



Quality of Service Performance Analysis of URL-to-Google Drive Transfer: Colab Runtime Versus Browser Local Transit

Dicky Miswardi^{1*}, Abdul Hamid Arribathi²

^{1,2} Magister Teknik Informatika, Fakultas Sains dan Teknologi Universitas Raharja, Tangerang, Indonesia

E-Mail: ¹dicky.miswardi@raharja.info, ²abdulhamid@raharja.info

Received May 08th 2026; Revised Jun 17th 2026; Accepted Aug 17th 2026; Available Online Oct 30th 2026

Corresponding Author: Dicky Miswardi

Copyright © 2026 by Authors, Published by Institut Riset dan Publikasi Indonesia (IRPI)

Abstract

URL-to-Google Drive transfer via a browser may be inefficient because the file must pass through the local client twice: first during download, then during upload. This study evaluates the Quality of Service (QoS) performance of two transfer architectures: browser local transit and Google Colab runtime transfer. The browser local transit method is defined as a two-stage workflow in which a 100 MB file is downloaded from a URL to local storage using Google Chrome and then uploaded to Google Drive. The Colab method executes the transfer within a cloud runtime, while the local computer serves as the control interface. Each method was tested in five repeated trials under the same cellular hotspot environment. The measured parameters included active transfer duration, payload throughput, local traffic consumption, local quota efficiency, checksum validation, and statistical significance. The browser method recorded an average duration of 233.693 seconds, throughput of 0.429 MB/s, and local traffic of 211.365 MB. The Colab method completed the transfer in 4.830 seconds, achieved 20.841 MB/s, and consumed only 0.610 MB of local traffic. The Colab method reduced time by 97.933% and local traffic by 99.711%.

Keywords: Browser Local Transit, Colab Runtime, Google Drive, QoS, URL Transfer

1. INTRODUCTION

Cloud storage has become an important infrastructure for storing, accessing, and sharing digital files across devices. Google Drive is widely used for its online storage, file synchronization, and integration with the Google Workspace ecosystem. However, the efficiency of cloud storage is not determined only by storage capacity or accessibility [1]. It is also affected by the transfer architecture used to move files into the cloud, which requires a standard taxonomy of quality metrics to ensure optimal performance [2].

A common problem occurs when users need to move a file from a direct Uniform Resource Locator (URL) source to Google Drive. In a browser-based workflow, the file must first be downloaded from the URL to the local computer and then uploaded again from local storage to Google Drive. This workflow is simple, but it creates a local transit path that imposes significant storage and bandwidth costs on the intermediary. The file payload passes through the user's local network twice, creating unnecessary file input/output (I/O) overhead [3], which increases active transfer time and consumes local internet quota. This problem becomes more relevant for users who rely on cellular hotspots, limited bandwidth, or quota-based internet access.

In this study, the conventional method is defined specifically as a browser local transit method. It refers to a two-stage URL-to-Google Drive transfer process: downloading a 100 MB file from a URL to local storage using Google Chrome and then uploading the same file from local storage to Google Drive. This definition is used to avoid ambiguity in performance characteristics across different cloud storage access protocols [4], because the term conventional method can otherwise refer to many different manual workflows that are often less efficient than dedicated agent-based interfaces [5].

An alternative transfer architecture is to execute the file transfer inside a cloud runtime environment. Google Colab provides a browser-accessible notebook environment that can execute Python code on cloud infrastructure, enabling very low local payload traffic, which is useful for users with limited local bandwidth or quota-based internet access [6]. In this model, the local computer is used mainly as the control interface, while the file payload is handled by the Colab runtime. This creates a clear separation between the control plane, which involves user commands and interface communication, and the data plane, which involves the actual movement of the file payload.

Previous studies have discussed cloud storage, Google Drive usage, Python automation, cloud API integration, data transfer optimization, and network Quality of Service. However, many studies focus on storage features, computation, or overall network performance rather than directly comparing browser-local transit and Google Colab runtime transfer in the same measured environment. In particular, few studies quantify how much local traffic can be reduced when the data plane is shifted from the local client to a cloud runtime, a decoupling that is essential for ensuring performance isolation in modern software-defined storage [7], [8].

Based on this gap, the problem addressed in this study is architectural rather than merely programmatic. The study does not only ask whether Python can transfer a file faster than a browser. It investigates whether a cloud-runtime-based transfer architecture can reduce active transfer duration, increase payload throughput, and minimize local traffic consumption compared with browser local transit, leveraging efficient framework principles to reduce operational overhead [9], [10]. The scientific contribution of this study is the integration of QoS measurement, repeated trials, local traffic monitoring, checksum validation, local quota efficiency, and statistical testing in a URL-to-Google Drive transfer experiment.

This study aims to compare the QoS performance of browser local transit and Google Colab runtime transfer. The measured parameters include active transfer duration, payload throughput, local traffic consumption, local quota efficiency, and SHA-256 checksum validation, which serves as a reliable industry standard for verifying data integrity [11], and statistical significance. Despite the small sample size (N=5), the analysis remains statistically valid for large observed effect sizes. The results are expected to provide empirical guidance for selecting a more efficient transfer architecture for users who need to move URL-based files to Google Drive while minimizing local bandwidth and quota usage.

2. MATERIALS AND METHOD

This research employs a quantitative experimental method to compare two URL-to-Google Drive transfer architectures: the browser-based local transit method and the Google Colab runtime-based method. This approach aligns with the taxonomy of cloud service quality metrics that emphasizes standardized performance measurement [2]. Each method was tested five times (N=5) using the same 100 MB file (104,857,600 bytes), identical source URLs, the same Google Drive destination, and a consistent cellular hotspot environment. The measured parameters include active transfer duration, payload throughput, local traffic consumption, local quota efficiency, SHA-256 checksum validity, and statistical significance to comprehensively evaluate the Quality of Service in file-sharing environments [12].

2.1. Experimental Topology

The first topology represents the browser-based local transit method. In this architecture, the file is downloaded from the source URL to the client computer via Google Chrome and subsequently uploaded from local storage to Google Drive. Consequently, the client computer becomes an integral part of the data plane, as the file payload traverses the local hotspot connection during both the download and upload phases. Understanding the separation of these data paths is essential for identifying how local I/O can become a bottleneck during large-scale data transfers [3].

The second topology represents the Google Colab runtime-based method. In this approach, the researcher accesses Google Colab through a browser, mounts Google Drive, and executes a Python script within the Colab runtime. The local computer functions solely as a control interface, while the file payload is transferred directly through the cloud runtime. Local traffic is monitored to verify whether the client carries the full payload or only control-plane traffic. This architecture implements the principle of decoupling the control plane from the data plane to optimize QoS provisioning and performance isolation in modern storage systems [9], [8].

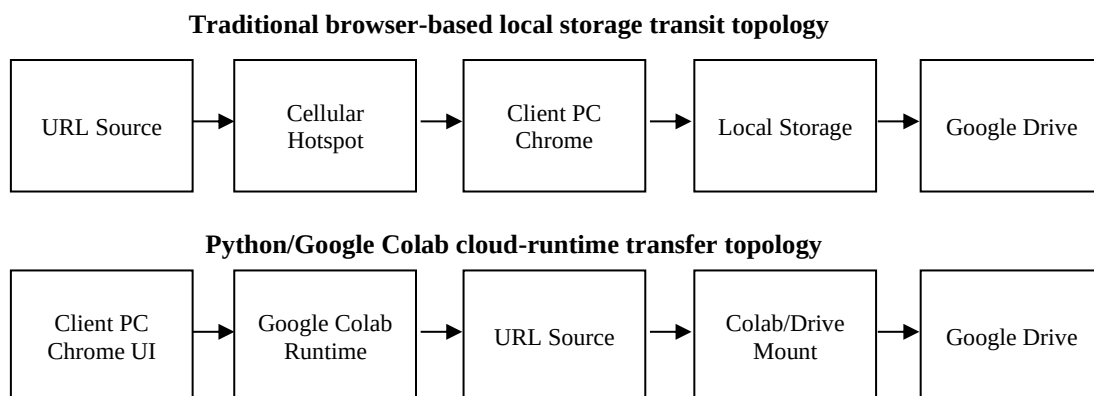


Figure 1. Experimental topology of browser local transit and Google Colab runtime transfer

2.2. Experimental Environment

The experiments were conducted using a Windows client computer connected to the internet via a cellular mobile hotspot. The hotspot device was positioned approximately 50 cm from the client computer to maintain consistent signal conditions using a 2.4 GHz network. Prior to the transfer tests, network conditions were verified using a baseline ping test, which recorded 0% packet loss and an average latency ranging from 51 to 59 ms. Utilizing a ping test as a network baseline is crucial for identifying the impact of communication latency on the energy consumption and performance of terminal nodes in cellular networks [13]. Packet-level loss during the actual file transfer was not directly captured and is noted as a limitation of the study.

Table 1. Experimental Environment

No	Component	Specification
1	Client computer	HP 280 G4 MT Business PC; Intel Core i7-8700; 15.83 GB RAM; Windows 11 Pro 64-bit
2	Browser and monitoring tool	Google Chrome 149.0.7827.103; Windows PowerShell network counter script
3	Local network	TP-Link Wireless N PCI Express Adapter; Wi-Fi 4 / 802.11n; WPA2-Personal; 2.4 GHz; channel 11
4	Hotspot and cellular network	Samsung Galaxy A15 SM-A155F/DSN; Android 16; 4G network; 3-SinyalKuatLuas provider; distance 50 cm
5	Test file and destination	100Mb.dat; 104,857,600 bytes; source URL: https://proof.ovh.net/files/100Mb.dat ; destination: Google Drive
6	Cloud runtime and analysis tools	Google Colab; Google Compute Engine Python 3; Python, Pandas, Matplotlib, CSV, JSON, SHA-256 hashing

Before transfer testing, the network condition was checked using a baseline ping test. The recorded baseline showed 0% packet loss and average latency of approximately 51–59 ms. Packet-level loss during file transfer was not captured directly and is treated as a study limitation.

2.3. Transfer Procedures

The browser-based local transit procedure involved two active phases. First, the 100 MB file was downloaded from the source URL to the local "Downloads" directory using Google Chrome. Second, the same file was uploaded from local storage to Google Drive. PowerShell was utilized to monitor bytes received and bytes sent through the active wireless adapter. The preparation gap between the end of the download and the start of the upload was recorded separately but excluded from the active transfer duration, as it represented user activity rather than network transfer time.

The Google Colab procedure was executed via a prepared notebook. The researcher mounted Google Drive, executed the Python transfer script, recorded start and stop timestamps, and monitored chunk-level throughput while exporting results in CSV and JSON formats. During execution, PowerShell monitored local client traffic to verify the architectural separation between the control plane and the data plane. This use of automated scripts and specialized software transfer tools is part of the Data Transfer Nodes infrastructure designed to accelerate big data movement between research networks and cloud storage [14]. This monitoring methodology is further supported by the use of measurement tools capable of regulating transfer rates and monitoring communication sessions in parallel [15].

2.4. Research Workflow

The research workflow consisted of environment preparation, browser local transit trials, Google Colab transfer trials, file validation, QoS calculation, descriptive statistics, and statistical significance testing.

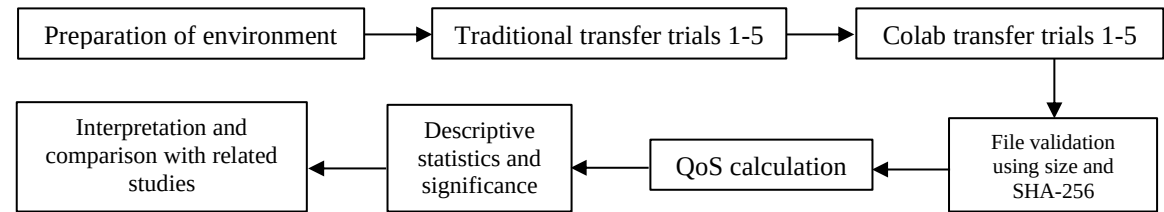


Figure 2. Research workflow of the URL-to-Google Drive transfer experiment

2.5. QoS Parameters and Data Validation

QoS evaluation was adapted for file transfer measurement. Since this study did not assess real-time audio or video communication, the primary QoS parameters selected were active transfer duration, average payload throughput, local traffic consumption, local quota efficiency, throughput improvement, and

measurement consistency. File integrity was validated using file size and SHA-256 checksums to ensure that the files remained unaltered and free from corruption during the transfer process [16], [17]. SHA-256 is widely recognized as a reliable industry standard for ensuring the authentication and validity of source files in cloud storage environments [18], [19].

Statistical significance was evaluated using a Paired t-test to compare the two methods across paired samples. Although the sample size was small ($N = 5$) the use of the paired t-test remains supporting statistical test and feasible for identifying large effect sizes in paired observations. Additionally, the Wilcoxon signed-rank test was utilized as a non-parametric support to ensure the reliability of benchmarking results for data that may not follow a normal distribution [20].

Table 2. QoS Formula and Description

No	Formula	Equation	Description
1	$D_{active} = D_{sdownload} + D_{supload}$	(1)	Active transfer duration for browser local transit, calculated from the sum of active download duration and active upload duration.
2	$D_{active} = t_{stop} - t_{start}$	(2)	Active transfer duration for the Google Colab method, calculated from the transfer stop timestamp and start timestamp.
3	$T = S_{file}/D_{active}$	(3)	Average payload throughput, calculated by dividing the transferred file size by active transfer duration.
4	$L_{client} = R_{client} + S_{client}$	(4)	Local client traffic consumption, calculated from total received traffic and total sent traffic through the client network interface.
5	$LQE = (1 - L_{client}/S_{file}) \times 100\%$	(5)	Local quota efficiency, used to measure how much local client traffic is avoided relative to the original file size.
6	$CV = (\sigma/\mu) \times 100\%$	(6)	Coefficient of variation, used to measure the relative consistency of repeated trial results.

3. RESULTS AND DISCUSSION

This section presents and discusses the results of five repeated trials for each transfer method. The analysis focuses on file integrity, active transfer duration, payload throughput, local traffic consumption, local quota efficiency, statistical significance, and the architectural reason for the performance difference.

3.1. File Integrity and Per-Run Performance

Before performance comparison, all transferred files were validated using file size and SHA-256 checksum. The test file was 100Mb.dat with a size of 104,857,600 bytes.

All successful trials from both methods produced the same SHA-256 checksum: 3E432ABB314FE350334D244D48070BB8556F774AA7876C6C524296C2AC4608BC. This confirms that the faster transfer result did not originate from an incomplete or corrupted file, as the SHA-256 algorithm remains a robust industry standard for verifying data integrity in cloud environments [11], [16].

Table 3 shows the per-run comparison between browser local transit and Google Colab runtime transfer. The browser method required more than 211 seconds in all trials, while the Colab method completed the transfer in less than six seconds in all trials. The local traffic difference was also consistent: browser local transit consumed more than 210 MB of local traffic, whereas Colab execution consumed less than 1 MB of local traffic. This suggests that the Colab method successfully bypassed the local I/O bottlenecks inherent in traditional multi-stage transfers [3].

Table 3. Per-Run Performance Comparison

Run	Browser Duration (s)	Colab Duration (s)	Browser Traffic (MB)	Colab Traffic (MB)	Browser Throughput (MB/s)	Colab Throughput (MB/s)
1	233.615	4.603	210.398	0.553	0.428	21.727
2	211.466	5.430	211.487	0.580	0.473	18.417
3	245.335	5.187	211.051	0.730	0.408	19.280
4	242.274	4.497	213.295	0.571	0.413	22.235
5	235.775	4.435	210.593	0.617	0.424	22.548

3.2. Descriptive QoS Summary

The descriptive statistics show a substantial QoS difference between the two architectures. Browser local transit produced an average active transfer duration of 233.693 seconds, average payload throughput of 0.429 MB/s, and average local traffic consumption of 211.365 MB. In contrast, the Colab method produced an average duration of 4.830 seconds, average payload throughput of 20.841 MB/s, and average local traffic consumption of only 0.610 MB. These measurements utilize throughput as a precise metric to evaluate the effective transfer capacity of cloud-based workflows [21].

Table 4. QoS Summary and Statistical Validation

Metric	Browser Local Transit	Colab Runtime Transfer	Improvement	Statistical Result
Active duration	233.693 ± 13.300 s	4.830 ± 0.449 s	97.933% reduction	t-test p = 0.0000029
Local traffic	211.365 ± 1.159 MB	0.610 ± 0.071 MB	99.711% reduction	t-test p = 0.000000000232
Payload throughput	0.429 ± 0.026 MB/s	20.841 ± 1.868 MB/s	48.57× increase	t-test p = 0.0000171
Local quota efficiency	-111.365%	99.390%	Colab avoided almost all local payload traffic	Descriptive QoS result
Checksum validation	Identical SHA-256	Identical SHA-256	No file corruption detected	Integrity confirmed

The paired t-test results indicate statistically significant differences between the two methods in active transfer duration, local traffic consumption, and payload throughput. Although the number of trials was limited ($N = 5$), the use of the paired t-test remains statistically valid and capable of identifying large effect sizes in small-sample paired observations. The Wilcoxon signed-rank test produced a p-value of 0.0625 for the same metrics. This non-parametric approach was selected to maintain statistical power without requiring normal distribution assumptions for paired datasets [22].

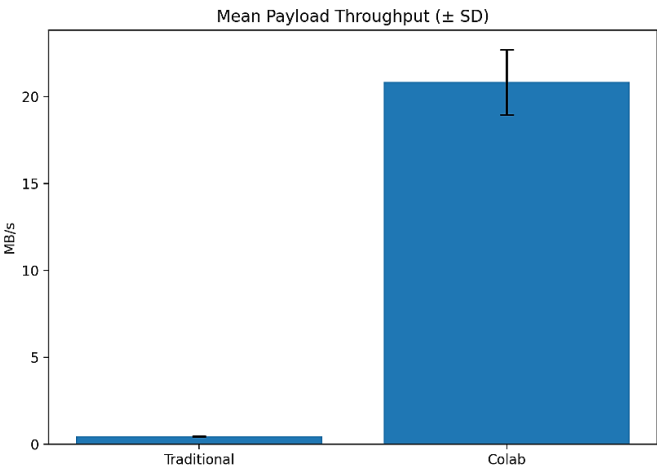


Figure 3. Mean payload throughput comparison between local browser and Colab runtime transfer

3.3. Architectural Discussion

The performance gap was caused mainly by transfer architecture, not merely by the use of Python. In browser local transit, the file payload passed through the client device twice (download and upload phases), making the local computer part of the data plane. The Google Colab method shifted the data plane to the cloud runtime, where the actual file acquisition occurred. This decoupling of the control plane and data plane is a foundational principle for ensuring performance isolation and QoS provisioning in modern software-defined storage systems [23], [8].

By separating these planes, the local client mainly transmits metadata and commands, while the high-bandwidth cloud environment handles the payload [7], [24]. This explains why the Colab method generated only 0.610 MB of local traffic. The baseline network check showed 0% packet loss and 51–59 ms latency, indicating a stable environment. However, since packet-level loss during transfer was not directly captured, it is treated as a study limitation in accordance with standard network performance evaluation practices [25].

3.4. Practical Implications

The practical implication of this study is that cloud-runtime transfer can be useful for users who need to move URL-based files to Google Drive while using limited local internet resources. In the browser local transit method, users must allocate local bandwidth and quota for both downloading and uploading the same file. This makes the method simple but inefficient for quota-based cellular connections. In contrast, the Colab runtime method reduces the local client's role to command execution and monitoring, while the file payload is handled by the cloud runtime.

This approach can benefit students, researchers, educators, and small organizations that frequently collect datasets, software packages, digital archives, or multimedia files from direct URL sources and store them in Google Drive. The method is especially relevant when the local device has limited bandwidth, limited storage, or an unstable connection. However, the Colab method still requires basic technical preparation, such as mounting Google Drive and executing a Python script, so it is more suitable for users who can operate notebook-based workflows.

3.5. Limitations

The results should be interpreted within the scope of the experiment, involving one 100 MB file and one network condition. Future research should evaluate larger datasets, multiple cloud providers, and operational costs to determine if similar efficiency patterns persist across different platforms [26], [10].

3.6. Threats to Validity

Several threats to validity should be considered. First, the experiment used a single 100 MB test file; therefore, the results may differ for smaller files, larger files, or mixed file types. Second, the experiment used one source URL and one Google Drive destination, so the observed performance may be affected by the specific source server, cloud runtime condition, and Google Drive session at the time of testing. Third, the local environment used one client computer, one wireless adapter, and one cellular hotspot provider. Different devices, adapters, internet providers, or network congestion levels may produce different transfer behavior. Fourth, packet-level loss during the actual transfer was not directly captured. For this reason, the packet loss value in this study is interpreted only as a baseline network condition before transfer testing. Despite these limitations, the repeated trials, checksum validation, local traffic monitoring, and consistent directional results support the reliability of the comparison within the tested environment.

4. CONCLUSION

This study compared two URL-to-Google Drive transfer architectures: browser local transit and Google Colab runtime transfer. The browser local transit method was defined as a two-stage workflow in which a 100 MB file was downloaded from a URL to local storage and then uploaded manually to Google Drive. The Google Colab method executed the transfer process inside a cloud runtime, while the local computer functioned mainly as the control interface.

The results show that the Google Colab runtime method outperformed browser local transit in all five repeated trials. Browser local transit recorded an average active transfer duration of 233.693 seconds, average throughput of 0.429 MB/s, and average local traffic consumption of 211.365 MB. In contrast, the Colab method completed the same transfer in an average of 4.830 seconds, achieved 20.841 MB/s, and consumed only 0.610 MB of local traffic. This means that the Colab method reduced transfer time by 97.933%, reduced local traffic consumption by 99.711%, and increased effective throughput by approximately 48.57 times.

The main finding of this study is that transfer architecture strongly affects URL-to-Google Drive performance. By separating the control plane and data plane, the Colab runtime method can move the file payload through the cloud runtime while keeping local client traffic very small. This approach is useful for users who rely on limited bandwidth, cellular hotspots, or quota-based internet access.

This study is limited to one 100 MB file, one URL source, one Google Drive destination, one Colab runtime environment, and one cellular hotspot condition. Future research should evaluate larger file sizes, different cloud storage providers, multiple network environments, multi-user scenarios, operational costs, energy consumption, and packet-level transfer behavior.

ACKNOWLEDGMENTS AND DECLARATIONS

Funding: This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Data Availability: The experimental data used in this study include PowerShell monitoring logs, Google Colab transfer logs, timestamp records, CSV files, JSON summary files, throughput charts, and SHA-256 checksum validation results. The data are available from the corresponding author upon reasonable request.

Generative AI Disclosure: Generative AI tools were used to assist the authors in improving manuscript structure, language clarity, table formatting, figure preparation, and revision mapping based on reviewer comments. The experimental design, data collection, raw measurement logs, statistical calculations,

interpretation of results, and final academic responsibility remain fully under the control and verification of the authors.

Conflict of Interest: The authors declare no conflict of interest.

REFERENCES

- [1] A. Q. Khan, M. Matskin, R. Prodan, C. Bussler, D. Roman, and A. Soyly, "Cloud storage cost: a taxonomy and survey," *World Wide Web*, vol. 27, no. 4, p. 36, Jul. 2024, doi: 10.1007/s11280-024-01273-4.
- [2] X. Gueron, S. Abrahao, E. Insfran, M. Fernandez-Diego, and F. Gonzalez-Ladron-De-Guevara, "A Taxonomy of Quality Metrics for Cloud Services," *IEEE Access*, vol. 8, pp. 131461–131498, 2020, doi: 10.1109/ACCESS.2020.3009079.
- [3] M. Lim, "C2CFTP: Direct and Indirect File Transfer Protocols Between Clients in Client-Server Architecture," *IEEE Access*, vol. 8, pp. 102833–102845, 2020, doi: 10.1109/ACCESS.2020.2998725.
- [4] S. Banerjee and P. (Dr) B. K. -, "A Theoretical Study On The Performance Characteristics Of Different Cloud Storage Access Protocols," *Int. J. Multidiscip. Res.*, vol. 7, no. 4, Aug. 2025, doi: 10.36948/ijfmr.2025.v07i04.52726.
- [5] Z. Liu, R. Kettimuthu, J. Chung, R. Ananthakrishnan, M. Link, and I. Foster, "Design and Evaluation of a Simple Data Interface for Efficient Data Transfer across Diverse Storage," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 6, no. 1, pp. 1–25, Mar. 2021, doi: 10.1145/3452007.
- [6] R. I. Zuhairnawan and H. Panduwiyasa, "Leveraging Free Cloud Services with rclone for Bandwidth-Optimized Cross-Cloud Transfers," in *2025 International Conference on Data Science and Its Applications (ICoDSA)*, IEEE, Jul. 2025, pp. 710–715. doi: 10.1109/ICoDSA67155.2025.11157583.
- [7] J. He, H. Wu, X. Xiao, R. Bassoli, and F. H. P. Fitzek, "Functional Split of In-Network Deep Learning for 6G: A Feasibility Study," *IEEE Wirel. Commun.*, vol. 29, no. 5, pp. 36–42, Oct. 2022, doi: 10.1109/MWC.003.2200060.
- [8] R. Macedo, J. Paulo, J. Pereira, and A. Bessani, "A Survey and Classification of Software-Defined Storage Systems," *ACM Comput. Surv.*, vol. 53, no. 3, pp. 1–38, May 2021, doi: 10.1145/3385896.
- [9] D. Han, S. Moon, K. Kim, S.-S. Park, and B. Nam, "Improving Remote File Access in Distributed Object Stores by Decoupling Metadata and Data Paths Using NVMe-oF," *IEEE Micro*, vol. 45, no. 6, pp. 16–23, Nov. 2025, doi: 10.1109/MM.2025.3564477.
- [10] S. Marru *et al.*, "Blaze: A High-Performance, Scalable, and Efficient Data Transfer Framework with Configurable and Extensible Features : Principles, Implementation, and Evaluation of a Transatlantic Inter-Cloud Data Transfer Case Study," in *2023 IEEE 16th International Conference on Cloud Computing (CLOUD)*, IEEE, Jul. 2023, pp. 58–68. doi: 10.1109/CLOUD60044.2023.00016.
- [11] S. S. Nitin Uniyal, Pradeep Semwal, "A Two-Phase Data Integrity Verification using Cyclic Redundancy Check and SHA-256 Algorithm for Security of Cloud Data," *Power Syst. Technol.*, vol. 48, no. 1, pp. 389–402, Apr. 2024, doi: 10.52783/pst.287.
- [12] Y. Sakurai and Y. Ito, "Assessment of QoS/QoE of Adaptive Control of Multi-Pathization Method by SDN for NFS services," in *2023 IEEE 12th Global Conference on Consumer Electronics (GCCE)*, IEEE, Oct. 2023, pp. 1129–1130. doi: 10.1109/GCCE59613.2023.10315396.
- [13] C. Caiazza, S. Giordano, V. Luconi, and A. Vecchio, "Edge computing vs centralized cloud: Impact of communication latency on the energy consumption of LTE terminal nodes," *Comput. Commun.*, vol. 194, pp. 213–225, Oct. 2022, doi: 10.1016/j.comcom.2022.07.026.
- [14] D. Regvart, M. Mikuc, and V. Dakić, "Study of Data Transfer Nodes Infrastructure in Enabling Big Data Movement between Research and Cloud Storage Networks," in *2023 46th MIPRO ICT and Electronics Convention (MIPRO)*, IEEE, May 2023, pp. 477–481. doi: 10.23919/MIPRO57284.2023.10159810.
- [15] Z. Gál, G. Kocsis, T. Tajti, and R. Tornai, "Performance evaluation of massively parallel and high speed connectionless vs. connection oriented communication sessions," *Adv. Eng. Softw.*, vol. 157–158, p. 103010, Jul. 2021, doi: 10.1016/j.advengsoft.2021.103010.
- [16] A. Alhussen and E. Arslan, "RIVACHain: Blockchain-based Integrity Verification for File Transfers," in *2020 IEEE International Conference on Big Data (Big Data)*, IEEE, Dec. 2020, pp. 3255–3261. doi: 10.1109/BigData50022.2020.9378235.
- [17] P. Boyanov, "Practical Applications of Hash Functions Md5, Sha-1, and Sha-256 Using Various Software Tools to Verify The Integrity of Files," *J. Sci. Appl. Res.*, vol. 27, no. 1, pp. 120–137, Nov. 2024, doi: 10.46687/jsar.v27i1.413.
- [18] K. R. Nanthagobal, P. Sharma, S. Das, Padmapriya, A. Wunnava, and D. N. Kirupanithi, "An Analytical Analysis of Hashing Functions in Blockchain Networks for Data Integrity Verification," in *2025 International Conference on Networks and Cryptology (NETCRYPT)*, IEEE, May 2025, pp. 1417–1421. doi: 10.1109/NETCRYPT65877.2025.11102394.
- [19] W. Uriawan, R. Ramadita, R. D. Putra, R. I. Siregar, and R. Addiva, "Authenticate and Verification

- Source Files using SHA256 and HMAC Algorithms,” Jul. 01, 2024. doi: 10.20944/preprints202407.0075.v1.
- [20] J. Carrasco, S. García, M. M. Rueda, S. Das, and F. Herrera, “Recent trends in the use of statistical tests for comparing swarm and evolutionary computing algorithms: Practical guidelines and a critical review,” *Swarm Evol. Comput.*, vol. 54, p. 100665, May 2020, doi: 10.1016/j.swevo.2020.100665.
 - [21] K. Fizza *et al.*, “QoE in IoT: a vision, survey and future directions,” *Discov. Internet Things*, vol. 1, no. 1, p. 4, Dec. 2021, doi: 10.1007/s43926-021-00006-7.
 - [22] D. Dell’Anna, F. B. Aydemir, and F. Dalpiaz, “Evaluating classifiers in SE research: the ECSEr pipeline and two replication studies,” *Empir. Softw. Eng.*, vol. 28, no. 1, p. 3, Jan. 2023, doi: 10.1007/s10664-022-10243-1.
 - [23] M. Miranda, “Distributed and Dependable Software-Defined Storage Control Plane for HPC,” in *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing Workshops (CCGridW)*, IEEE, May 2023, pp. 319–321. doi: 10.1109/CCGridW59191.2023.00071.
 - [24] O. Michel, R. Bifulco, G. Rétvári, and S. Schmid, “The Programmable Data Plane,” *ACM Comput. Surv.*, vol. 54, no. 4, pp. 1–36, May 2022, doi: 10.1145/3447868.
 - [25] E. Saeedizade, B. Zhang, and E. Arslan, “Demystifying the Performance of Data Transfers in High-Performance Research Networks,” in *2023 IEEE 19th International Conference on e-Science (e-Science)*, IEEE, Oct. 2023, pp. 1–11. doi: 10.1109/e-Science58273.2023.10254940.
 - [26] L. Luo, Y. Zhou, L. Yu, J. Shen, H. Yu, and S. Dustdar, “Optimizing Timely Bulk Data Transfers With Hybrid Elastic Cloud Resources,” *IEEE Trans. Cloud Comput.*, vol. 14, no. 1, pp. 337–351, Jan. 2026, doi: 10.1109/TCC.2025.3630149.