ADVANCES IN CYBER-PHYSICAL SYSTEMS

Vol. 9, No. 2, 2024

CLASSIFYING SERIALIZATION FORMATS FOR INTER-SERVICE COMMUNICATION IN DISTRIBUTED SYSTEMS

Eduard Maltsev[1], Oleksandr Muliarevych[1], Asmad Razzaque[2]

[1]Lviv Polytechnic National University, 12, Bandera Str., Lviv, 79013, Ukraine,
[2]Sapienza University of Rome, Piazzale Aldo Moro, 5, 00185, Roma, Italy.
Authors' e-mails: *eduard.y.maltsev@lpnu.ua, oleksandr.v.muliarevych@lpnu.ua, asmadbin.razzaque@uniroma1.it*

https://doi.org/10.23939/acps2024.02.175

Submitted on 01.10.2024

***Abstract*: This study focuses on classifying serialization formats used in inter-service communication (ISC) within distributed systems and exploring their historical development. We have examined key features of human-readable formats such as XML, JSON, and YAML, binary formats like Protocol Buffers and Apache Avro, and columnar formats such as Apache Parquet and ORC, among others. Our results have indicated a significant shift toward binary formats optimized for speed and compactness in recent years. The industry demand score for Apache Avro and Google Protocol Buffers has been shown to be much higher than for Thrift. JSON remains on top, showing the best score for general technology adoption and industry demand score; Zero-copy formats like Can'n proto and Flatbuffers show lower industry demand scores in comparison to AVRO and Protocol Buffers but are useful in specific scenarios.[1]**

***Index Terms*: Big Data applications, Data communication, Distributed processing, Encoding, Information exchange, Protocols, Software Architecture.**

I. INTRODUCTION

The ever-growing complexity of software systems has necessitated a paradigm shift towards microservices architectures. These architectures decompose functionalities into independent, loosely coupled services that communicate to achieve a unified goal. Inter-service communication (ISC) acts as the lifeblood of distributed systems. However, the efficiency of this exchange directly impacts the overall performance, scalability, and resource utilization of the entire system. One critical element influencing ISC efficiency is the choice of serialization format. Serialization refers to the process of transforming data structures into a transmittable format, allowing data to traverse network boundaries. The receiving service then deserializes the data back into its original form. Compact serialization formats minimize the size of the transmitted data, leading to reduced bandwidth usage and improved network performance.

[1]This article uses the materials and results obtained by the authors during the research work "Intelligent design methods and tools for the modular autonomous cyber-physical systems," state registration number 0124U002340 dated 09.03.2024, which is carried out at the Department of Electronic Computing Machines of the Institute of Computer Technologies, Automation and Metrology of Lviv Polytechnic National University in 2024-2028.

The proliferation of microservices architecture, cloud-native applications, and service-oriented paradigms has fueled an increasing need for robust and efficient inter-service communication [1]. As services interact and exchange data, the choice of serialization format becomes a critical factor influencing overall system performance, interoperability, and security [2]. Serialization formats dictate how in-memory data structures are transformed into a transmittable format, enabling seamless communication across diverse platforms and technologies. This process is essential for modern distributed systems, impacting various aspects from data persistence to efficient communication in resource-constrained environments [3,7]. This paper presents a comprehensive classification of serialization formats commonly employed for inter-service communication. We analyze these formats through a multi-faceted lens, considering various dimensions crucial for effective data exchange in distributed environments [4]. We investigate how different formats encode information, distinguishing between text-based approaches like JSON and XML and binary formats such as Protocol Buffers and Apache Avro [4,8]. This analysis highlights the impact of encoding choices on message size, readability, and processing overhead, echoing concerns raised in previous work on serialization efficiency [5]. We also delve into the performance implications of each format, evaluating factors such as encoding/decoding speed, message size, CPU utilization, and memory footprint [7], helping identify formats best suited for high-performance scenarios with stringent latency requirements.

II. LITERATURE REVIEW AND PROBLEM STATEMENT

The literature on serialization formats reveals a rich array of approaches to inter-service communication, each with distinct trade-offs. Source [1] reviews common data serialization methods and challenges in achieving interoperability. Research [2] suggests that Protocol Buffers and XDR are the most efficient binary serialization formats for IoT devices. Study [3] presents

PSON as a new serialization format for IoT sensor networks that simplify serialization/deserialization and minimizes message size. In our paper [4] we explored that alternative formats like binary Avro and MessagePack can reduce data size by over 30% compared to JSON for efficient inter-service communication in distributed systems. Research [5] evaluates serialization protocols to replace Java Object Serialization for efficient inter-service communication in the dCache distributed storage system. Study [6] compares performance characteristics of REST, gRPC, and Thrift communication protocols for microservice applications, finding that Thrift and gRPC are faster than REST. Source [7] suggests that binary data serialization approaches like Protocol Buffers and Apache Avro are more efficient than text-based formats like JSON/GeoJSON for storing and sharing geospatial data. Source [8] proposes a novel technique for compressing XML documents to improve communication efficiency in web-based systems. Study [9] evaluates different inter-service communication mechanisms for microservice architecture, finding that gRPC performs better than HTTP and WebSocket in terms of response time and throughput. Analysis [10] compares serialization formats like FlatBuffers and Protocol Buffers for deep neural networks, finding FlatBuffers provides better performance in model loading time and memory usage. Existing literature needs a comprehensive classification and overview that addresses the performance needs of modern real-time systems, big data processing, and scalability requirements. Our study bridges this gap by offering an updated classification of serialization formats, including columnar and binary types, with a focus on performance metrics and industry trends.

III. SCOPE OF WORK AND OBJECTIVES

The scope of this study is to classify serialization formats used in inter-service communication, including human-readable formats (XML, JSON) and binary formats (Protocol Buffers, Avro), among others. We also aim to highlight the key features and historical development of these formats.

IV. METHODOLOGY

We collected the number of Google search results by issuing the queries specified in Table 1. The template variables X1, X2, and X3 represent different spelling variations of the target serialization format name. For example ("Google Protocol Buffers" OR "Protocol Buffers" OR "protobuf"). To assess industry demand, we performed a similar search but included specific popular job posting websites. We collected the number of results from each job posting website and then calculated the total for each serialization format. Job posting websites used in this study are: indeed.com, linkedin.com/jobs. The adoption and industry demand scores A are calculated based on the corresponding number of Google results as $A = \log_{10} G$, where G is the number of Google results.

Table 1

Queries to get adoption and demand results

General adoption	("serialization format" OR "serialization formats") AND ("X1" OR "X2" OR "X3")
Industry Demand	("serialization format" OR "serialization formats")AND site:jobwebsite.com/jobs AND developer AND software AND ("X1" OR "X2" OR "X3")

V. CURRENT TRENDS IN SERIALIZATION TECHNOLOGY ADOPTION

Let us explore the results in Fig. 1 and Table 2. We can clearly see several patterns, as follows: XML and JSON have the highest number of Google search results and the highest popularity scores. XML leads with 211,000 results, followed by JSON with 70,200 results. (2) YAML ranks third with 22,800 results. It is probably due to the fact that YAML is appreciated for its human-readable format and is often used in configuration files. Its popularity is bolstered by its usage in modern development tools and platforms like Docker Compose and Kubernetes. (3) Apache Avro, Google Protocol Buffers, MessagePack, and Apache Thrift have moderate adoption scores ranging from 4.19 to 4.32. These formats are optimized for performance and efficient serialization, making them suitable for high-performance inter-service communication. The lower search volumes compared to XML and JSON may be due to their specialized use cases and steeper learning curves. (4) Formats like CBOR, BSON, Smile, FlatBuffers, and Cap'n Proto have lower popularity scores, ranging from 3.58 to 4.01. These formats serve specific needs, such as compact binary representation (CBOR), efficient storage of JSON-like documents (BSON), or zero-copy deserialization (FlatBuffers, Cap'n Proto). Their adoption is more niche, often within specific communities or projects that require their unique features.

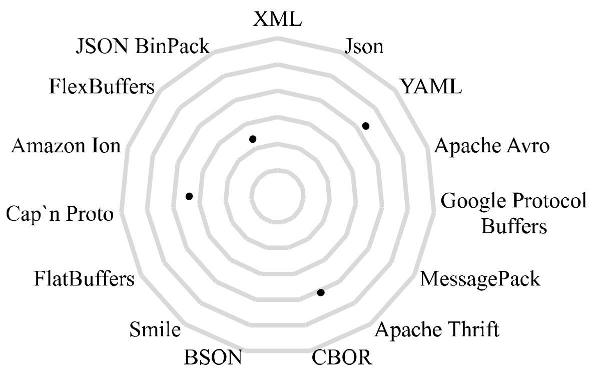


Fig. 1. General adoption score based on search engine data—the closer to the outer circle, the better

Table 2

Technology adoption and industry demand score

Format	Google Results	Technology adoption	Industry Demand
XML	211000	5.32	2.35
Json	70200	4.84	2.74
YAML	22800	4.35	1.28
Apache Avro	21200	4.32	2.51
Google Protocol Buffers	18400	4.26	2.36
MessagePack	17100	4.23	0.30
Apache Thrift	15600	4.19	0.60
CBOR	10300	4.01	<0.1
BSON	9410	3.97	0.30
Smile	7830	3.89	0.30
FlatBuffers	5980	3.77	0.48
Cap`n Proto	3770	3.57	0.3
Amazon Ion	1060	3.02	<0.1
FlexBuffers	733	2.86	<0.1
JSON BinPack	222	2.34	<0.1

VI. INDUSTRY DEMAND

Analyzing the industry demand scores for various serialization formats provides insight into current market preferences and trends. As it is shown in Table 2. and Fig. 2, JSON tops the list with the highest industry demand score of 2.74, highlighting its dominance and widespread adoption in web development, APIs, and data interchange. Its simplicity and compatibility with JavaScript make it a go-to choice for many developers.

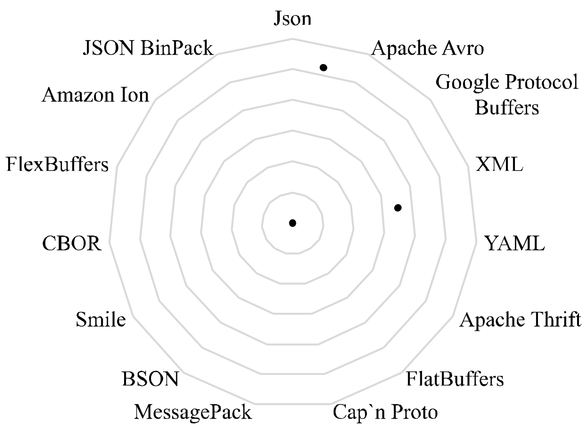


Fig. 2. Industry demand score based on search engine data— the closer to the outer circle, the better

Apache Avro follows closely with a score of 2.51, indicating significant industry interest, particularly in big data applications and frameworks like Apache Hadoop. Its compact binary format and schema evolution support make it attractive for data serialization in distributed systems.

Google Protocol Buffers and XML have industry demand scores of 2.36 and 2.35, respectively. Protocol Buffers are valued for their efficiency and performance in network communications and data storage, especially in systems where bandwidth and speed are critical. XML, despite being older and more verbose, maintains a strong presence due to its extensive use in enterprise systems and legacy applications.

YAML has a moderate industry demand score of 1.28, reflecting its niche usage in configuration files and scenarios where human readability is prioritized. Its whitespace sensitivity and complexity in parsing compared to JSON may limit its broader adoption.

Serialization formats like Apache Thrift (0.60), FlatBuffers (0.48), MessagePack, BSON, Smile, and Cap'n Proto (each around 0.30) show lower industry demand.

CBOR, Amazon Ion, FlexBuffers, and JSON BinPack, indicate minimal to no current demand in the industry. This could be due to their recent introduction, limited marketing, or specialized functionality that hasn't yet resonated with a broader audience.

VII. HISTORICAL DEVELOPMENT OF SERIALIZATION FORMATS

Serialization formats have evolved to support efficient inter-service communication. In the 1950s, Lisp's S-expressions enabled hierarchical data. CSV appeared in the 1970s for simple data exchange but lacked complex data support. The 1980s introduced ASN.1 and encoding rules like BER, standardizing data for telecom and security protocols like SSL/TLS. Sun Microsystems' XDR enabled cross-architecture serialization for RPC systems. The 1990s saw language-specific formats (e.g., Python's Pickle, Java's Serializable) but limited interoperability. XML, despite its verbosity, structured SOAP web services. JSON rose in the 2000s for web apps and REST APIs, while YAML supported configuration files. Binary formats like Protocol Buffers, Apache Thrift, and Apache Avro enabled efficient cross-language data interchange for high-performance services. MessagePack offered binary efficiency with JSON simplicity. In the 2010s, Apache Parquet and ORC optimized big data, while Apache Arrow improved data processing interoperability. Cap'n Proto and FlatBuffers minimize serialization overhead for real-time communication. Other formats like CBOR and Amazon Ion extended JSON's flexibility with compact encoding and richer data types.

VIII. TEXTUAL FORMATS

Textual row-based formats organize data as sequences of records, where each record consists of fields that contain the values for a single entity. These formats are human-readable, which makes them easy to debug, inspect, and share across platforms. The most commonly used textual row-based formats in inter-service communication include:

JSON (JavaScript Object Notation): A lightweight, text-based format that organizes data as key-value pairs. It is widely used for RESTful APIs and is natively supported by many web technologies.

XML (eXtensible Markup Language): A more verbose format with a hierarchical structure, often used in SOAP-based web services. XML is highly flexible and supports schema validation.

The result of the serialization of the reference data structure using JSON format is presented in Fig. 3.

Serialized Form - JSON (34 bytes)																			
Byte #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Hex	7B	22	6D	6F	64	65	6C	22	3A	22	43	41	54	20	44	31	31	22	2C
Dec	123	34	109	111	100	101	108	34	58	34	67	65	84	32	68	49	49	34	44
Text	{	"	m	o	d	e	l	"	:	"	C	A	T		D	1	1	"	,
Byte #	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34				
Hex	22	77	65	69	67	68	74	22	3A	38	35	30	30	30	7D				
Dec	34	119	101	105	103	104	116	34	58	56	53	48	48	48	125				
Text	"	w	e	i	g	h	t	"	:	8	5	0	0	0	}				

Fig. 3. Result of serialization using JSON data format, size is 34 bytes in total

A. *MESSAGE SIZE AND VERBOSITY*

Textual formats like JSON and XML are verbose, often resulting in large message sizes due to the use of human-readable text. Key names are included in every record, adding redundant data to each transmitted message.

For example, a JSON message with multiple records will repeat the field names (e.g., "price": 100, "price": 120) for each record. Larger message sizes require more bandwidth for transmission, increasing the time taken to send and receive messages. For example, a JSON-based API sending user profiles with fields such as name, age, and address must transmit these field names in every record, increasing the payload size.

This can be a significant issue in high-frequency communication between microservices, where bandwidth consumption directly contributes to network latency.

B. *PARSING AND SERIALIZATION OVERHEAD*

Textual formats must be serialized and deserialized as strings, which is computationally expensive compared to binary formats. JSON and XML parsing libraries convert strings into objects or data structures in memory, a process that is non-trivial for large payloads or high-frequency requests.
The process of converting JSON or XML into in-memory data structures (and vice versa) consumes CPU cycles, which increases latency.

One of the main advantages of textual formats is that they are human-readable, making debugging and monitoring easier. However, this readability comes at the cost of increased message size and serialization complexity.

While human-readable formats are beneficial for debugging, they introduce performance bottlenecks in latency-sensitive applications. In scenarios like telemetry data exchange in IoT systems, where devices communicate frequently, the added overhead of textual formats can lead to unacceptable delays.

C. *PERFORMANCE OPTIMIZATION STRATEGIES*

To mitigate the intrinsic issues related to verbosity and computation costs of textual formats like JSON and XML, several performance optimization strategies can be implemented. These not only aim to reduce latency but also help in managing bandwidth and computational overhead more effectively.

Data compression is a crucial technique to reduce message size. Compression algorithms like GZIP or Brotli can be applied to JSON and XML messages before transmission, significantly cutting down the data volume.

IX. BINARY FORMATS

Binary serialization involves converting complex data structures into a compact binary format that can be easily transmitted over a network and reconstructed later.

This contrasts with text-based formats like JSON or XML, which, while human-readable, are bulkier and require more processing time for parsing and serialization. Binary formats eliminate unnecessary characters such as whitespace and tags, resulting in smaller message sizes.

In Fig. 4 (AVRO) we can clearly see the huge differences in the resulting serialized representation of the same messages when compared with Fig. 3. (JSON), 11 vs 34 bytes while serializing the same data structure.

Serialized Form - AVRO (11 bytes)											
Byte #	1	2	3	4	5	6	7	8	9	10	11
Hex	0E	43	41	54	20	44	31	31	90	B0	0A
Dec	14	67	65	84	32	68	49	49	144	176	10

Fig. 4. Result of serialization using AVRO data format, size is 11 bytes in total

This reduction in data size directly impacts network transmission times, as smaller payloads take less time to send and receive, thereby reducing latency.

Furthermore, binary serialization formats often come with schema definitions that both the sender and receiver agree upon. Examples include Protocol Buffers by Google, Apache Thrift, and Apache Avro.

These schemas define the structure and data types expected, allowing for efficient serialization and deserialization processes. Knowing the exact data types in advance enables services to allocate appropriate resources and parse incoming data quickly, minimizing CPU overhead. This is especially important in high-throughput systems where services handle a large number of requests per second.

Another advantage is that binary formats can represent primitive data types in their native binary form. For instance, integers, floats, and booleans can be serialized without converting them to text, as it would be necessary in JSON or XML.

In addition, binary formats can efficiently handle complex nested structures and repeated fields without significant overhead.

In latency-sensitive applications such as real-time analytics, financial trading platforms, and large-scale web services, every millisecond counts, in such architectures, where a single user request may involve multiple inter-service communications, the cumulative latency savings can be substantial.

X. ENVIRONMENT SPECIFIC FORMATS

Environment-specific serialization formats are tailored to the features and idioms of a particular programming environment. As it is shown in Fig. 5, we separated these kinds of formats into a separate category. They often provide seamless integration and can leverage language-specific optimizations. Examples include:

Java Serialization: Uses the Serializable interface to convert Java objects into a byte stream.

Python Pickle: Serializes Python objects into a byte stream, preserving Python-specific data types.

Environment-specific formats are particularly useful in scenarios where components are developed and deployed within a single programming environment, leveraging environment-specific serialization allows for deep integration and optimal performance.

XI. COLUMNAR FORMATS

The adoption of columnar formats for inter-service communication is often driven by specific use cases where read-heavy operations, analytical queries, or batch processing dominate. Please take a look at the structure difference between row-based and columnar formats in Fig. 6. Traditionally, communication protocols between services favor row-based formats due to their ease of use and suitability for transactional operations.

However, as data volumes increase and the need for efficient data access grows, columnar formats present an attractive alternative, especially in scenarios.

Row-Based Format				
1	timestamp	1695472300	price	102.3
2	timestamp	1695472360	price	103
3	timestamp	1695472420	price	101.5

Columnar Format				
1	timestamp	1695472300	1695472360	1695472420
	price	102.3	103	101.5

Fig. 6. *Columnar vs. row-based formats, columnar uses less space, even visually.*

XII. CLASSIFICATION

Fig. 5 offers a classification of data serialization formats, divided into textual and binary categories, each tailored for specific applications and data handling requirements. Human-readable formats include XML, JSON, and YAML, each serving distinct purposes like web services, web applications, and configuration settings, respectively. XML supports complex structures and is extensively used, whereas JSON is preferred for its simplicity and performance, particularly in web contexts.

YAML excels in configurations due to its readability and natural representation of hierarchical data.

On the binary side, formats like Protocol Buffers, Apache Avro, and Thrift are highlighted, known for their efficiency in processing and bandwidth usage, making them suitable for high-performance computing environments. These binary formats offer advantages in serialized data size and processing speed, making them ideal for environments where performance is critical. The diagram also mentions columnar formats such as ORC, Apache Arrow, and Parquet, which show great utility in both storage and rapid retrieval scenarios.

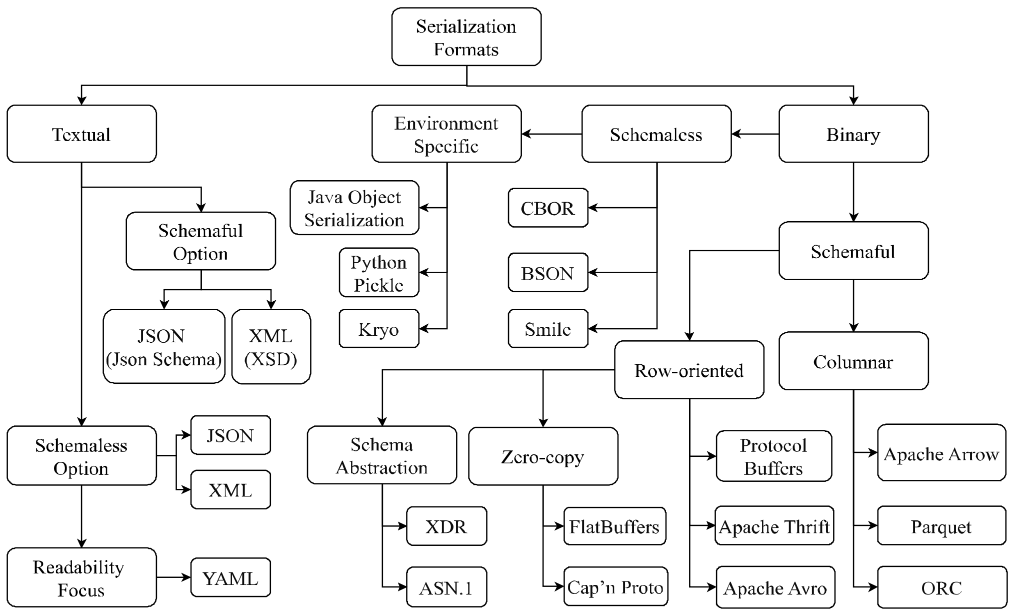


Fig. 5. *A classification of serialization formats for inter-service communication*

XIII. CONCLUSION

This study systematically classified serialization formats, offering a comprehensive understanding of their application in inter-service communication within distributed systems. Our analysis identified key differences between human-readable formats, such as XML, JSON, and YAML, and binary formats, like Protocol Buffers, Apache Avro, and Thrift. While textual formats provided ease of use and superior readability, which facilitated debugging and integration, they inherently suffered from increased data payload and computational overhead during parsing, making them less suited for high-performance or latency-sensitive applications.

Our findings indicated a significant industry adoption and demand of binary formats, particularly in domains requiring rapid and efficient data processing capabilities, such as real-time systems and microservice architectures.

The choice of serialization format should be strategically aligned with system requirements, balancing factors like performance, ease of integration, and future scalability.

References

[1] Adibfar, A., Costin, A., (2021). Review of Data Serialization Challenges and Validation Methods for Improving Interoperability in *Computing. In Civil Engineering,* 522–529. DOI: 10.1061/9780784483893.065.

[2] Friesel, D. and Spinczyk, O., (2021). Data Serialization Formats for the Internet of Things. *Electronic Communications of the EASST*, vol. 80. DOI: 10.14279/TUJ.ECEASST.80.1134.1078.

[3] Luis, Á., Casares, P., Cuadrado-Gallego, J., Patricio, M., (2021). PSON: A Serialization Format for IoT Sensor Networks. *Sensors*, vol. 21, no. 13, 4559. DOI: 10.3390/s21134559.

[4] Maltsev, E., Muliarevych, O., (2024). Beyond JSON: Evaluating Serialization Formats for Space-Efficient Communication. *Advances in Cyber-Physical Systems*, vol. 9, no. 1, 9–15. DOI: 10.23939/acps2024.01.009.

[5] Morschel, L., (2020). dCache – Efficient Message Encoding For Inter-Service Communication in dCache: Evaluation of Existing Serialization Protocols as a Replacement for Java Object Serialization. *EPJ Web Conf.*, vol. 245, 05017. DOI: 10.1051/epjconf/202024505017.

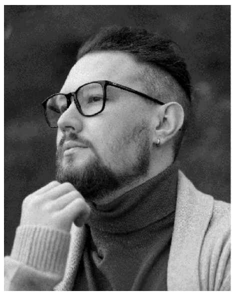
[6] Kumar, P., Agarwal, R., Shivaprasad, R., Sitaram, D., Kalambur, S., (2021). Performance Characterization of Communication Protocols in Microservice Applications. In *2021 International Conference on Smart Applications, Communications and Networking (SmartNets)*, Glasgow,1–5. DOI: 10.1109/SmartNets50376.2021.9555425.

[7] Mooney, P., Minghini, M., (2022). GEOSPATIAL DATA EXCHANGE USING BINARY DATA SERIALIZATION APPROACHES. *Int. Arch. Photogram. Remote Sens. Spatial Inf. Sci.*, vol. XLVIII-4/W1-2022, 307–313. DOI: 10.5194/isprs-archives-XLVIII-4-W1-2022-307-2022.

[8] Tiwary, G., Stroulia, E., Srivastava, A., (2021). Compression of XML and JSON API Responses. *IEEE Access*, vol. 9, 57426–57439. DOI: 10.1109/ACCESS.2021.3073041.

[9] Weerasinghe, S., Perera, I., (2022). Evaluating the Inter-Service Communication on Microservice Architecture. In *2022 7th International Conference on Information Technology Research (ICITR)*, 1–6. DOI: 10.1109/ICITR57877.2022.9992918.

[10] Parashar, A., Anand, P., Abraham, A., (2020). Performance Analysis and Optimization of Serialization Techniques for Deep Neural Networks. In *Computer Vision, Pattern Recognition, Image Processing, and Graphics*, 250–260. DOI: 10.1007/978-981-15-8697-2_23.



Maltsev Eduard obtained his Master's in Computer Engineering, specializing in Computer Systems and Networks, from Lviv Polytechnic National University in 2013. In 2021, he became a Certified Cloud Architect and is currently working towards a Ph.D. in Computer Engineering.

Oleksandr Muliarevych is an associate professor at the Computer Engineering Department at Lviv Polytechnic National University. He earned his PhD degree in Computer Systems and Components at Lviv Polytechnic National University in 2016

Asmad Razzaque obtained his master's in electrical engineering from the National University of Science and Technology (NUST) in 2022. He is pursuing a Ph.D. in Information and Communications Technology (ICT) at University of Rome Sapienza. He has published a few papers and received a Best Paper Award in an IEEE conference for his contributions.