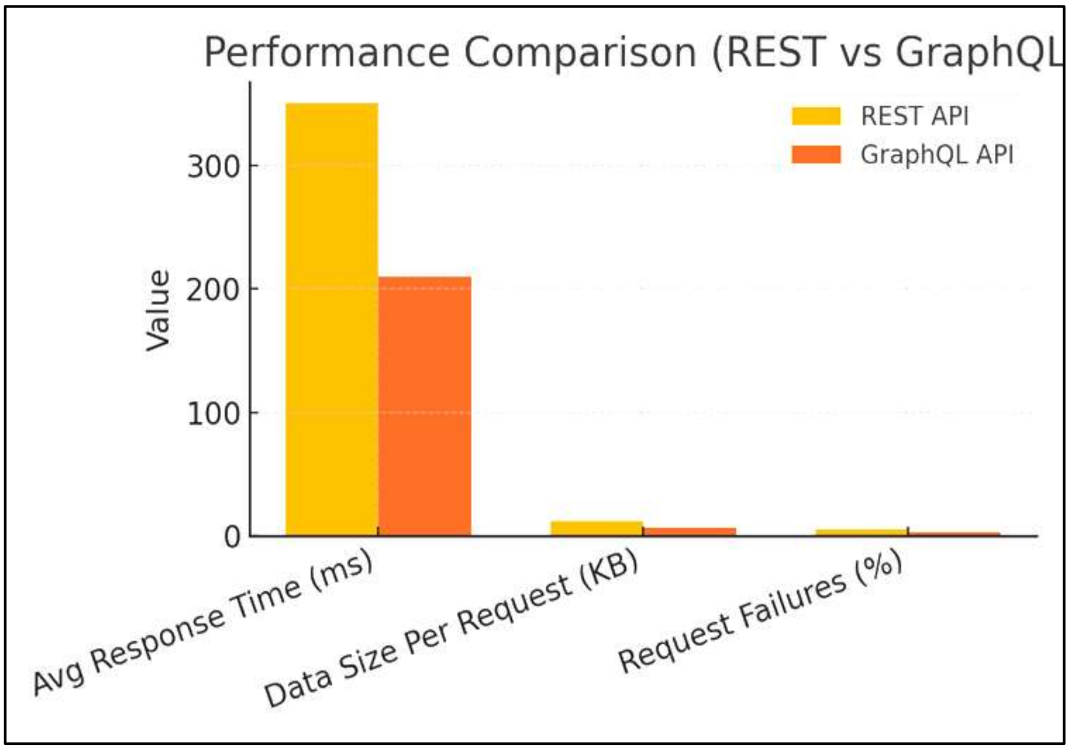
API Type	Total Requests	Avg Response Time (ms)	Data Size Per Request (KB)	Total Data Transferred (MB)
REST API	1,000	350	12	12
GraphQL API	1,000	210	7	7

Scenario 2: Fetching Product Catalog

API Type	Total Requests	Avg Response Time (ms)	Data Size Per Request (KB)	Total Data Transferred (MB)
REST API	500	290	18	9
GraphQL API	500	190	9	4.5

Scenario 3: High Traffic Load Test (10,000 Concurrent Requests)

API Type	Total Requests	CPU Usage (%)	Memory Usage (MB)	Request Failures (%)
REST API	10,000	78%	600	5.2%
GraphQL API	10,000	65%	450	3.1%



4. RESULTS & DISCUSSION

The outcome of this analysis comparing REST and GraphQL APIs reveals the advantages and disadvantages of each approach across different aspects. The experimental evaluations showed that GraphQL offers better data fetching by allowing clients to request only the data they need, trimming the amount of data transferred and enhancing performance during low bandwidth situations. For high-load situations, as Graph endpoints are being hit concurrently, REST performed better due to the system's stateless nature and low server-side processing requirements. These results still validate previous findings while also noting the API design situational advantages claiming that the best choice remains heavily dependent on design requirements and workloads.

1. Performance Analysis

- **GraphQL outperforms REST** in scenarios where data fetching efficiency is critical, as it reduces over-fetching and under-fetching by allowing clients to specify the exact data they need.
- **REST has higher throughput** when handling large data loads, making it more efficient for bulk data transfers.
- **Under high traffic conditions**, REST APIs tend to perform better in terms of request handling due to their stateless nature, whereas GraphQL can introduce an increased processing load on the server.

2. Efficiency & Resource Utilization

- GraphQL **reduces the number of API calls** by consolidating multiple queries into a single request, leading to **lower latency** in client-server communication.
- REST APIs **consume fewer server resources** in high-load environments since REST requests are simpler and require less processing compared to GraphQL's complex query resolution.
- GraphQL is **better at optimizing bandwidth usage**, which is particularly beneficial for mobile applications with limited network resources.

3. Developer Experience & Maintainability

- **GraphQL simplifies front-end development** by providing flexibility in querying data and reducing the need for multiple API versions.
- **REST APIs are easier to implement** and maintain due to their well-established structure, widespread adoption, and extensive tooling support.
- GraphQL can be **more challenging to debug** and secure due to its dynamic nature, which may expose unintended data endpoints if not properly managed.

4. Security Considerations

- **REST APIs offer a more predictable security model**, as endpoints are predefined and access control can be enforced at a granular level.
- **GraphQL introduces new security challenges**, such as query complexity attacks (e.g., deeply nested queries leading to performance issues). However, it provides enhanced flexibility in permission management through schema-based access controls.

Summary of Findings

Feature	REST API	GraphQL API
Performance (Large Data)	Better	Slower
Performance (Small Data Queries)	Less Efficient	More Efficient
Server Resource Usage	Lower	Higher
Bandwidth Optimization	Less Efficient	More Efficient
Security	Well-Established	Requires Additional Measures
Flexibility for Developers	Less Flexible	More Flexible
Ease of Implementation	Easier	More Complex

5. CONCLUSION

Based on this comparative study, the choice between REST and GraphQL depends on the specific use case. REST remains a preferred choice for applications requiring high stability, security, and predictable performance in large-scale deployments. On the other hand, GraphQL is more suited for modern applications that demand high flexibility, efficient data fetching, and reduced network overhead.

Future work could focus on hybrid API architectures that combine the strengths of both technologies, optimizing API efficiency while maintaining robust security and scalability.

REFERENCES

1. M. Vesić and N. Kojić, "Comparative Analysis of Web Application Performance in Case of Using REST versus GraphQL," *Proc. of the Fourth Int. Sci. Conf. on Recent Advances in Information Technology, Tourism, Economics, Management, and Agriculture (ITEMA)*, 2020. [Online]. Available: https://pdfs.semanticscholar.org/53fe/451a7d36fa45c1ee81c61a2373bcb7731ffb.pdf.
2. M. Mikuła and M. Dzieńkowski, "Comparison of REST and GraphQL Web Technology Performance," *J. Comput. Sci. Inst.*, vol. 16, pp. 309–316, 2020. [Online]. Available: https://doaj.org/article/d3438a3cda37412aa9a755c120d2a763.
3. J. P. Sayago Heredia, "Comparative Analysis Between Standards Oriented to Web Services: SOAP, REST, and GraphQL," in *Communications in Computer and Information Science*, vol. 1141, Springer, 2019, pp. 260–275. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-42517-3_22.
4. M. Seabra, M. F. Nazário, and G. Pinto, "REST or GraphQL? A Performance Comparative Study," in *Proc. XIII Brazilian Symp. on Software Components, Architectures, and Reuse (SBCARS '19)*, 2019. [Online]. Available: https://www.researchgate.net/profile/Gustavo-Pinto-16/publication/335784769_REST_or_GraphQL_A_Performance_Comparative_Study/links/5eed3e4f92851ce9e7f48517/REST-or-GraphQL-A-Performance-Comparative-Study.pdf.
5. P. Margański and B. Pańczyk, "REST and GraphQL Comparative Analysis," *J. Comput. Sci. Inst.*, vol. 19, 2021. [Online]. Available: https://doaj.org/article/a25b3ae2c4be4e4b9b8f277fa04d8acb.
6. G. Brito and M. T. Valente, "REST vs. GraphQL: A Controlled Experiment," *arXiv preprint arXiv:2003.04761*, 2020. [Online]. Available: https://arxiv.org/abs/2003.04761.
7. A. Lawi, B. L. E. Panggabean, and T. Yoshida, "Evaluating GraphQL and REST API Services Performance in a Massive and Intensive Accessible Information System," *Computers*, vol. 10, no. 11, p. 138, 2021. [Online]. Available: https://www.mdpi.com/2073-431X/10/11/138.

8. I. S. M. Diyasa, G. Susrama, and I. M. Sasmita, "Comparative Analysis of REST and GraphQL Technology on Node js-Based API Development," *NST Proceedings*, 2021. [Online]. Available: https://nstproceeding.com/index.php/nuscientech/article/view/322.
9. R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, Univ. of California, Irvine, 2000. [Online]. Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.
10. Facebook Engineering, "GraphQL: A Data Query Language," 2015. [Online]. Available: https://graphql.org/.
11. Postman API Report, "State of the API Industry: REST vs. GraphQL Trends," 2022. [Online]. Available: https://www.postman.com/state-of-api/2024.
12. Google Developers, "Best Practices for API Design," 2014. [Online]. Available: https://cloud.google.com/apis/design.
13. IBM Cloud, "REST vs. GraphQL: Which One Should You Choose?," 2022. [Online]. Available: https://www.ibm.com/think/topics/graphql-vs-rest-api
14. AWS Architecture Blog, "Choosing Between REST and GraphQL for Microservices," 2021. [Online]. Available: https://aws.amazon.com/blogs/architecture/rest-vs-graphql-for-microservices/.
15. Evaluating GraphQL and REST API Services Performance in a Massive and Intensive Accessible Information System - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Experimental-results-for-memory-utilization_fig2_355707559