ESP-IJACT
ESP International Journal of Advancements in Computational Technology
ISSN: 2583-8628 / Volume 2 Issue 3 August 2024 / Page No: 61-76
Paper Id: IJACT-V2I3P106 / Doi: 10.56472/25838628/IJACT-V2I3P106

Original Article

Low Latency High Throughput Data Serving Layer for Generative AI Applications using the REST-based APIs

Lakshmana Kumar Yenduri
Senior Staff Software Engineer, Tech Lead for Apple Pay Generative AI Applications | IEEE Senior Member, Visa Inc, San Francisco Bay Area, California, 94404, USA

Received Date: 29 May 2024 Revised Date: 02 July 2024 Accepted Date: 01 August 2024

Abstract: *Based on the discussion of generative AI and other successful neoteny complex AI applications such as large language models and synthesis of images and other forms of AI-generated creativity, the efficiency of an application depends much on data management systems. Most of these applications are computationally intensive. Hence, the serving layer being required to have high I/O and response rates, in essence, being real-time. The generic data-serving architectures differ and are shown to be unsalable and slow when it comes to the case of generative AI. The subsequent paper outlines a novel architecture for managing the four factors using REST-based API's for integration and interaction. The idea is to reveal the state-of-art technologies consisting of the multi-level caching approaches, distributed databases, and the optimal RESTful API architecture to construct the fully independent, reliable, and beautiful data-serving layer. With respect to the important characteristics involved in data handling, such as data access pattern optimization, query optimization, and network, the architecture offered a response time that was significantly slower than the increasing load. The use and integration of distributed databases ensure that the system has the characteristic of being horizontally scalable, meaning that the increase in the amount of data that the system has to process does not compromise efficiency. On the same note, the caching architectures are used to conserve frequency by storing data in the regions of usage. In this paper, discussions revolve around structures of design and total performance evaluation of design with various inputs credited to the strategies employed in implementing the design. Extracting an average of the performance metrics, it can be seen that the architecture meets the needs as requested and is heavily optimized for low-latency and high-throughput, let alone integrated real-time generative AI applications. From this work, we understood the concept and efforts that need to be applied to make data-serving layers in the future advancements of the field of artificial intelligence and laid down the development path for future expansion of this ever-evolving technology.*

Keywords: *Low latency, High throughput, Data serving layer, Generative AI, REST APIs, Scalability.*

I. INTRODUCTION

Regarding generative AI, the rate of AI advancements and its utilization across the fields is accelerated, and the fields that are associated with generative AI are text generation, image synthesis, and music production. Thanks to the advance of machine learning algorithms in general, as well as GANs and transformer-based architectures mainly. All these applications presume one's ability to synthesize original, intricate material from large quantities that have been trained; this, in return, necessitates efficient data accessibility and data handling abilities. As the creation of the models and sizes of data by which those models are trained increases, new requirements show up regarding data storage. [1-5] there is now a requirement to increase the speed at which such data is retrieved and used in performing these models that are comprised of such tasks as real-time generation of content and also large-scale data analysis. These models inherently need a lot of parallel computing and fast access to data whiles the traditional way to serve. Thus, there is a great need to advance the best data management practices, referring to perspectives and scopes that appear in generation modern generative AI systems. This has led to paying attention to what architectures should be, high-performance data serving systems, growing complexities and sizes of models, and artificial intelligence.

A. Advantages of Restful APIs in AI Systems

a) Simplicity and Flexibility:

Among the types of APIs, there are RESTful APIs, which are very popular due to their simplicity and flexibility; thus, they are integrated into the majority of AI systems. [6,7] Consequently, the fundamentals of the REST architecture are fairly straightforward, and it is thereby comparatively straightforward to implement the APIs of REST without needing countless specifications. Like any other regular processes, the REST APIs interface with AI systems via HTTP forms, including get, post,

put, and delete, among others. Scalability is another pro in using RESTful APIs; such APIs are scalable because the APIs are not in any way associated with the architecture, environment, or even a programming language. This feature is especially relevant in case an AI system needs to interact with other parts that are also delivered by this service, such as data storage services, the models already trained, and external resources.

*b) Scalability***:**

Subsystem interdependence is one of the crucial features of AI systems because these systems can work with large amounts of data and receive multiple requests. Firstly, it should be noted that all RESTful APIs are designed based on their inherent characteristics to conform to the scalable architecture. The clients are limited and unknown, making them stateless; therefore, every one of them is an independent request and does not have a server to store extensive information between them. This statelessness also aids RESTful APIs because it allows loads to grow in multiple servers or clusters, making it possible to do horizontal scaling. More servers can easily be incorporated into an AI system in a way that does not necessitate a big shift to the API as the need for the system increases. This capability is vital in domains where the amounts of data or computations can be huge, and the expansion of the solution is nearly infinite.

c) Interoperability:

Since AI systems employ RESTful APIs, they similarly have another benefit in interoperability. RESTful APIs run on HTTP, making them open standards that can interact with other web standards. This capability is highly essential for AI systems because such systems are mainly designed for interoperation with other services, platforms, or external data. For instance, it may mean the AI system needs to read from a particular cloud storage app to get some data, or it has to forward particular results back to a different web app, or there is an identification of expectations for the AI model to spotlight to another AI system. Working and Collaboration: These interactions include the exchange of information and integration through powerful techniques such as the use of RESTful APIs, which help in the interconnection of the systems. Among the social characteristics of RESTful APIs, the ability to develop and deploy AI systems without connecting with numerous other services is vital; therefore, these API systems are crucial to developing AI systems.

d) Security:

Another significant issue in AI systems is security, for example, in cases where the system deals with sensitive information or performs vital roles. Security is also well implemented in RESTful APIs that assist in maintaining the integrity of the AI systems against intruders. The first of the principal security measures is using HTTPS to encrypt data being transferred between the clients and the servers so that it cannot be intercepted or modified in any way. Also, RESTful APIs can support OAuth or JWT (JSON Web Tokens) or API keys, which define the restriction of access to a particular user or application only. All these layers of security can easily be applied to RESTful API to ensure a secure line of communication by the AI systems. Accounting for this, RESTful APIs' flexibility enables organizational security measures appropriate to individual AI application risks.

B. Role of REST-Based APIs in AI Systems

a) *Integration of AI Models with REST APIs*

REST specific APIs can be stated as one of the inputs used while re-implementing AI models into applications or systems. Since they contribute to informing other software components that an object under consideration contains the functionality of the AI model, they are language- and platform-independent. [8-11] REST APIs can be utilized by the developers in making their AI models further as a service which makes the predictions, data readiness or other AI aspects easily available. This approach allows AI capabilities to be implemented in web and mobile applications, corporate systems, and IoT devices so that useful components of AI can be provided to as many people as possible without worrying about the format of the AI model.

b) Scalability and Performance in AI Workflows

REST is one of the attributes of the architectural style that enables AI systems to handle an enormous number of requests and amounts of data. They provide a well-developed structure for real, fast, and efficient communication of AI service and application clients. Since RESTful services do not rely on any state, the services can be planted on different servers or cloud environments; hence, AI systems are horizontally scalable to accommodate real-time computation. Such scalability is providential for real-time AI systems comprising immediate responses like finance, trading, recommendation systems, self-driving cars, etc. Moreover, REST APIs offer support for such concepts as load balancing and caching, and they include support for asynchronous processing in addition to the combined efficiency of AI processes.

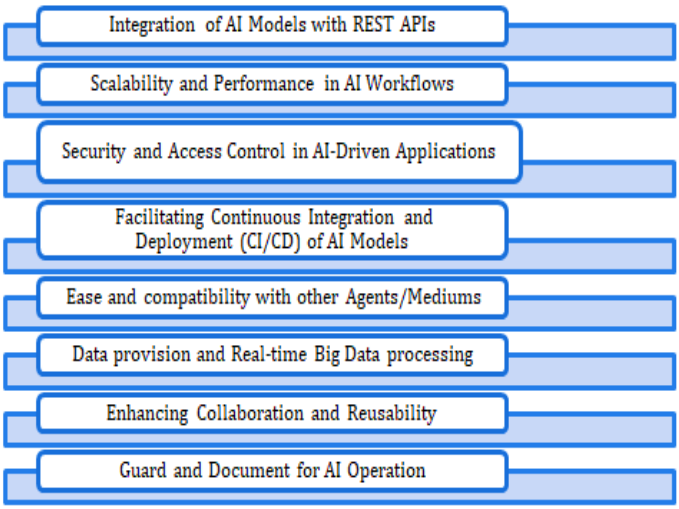

Figure 1: Role of REST-Based APIs in AI Systems

C) Security and Access Control in AI-Driven Applications

In a nutshell, in the domain of AI systems, it is crucial to focus specifically on the problem of security and access control, especially if it is a question of processing typical or limited data. REST provides a formal way of implementing security into AI service offerings via authentication and authorization techniques, including OAuth, API authentication, and token-based authentication. These above security parameters allow AI to be used only by allowed personnel or the network as a whole, thus blocking invasions and/or piracy. Also, REST APIs facilitate easing on the one hand and the configuration of such matters as rate limiting, IP whitelisting, and others, which prove to be crucial while addressing the question of data security AI solutions in sectors such as healthcare, finance and government.

d) Facilitating Continuous Integration and Deployment (CI/CD) of AI Models

Regarding CI and CD integration in AI systems, REST-based APIs are inevitable. They assist in creating one or more versions of the AI models and their updating and deployment methods. In fact, in the REST APIs, the developers can put new versions or new updates of the model in production, irrespective of other services. This is needed to refresh the models and bring them back on track in case of highly dynamic data, which is characteristic for such fields as e-commerce or social networks. The use of CI/CD in the organizational implementation and improvement of AI models should integrate REST APIs' elements to enhance the advancement and deployment of the models.

e) Ease and compatibility with other Agents/Mediums

That is why employing the architecture of REST-based APIs offers a multitude of advantages in the field of connectivity, where the AI systems can easily connect to other various platforms. Thus, depending on the target language, which can be Python, R or any other language, a REST API may form a layer of the same kind. This is especially useful in core environments where risk stems from dissimilarities in systems, databases and the services provided. REST APIs also work with other formats, such as JSON and XML, for better AI integration with enterprise systems, cloud services or even mobile and web applications. This indicates that existing and today's planned AI technologies can be incorporated or applied in numerous technological settings.

f) Data provision and Real-time Big Data processing

Most real-time data serving and analysis in AI systems can be carried out through REST-based APIs. They introduce the value of enabling AI applications to manage big data commonly associated with real-time decisions effectively. For instance, if predictive analysis or the recommendation system is employed, REST APIs could be hired to feed data to the AI algorithms and serve back the effects in real-time. In handling concurrent requests, that is, in the analysis and response required in real-time analysis, scaling is also the strength of REST APIs, which makes them critical components and part of the AI systems that require such analysis and response within the shortest time possible.

g) Enhancing Collaboration and Reusability

Adapting REST-based APIs in the framework enhances interoperability and modularity while releasing AI services as they are developed independently; hence, the concept of reusability is enhanced. The moment an AI model has been deployed via an API over HTTP rest, it becomes a tool that can be utilized in different environments by different projects and teams, and more often than not, it needs to be reinvented from the ground up. This means that data scientist and the operations employees of the developers always work in harmony. Also, the characteristic realizations of REST APIs include reusability, meaning that the AI models developed in one application can be reused in others, greatly reducing development time and resources. This also enables the developers of AI to share the functionalities with other partners/clients, and hence, there will be an expansion of the use of AI across different fields.

h) Guard and Document for AI Operation

Logging and monitoring are other key areas where REST-based APIs are used to represent the status, patterns, logs, and vitals of the AI models in production. Whenever logging is integrated with it or utilized while applying REST APIs, such organizations will be capable of tracking the employed APIs, monitoring the response rate, and logging some error or any conspicuous event. This data is really valuable when it is high time to fix the bugs, improve the algorithms' running speed and deal with the dependability of AI systems. Also, they can be hooked into existing monitoring systems and utilities with which developers can observe AI models in real-time mode and address emerging problems. This capability is required to prevent the application of Artificial Intelligence from causing instability in an application and work optimally according to the role that it plays in a specific firm or organization.

C. Challenges in Data Serving for Generative AI

a) High-Volume Data Processing:

A major issue concerning the data serving for generative AI must be addressed is data storage for training and inference. The generative AI models, especially LLM and GANs, need large volumes of input data to produce good quality output. Delivering these data in an efficient manner is, at times, not easy, more so when these data are in the unstructured form or from multiple sources. As the stone of the data increases, it is very challenging to meet the low latency access while at the same time managing the throughputs for this data, to which end the storage, retrieval and pipelines for the data must be robust and efficient enough to handle the load.

b) Latency and Real-time Constraints:

Since most uses of generative AI need to respond in real-time or nearly real-time, the system requires high levels of processing to perform the needed computations. For example, use cases such as text generation in real-time, generation of images, or evolution of videos need large data search and processing in real-time to fit the users' standards. The focus is, therefore, to reduce latency in all data-serving systems, from the data acquisition and preparation stages to the model computation and result delivery stages. This entails highly efficient architectures for data serving and caching combined with employing caching mechanisms, in-memory databases and other features to prevent delay.

c) Data Consistency and Integrity:

Another significant concern is data consistency and referential integrity because this is a severe problem starting from the time when data is used to serve for generative AI to design perspective, especially acting with a distributed system or if the data origin is multiple and may be asynchronous. The great instability of the data can reduce the performance level of a particular model or present wrong output when one is using the real-time feed data from the model. It is essential and rather challenging that the data used in the model should always be correct, timely and in harmony with the several databases/systems feeding the model. This might require considerable use of consensus algorithms or distributed DBMS.

d) Scalability of Data Infrastructure:

What is more important, for the generation of the AI models themselves, which are evolving from simple to complex entities and growing bigger and bigger, the need for the kind of data structure is much the same. This means the expansion of not only the amount of rings that need to store data or accommodate them but also the capability of the serving systems that must meet these demands. Regardingthe scalability difficulties mentioned, it is worth following up on the load problems for databasesand the issues with partitioning or distribution of the data among nodes or clusters. Moreover, the structure should be expandable to fit really high loads, especially when applied to applications that possess irregular loads, such as generative chatbots or creation instruments => usage of which can sharply increase or sharply decrease at any given time.

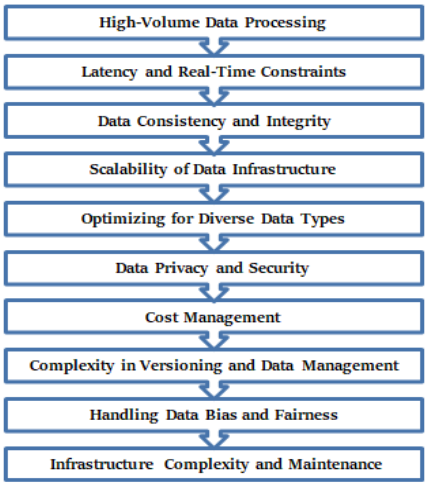

Figure 2: Challenges in Data Serving for Generative AI

e) Optimizing for Diverse Data Types:

Some generative AI models deal with different data types, such as text, image, voice, and even video. Feeding WHS with such diversified data patterns is relatively cumbersome since there are various types of data with unique treatment, storage, and retrieval methods. For instance, when it comes to handling large image or video files, it must be done with little or no delay, which is very different from handling random text. Also, when data is highly voluminous or retrieved as needed, some form of preprocessing is often necessary for images, such as resizing or text; for instance, normalization further complicates the data-serving process. This calls for a very flexible and optimized layer of data serving that must be able to parse and handle a number of data forms simultaneously.

f) Data Privacy and Security:

Another crucial issue related to data serving for generative AI is that dealing with personal information is also rather problematic. These models usually require the use of huge amounts of data, some of which may be sensitive to a certain party or private information. Security becomes an important aspect of managing data as it is fed to the generative models to avoid cases where data is exposed to the public domain, stolen, or accessed by unauthorized persons. This relates to applying encryption, access control, and data anonymization methods on the data-serving architecture. Moreover, following the requirements of specific regulations like GDPR or HIPAA makes the problem even more significant, as it implies the necessity to follow the legal demands and design the systems serving data with privacy by design principles.

e) Cost Management:

Storing and serving data to drive generative AI at the expected scale can be very expensive, especially if it demands high throughput and low latency solutions. The cost of data storage, data traffic, and data processing might prove to be too expensive, particularly when the application is gigantic or where cloud solutions are applied. The key difficulty here is to control these costs effectively while keeping up to par with the performance requirements. This must be done with clear analysis and determination of the efficiency/effectiveness of the used resource, which might involve data compression, tiered storage or pre-fetched data caching, and the use of spot instances in the cloud environments.

f) Complexity in Versioning and Data Management:

Introducing multiple versions of the data fed to the generative AI models creates another layer of work in the process of data serving. With evolving models, the models themselves can either need a different data set or a different version of the same data set – creating major issues related to data versioning. Precisely, concerning the rollout of the correct version of the data to the right model at the right time is important to the model's reliability and accuracy. This requires efficient practices in handling data, such as versioning datasets and metadata and the ability to revert back or update datasets and not affect the working of other datasets in the organization.

g) Handling Data Bias and Fairness:

One major impact on generative AI is data bias, and the fairness problem is how data is served, which worsensit. There is a risk of introducing a new bias in the data-serving process. Those outputs can be detrimental when using the outputs for sensitive applications such as content generation, recommendation systems, or other decision-making devices. Solving this challenge would involve designing the data-serving layer well and presenting balanced and centrist data to the AI models. This may need the use of approaches such as data sampling methods, bias identification tools, and fair data handling methods to enable the AI produced to be fair.

h) Infrastructure Complexity and Maintenance:

The structure of the data-serving infrastructure is a challenge that lies within the first challenge; this is so since the structure can be exceedingly large, complex, and intricate, especially as it advances to cater to generative AI. Managing such an infrastructure implies having the availability, performance, growth, upgrade and problem-solving of the infrastructure and distribution. The highavailability requirement, disaster-recovery need, and the inevitable maintenance task contribute to the operational pressure that increases the requirement for professionally skilled people and better tools to run the infrastructure. Managing the trade between designing highly effective, sophisticated generative AI and the capacity to sustain and manage it as the versatile data-serving layer is a never-ending effort at the best of circumstances.

II. LITERATURE SURVEY

A. Overview of Generative AI

Regarding generative AI, one should remember that it is a comparatively new division of AI that implies the creation of new models. They have gone ahead and transformed areas such as entertainment, health, and even finance through automation of content creation, creativity, and the provision of solutions tailored to suit individuals' needs and wants. [12-15] Thus, the essence of generative AI, for the most part, is contained in deep learning configurations that embrace Generative Adversarial Networks GANs, Vibrational Autoencoder VAEs, and comparatively more contemporary deep learning architectures such as transformers like GPT Generative Pre-trained Transformer. These models are fed with huge databases, and the learning process of these patterns and generation of new data involves many computations and some detailed algorithms. The training phase is, therefore, the process of learning the input data distribution,

While the inference phase entails generation of new instances, which is usually real-time in most applications. While these models can be aided by the latest advancements in hardware, such as GPUs and TPUs, in scaling these models, the efficiency of dealing with the data is still a problem. Other researchers, such as Goodfellow et al. (2014) and Radford et al. (2019), have demonstrated how generative models can be applied to produce outputs as realistically as possible, and that is why proper architectures of data serving systems should be put in place to support such systems.

B. Data Serving in AI Applications

Most strikingly, data serving is one of the core phases in almost all aspects of AI, primarily because it directly impacts the decision-making of the opportunities and scales of AI solutions. As a result, the traditional AI techniques employed batch processing in which one can upload a large amount of data beforehand and proceed with the processing in a non-interactive manner to support the training and inference mechanics. It is, however, fast becoming a thing of the past, particularly in the generative AI uses that demand instantaneous data access and processing. Thinking about conversational agents, real-time image generators or other real-time AI systems, one has to understand that they have to be ready to process data and produce results on the spot, switching from batch processing to serving, i. e. Dean et al. (2012) provided the idea of distributed systems and cloud solutions for enhancing data access and reliability; simultaneously, such systems have to be designed for low latency, which is specific to generative AI. This is seen by the serving pattern of data structures, starting from the relational database to NoSQL databases and in-memory data stores, which show an effort towards addressing the issues of real-time data ingestion to AI systems.

C. REST-Based APIs in Modern Applications

Representational State Transfer (REST) has now earned its place as the most preferred architectural style in creating applications rooted in the World Wide Web and other related ones. RESTful APIs are used because of their simplicity and scalability, which are stateless and key characteristics of distributed systems. However, the lack of a state creates some issues if we apply REST in the high throughput and low latency environment that generative AI will require. REST APIs primarily operate over HTTP; HTTP is a stateless protocol, which means each request that the client sends to the server needs to contain all the

information required by the server to process the request. This is good for scalability purposes, but it also raises latency levels because excessive amounts of state information have to be duplicated over and over again. The author of the REST originally, Fielding (2000) presents these trade-offs and also several approaches – those approaches are caching, load balancing and optimized usage of payload. As a continuation of the work above, including but not limited to Richardson and Ruby (2007) have attempted to introduce how one can enhance the performance of a given RESTful API through ways such as HTTP/2 that allow a set of requests and the corresponding replies to be streamed over a connection. Therefore, the major focus of AI applications created through web APIs can be described as REST principles, making applications as plain and as light as possible while simultaneously being as efficient as possible to provide low latency and high throughput.

D. Related Work on Low Latency and High Throughput Systems

Efficiently delivering data to consumers at a low latency and high rate has been a well-studied area broadly categorized under distributed computing and database systems. Based on the original development by Abadi et al. (2009), distributed databases set the tone for today's data-serving architectures capable of horizontal scaling to accommodate vast amounts of data while always delivering fast access. It is for this reason that methods like data partitioning, replication, or even the use of in-memory database techniques have been found useful in reducing latency and increasing throughput. New posts, including NewSQL databases that adhere to NoSQL scalability while providing ACID guarantees without degrading the performance of SQL, are promising the development of AI-backed programs. Some of the research done by Stonebraker and Cattell (2010) has indicated that these systems have the ability to offer the required low-latency access for real-time AI applications. Also, techniques such as in-memory stores such as Redis and Memcached have been discussed to minimize the access time because such data is kept in memory rather than disk I/O. According to the study of Han et al. (2016), the internal protocols used in the system should be fine-tuned, for instance, utilizing RDMA, to cut the costs of data transfer in distributed systems since this factor significantly affects throughput in AI applications.

E. Challenges in Serving Data for Generative AI

To formulate data for generative AI, additional procedures must be followed beyond the cases mentioned for other forms of AI. Firstly, the problem is identified with huge quantities of data to process and needs to be fed into these kinds of models. Generative AI models require access to large amounts of data and often need to process it in real-time, which can put a heavy load on the most sophisticated data-serving systems. In addition, since REST APIs are stateless, something favorable for scalability, session management, and data integrity issues reared its ugly head. Keeping data synchronized in distributed systems is perhaps one of the well-known problems in distributed computing, especially when several nodes may be processing and modifying data simultaneously, as Brewer pointed out in the CAP theorem. On top of that, generative AI solutions frequently define precise data access patterns throughout the process as a means of performance enhancement, which is challenging to apply with ease utilizing the RESTful abstractions. This need further complicates those issues because any latency in data retrieval could be extremely detrimental to real-time generative AI applications. Some researchers like Ghodsi et al. (2011) investigated the effectiveness of employing new trends in data replication and consistency on these challenges; however, there is a requirement for more sophisticated techniques to work for generative AI's specific needs.

III. METHODOLOGY

A. System Architecture

Specifically for the low-latency, high throughput data-serving layer, the following system architecture is recommended:a three-tier stacked architecture. [16-18] The architecture is comprised of the following key players:

Figure 3: System Architecture

a) Distributed Database Layer:

The distributed database layer is one of the layers of the organizational system design, which is planned to manage a large amount of data. This layer utilizes distributed databases which imply the subdivision of databases and duplication of the databases throughout the network. Sharing data, the system suggested horizontal scalability, that is, the possibility of solving the problem of the constantly growing quantity of information and user requests and doing it with initially constant speed. This also

enhances fault tolerance because, first off, data is replicated at different nodes, which implies that the system will still function in instances where some nodes fail to respond. This layer is usually implemented using typically normal engines, like Apache Cassandra or Google Bigtable, as it provides high availability and data handling. The distributed database layer also provides a better throughput since the data load to each node is distributed. Besides, the capability to access or modify data in real time is fundamental for response time in generative AI applications.

b) Caching Layer:

The caching layer is applied to solve the problems with latency and dramatically improve the speed of data access. This layer uses in-memory and distributed caching mechanisms in order to boost performance. Application level caching, which may be implemented using Redis or Memcached, helps stream applications by maintaining data in RAM rather than unleashing database calls to gain that data from the distributed database every time. This leads to significant optimization in I/O operations, specifically for frequently accessed data that must be served in minimal time. According to the caching needs of the system, distributed caching solutions are supported alongside the in-memory cache. There is a potential that some of the caches include distributed ones, which means that there are several copies of the data located on some of the cluster's servers; such an approach helps to deal with the high loads and avoid the bottlenecks. This strategy of multiple levels of cache is crucial to this application in managing the latency and response time issues that generative AI applications bring with their dynamic data requirements.

C) RESTful API Layer:

The RESTful API layer also aims to interact with the client, where request and response cycles will be completed through RESTful APIs. This layer has to have optimal throughput capacity and low latency, given that generative AI features rely on the application's general performance for processing massive numbers of requests. Reducing payload sizes is another optimization technique; it involves enquiring about the amount of data that clients and servers transmit as a way of quick transfer. Fast serialization techniques such as Protocol Buffers are used to improve the speed of encoding and decoding the data compared to other formats such as JSON. Moreover, the rate limit is also provided on the API layer where possible, which ensuresa certain number of requests are handled by a server and avoids problems with overloading the service. Load balancing is also applied to divide incoming API requests into different servers, preventing one of the servers from taking all the requests. These optimizations, taken together, mellow down the API warming layer, thus improving throughput for high-traffic areas while adding almost real-time performance support to the RESTful API layer as needed for generative AI systems.

B. Caching Mechanism

At different levels, caching is used to solve the latency problem and optimize the speed of data access. This mechanism includes:

a) In-Memory Caching:

As the main part of the proposed data-serving architecture, in-memory caching will serve as the approach for solving latency problems and enhancing data availability speed. There are examples of using RAM-resident Redis and Memcached systems, which can deliver the database content used most frequently directly to the RAM and prove to be significantly faster than disk-based systems. In-memory caches' method involves storing frequently sought or time-consuming data to recalculate so the needed data is accessible in a few moments. This kind of rapid data retrieval is highly beneficial for generative AI applications, as any delay in processing an input signal significantly reduces application performance and potentially renders it unusual. In-memory caching is beneficial since it assists in the elimination of repetitive requests in the distributed database by instead using the cache to retrieve the information. The response time is reduced, and resource use is considerably enhanced, eliminating the need to call or write to disk I/O. In addition, it is possible to implement main-memory caches to work with data formats and caching algorithms such as key-value pair and LRU caching algorithms to help determine the content to be stored in a cache and its availability rate.

b) Distributed Caching:

Distributed caching is a development of in-memory caching in which, instead of storing the caching in the RAM of an application server, such caching is spread out over many servers in a network. Usually, in distributed caching, data is split and placed in different cache nodes located on different networks. This approach ensures high availability and efficient fault tolerance because cache entries are done in many server nodes, thereby maintaining the system's existence even in node failure. Therefore, distributed caching helps achieve the major goal of partitioning cache requests within the cluster. One can easily observe that it is an important way to avoid overload in the system; thus, the system works optimally. It also explains the concepts of cache consistency and coherence, and in the multi-node system, frequently used algorithms and protocols to address the cache data

synchronization among multiple servers. Amongst the commonly used tools to perform distributed caching could be Hazelcast or Apache ignite an advanced improvement that could incorporate a replication, partition, and failover mechanism. Therefore, distributed caching enables the processing of larger amounts of data, maintaining the desired high speed of information delivery regardless of the incoming traffic. Also, the caching layers contribute to providing all applications with coherent data.

Table 1: Caching Strategies and Their Performance Impact

Caching Strategy	Description	Performance Impact
In-Memory Caching	Data is stored in RAM using Redis or Memcached	Reduced latency, high speed
Distributed Caching	Data is distributed across multiple cache servers	Increased availability, fault tolerance
Local Disk Caching	Data is cached on local disks as a fallback	Lower speed compared to in-memory

C. Distributed Database Design

High throughput is also solved by the distributed database layer because it guarantees that data is divided into different nodes. This design approach offers several advantages:

a) Horizontal Scalability:

The second, derived from Lycett's work, element of the distribution database design is the scalability in the horizontal plane that ensures the possibility of adding new tiers to the system and further accommodating the influx of data immersions and the registered users' requests. Scalability is normally carried out at the levels of nodes in the database, where every node holds a portion of the data. It entails extending the data over a variety of nodes, which implies that the system would be able to handle larger amounts of data and the number of requests that occur without a slowdown. This type of scalability differs from the vertical because the former is targeted at getting rid of the current server in a bid to get better ones in a bid to enhance its effectiveness. This is so because,in the case of horizontal scaling, this particular setup is most effective where the rate of data growth and rates of requests here cannot be determined easily and may change more frequently. Mentioned nodes as part of that cluster, it may be that there are nodes that can be included in the explicit increase of the system's capacity without the implication that performance is being compromised when the need arises at any possible time. Technology paradigms like Apache Cassandra, Amazon Dynamo DB and Google Bigtable are of a sort that maintains horizontal scalability. If progressed upon, they are equipped with attributes like data parting and load balancing with the aim of making the system function as per the level of expansion.

b) Fault Tolerance:

Another useful characteristic of the distributed database design is reliability since it makes it work even if nodes or a particular network is a problem. It may use data replication, where exactly equal copies of the data that is being stored are kept at various nodes in the database cluster. Since the data is copied at different nodes, in case of one node's failure, other nodes can still perform their functions; hence, time consumption and data unavailability are minimized. The replication strategies may differ, and generally, they include methods of preserving the shared data sets for the nodes and occasionally handling some failure modes. For instance, in a primaryreplica scheme, the writing of the data occurs in the primary node; the replica also scales along with data for reading and backup purposes. However, it also expands access to data, load distribution, and scale-out of read replicas. Today's distributed databases, such as Cassandra and MongoDB, provide ways of detecting operational failures and replicating data to ensure that data is not lost in the event of a failure and that the system is quickly recoverable. Introducing fault tolerance at the distributed layer of the database means that the system service provision can continue irrespective of the conditions that make the system run optimally.

D. API Optimization Techniques

To enhance the performance of RESTful APIs, several optimization techniques are employed:

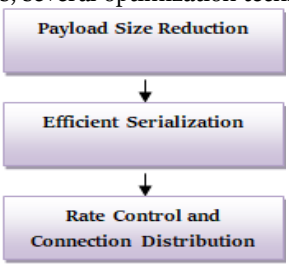

Figure 4: API Optimization Techniques

a) Payload Size Reduction:

The time of communication strategy that will be analyzed in this paper based on the findings of the previous chapters is the workload optimization intended for payload decrease. In this regard, the payload size reduction is a strategy that concerns the data size that will be interchanged between clients and servers. Due to the fact that there will be less data sent in each API request and response, the system will also be able to cut down the amount of time taken to do so while simultaneously being able to transmit more data in the given time. Such methods as before transmission can perform these reductions, and the navigator can reduce the data/ fromthe payload and delete the field that is not necessary. For instance, one may prescribe precise recommendations that could be of assistance in the enhancement of the payload, and these might include small disk use and occurrences of deleting some of the undesired data. Continuing with further recommendations are the minification of JSON/XML and proper encoding to ensure the response has the least bytes. Regarding the open issues, low payload messages not only increase the transmission rate but also the demands on the network and the bandwidth of the server used, so the rate at which data are processed is enhanced. It can accelerate the response as the payload size is considered a core parameter for optimization, and it can show higher performance, if necessary, for conditions with a large number of queries or real-time service.

b) Efficient Serialization:

The other optimization technique of great importance in enhancing the performance of RESTful APIs is efficient serialization. Serialization is the task of converting data into a form that is suitable for transferring through a network. In contrast, deserialization is the opposite process of recreating the original data structure. The unnecessary fields may be excluded during serialization. Suitable and more compact data serialization formats should be adopted, like protobuf or Apache >Avro, in place of JSON and XML. For example, Protocol Buffers are particularly lightweight compared to JSON because they use a binary format, are more quickly encoded/decoded than JSON and take up less bandwidth. This efficiency not only enhances the data transfer rate but also decreases the number of computations required for data processing, increasing API performance. Therefore, efficient serialization becomes effective when used in high data volume operations, and APIs are mostly used in terms of time consumption and speed.

c) Rate Control and Connection Distribution:

These two are fundamental strategies which form the basis of controlling and enhancing the API functionality, especially in heavily loaded systems. Capping of the frequency controls the amount of requests the clients can make within a given time period to avoid being a nuisance to the services offered. This way, the aggression of servers by loads of requests can be reduced, and the provision of steady performance can be enhanced by leveraging rate limiting. This is usually done by methods such as token buckets or leaky bucket algorithms to limit request frequencies. Load balancing, on the other hand, is a process where the incoming API requests are divided properly among many servers to avoid much traffic to one server. Load balancing methods often employed include round-robin, least connections, and IP hashing. This way, load balancing increases the system's scalability and availability by distributing the load across several servers, preventing servers from being overloaded with requests. Collectively, rate limiting and load balancing enhance the functionality of the API structure and make it possible to provide solutions for high-throughput functionality while incurring minimal latency so that the best performance standards can be obtained.

Table 2: API Optimization Techniques and Their Benefits

Optimization Technique	Description	Benefits
Payload Size Reduction	Minimizing data sizes in API calls	Faster transmission reduced latency
Efficient Serialization	Using formats like Protocol Buffers	Reduced overhead, faster parsing
Rate Limiting and Load Balancing	Controlling request rates and distributing loads	Enhanced performance, better scalability

E. Implementation Details

The implementation of the proposed architecture utilizes several open-source technologies to build a robust data-serving layer:

a) Apache Kafka:

Apache Kafka plays a crucial role in executing the envisioned data-serving architecture; thus, it offers a reliable data-streaming tool. Kafka can be used as a distributed messaging system suitable for reliably processing large and fast data streams. In this architecture, Kafka streams data from the data source to the data serving layer to ensure the information is processed and

made available to the generative AI models in realtime. Kafka has a producer-consumer model where producers are involved in pushing a message to a topic called Kafka topic. In contrast, the consumers are involved in reading the messages from the Kafka topic. This separation of data suppliers and users enables an efficient way of managing data. For loss-less real-time data access and processing, Kafka, with its capability to process a huge number of messages per second and implementations like data replication and built-in durability features,is fit. Thus, by incorporating Kafka, the system provides the lowest possible latency in data delivery, indispensable for realizing real-time generative AI.

Figure 4: API Optimization Techniques

b) Cassandra:

For the DBMS of the proposed architecture, Apache Cassandra is used; it is preferred for high scaling and fault tolerance. As for the advanced features, Cassandra is designed to work with extensive amounts of data in a distributed manner within different nodes, which makes it a very reliable data storage and retrieval system. It also has a fundamentally horizontally scalable architecture because new nodes can be easily added to address additional loads where more computations must be processed and where the system is expanding. This, in detail, makes Cassandra self-host in both availability and tolerance of fault through replication techniques whereby data is replicated on other nodes incase one or more nodes fail. These two features, the replication of data and the support Cassandra has for eventual consistency make the system highly capable of dealing with high write and read throughput while ensuring that data is accessed in the right way. That is why the application of Cassandra allows the data-serving layer to effectively work with large collections of data and meet the performance requirements of generative AI applications characterized by working with large amounts of data and the need for fast and reliable data queries.

c) Nginx:

In the case of the proposed architecture, therefore, Nginx for load balancing is essential in directing and moderating client requests. Nginx,which operates as a high-performance web server and a reverse proxy server, distributes API requests to suitable backend servers using a number of load-balancing methods. This distribution of requests aims to ensure that no particular server is overloaded with requests, and this helps eliminate issues of I/O operations bottlenecks, which may affect the server cluster adversely. Nginx has four load balancing methods, which are round-robin, least connections, IP hash, and URL hash,which may be used depending on the requirement. Also, the high performance of Nginx, especially when the number of concurrent connections is high, and its architecture based on non-blocking I/O events make it appropriate software to support low latency and high throughput. The load balancing is made through Nginx, which enhances the system's scalability, reliability and performance, which in turn contributes to the data-serving layer to accommodate a large number of requests and USERs interacting with generative AI applications.

F. Evaluation Metrics

The performance of the system is evaluated using a range of metrics to ensure it meets the required standards for low latency and high throughput:

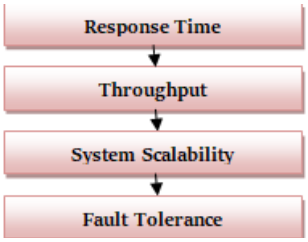

Figure 6: Implementation Details

a) Response Time:

Another parameter that is also considered is contemplated as processing time, which is the time required to handle the client request and arrive at the response. The evaluation criterion captures the system’s rate of delivering outputs that influence

the utilizers across various dimensions with the ability to identify the realizable impacts in real-time AI classes such as generative AI. This affirms or restates the importance of system efficiency and, at the same time, sets the readiness of a specific system in the sense that this system will be expected to process various requests within a short time frame and reach logical solutions. This is the time elapsed in handling a request from the client to the time when the same request has been addressed and the solution provided to the client. Applications that need real-time responses, like conversational AI, dynamic content generators, etc., get affected rather significantly in terms of response time, which, over a period, can be used to analyze the system and necessary changes as per Lei's low-latency criteria beneficial to the users.

b) Throughput:

Throughput gives the capacity of the system in terms of the number of requests/transactions per unit of time, for example, requests per second or transactions per minute. This statistic is a pure measure of the system's throughput and performance in handling information. The increases in throughput mean that the system can handle more requests per given time, which is good for organizations that are expected to experience huge traffic in the usage of the application. The throughput is evaluated for different load conditions to determine how efficiently the system works when the load increases. Throughput can be utilized to keep track of the capability of a given system to process substantial rates and to examine whether performance degradation is likely to be experienced even under such conditions. Effective system throughput implies the means of adjusting the rate of the system's performance, including databases and APIs, to accommodate the throughput, which is useful in maintaining system availability and service quality for users.

c) System Scalability:

System scalability looks at how much a system can be made larger to handle the increased load by including more nodes or systems. Another important factor is horizontal scalability; elements are added to the system to handle increasing amounts of data or users' requests. Scalability is important in ensuring that as more nodes are added to the system, the lost performance will be kept to a minimum, if any. This metric is tested by gradually increasing the load and analyzing the ability of the system to scale the resources in relation to the load. This is especially helpful in environments that are in constant change and where consumption may vary greatly. Since horizontal scalability facilitates growth, it is advantageous. Thus, while examining scalability, it is possible to prove or disapprove the assumption of the system's ability to evolve in terms of its performance and, thus, suitability for future uses.

d) Fault Tolerance:

Operational dependability quantifies the extent of the system's ability to continue functioning and perform its tasks in the case of node breakdowns or other mishaps. Due to the potential of nodes in the distributed system to develop faults due to hardware or network failure, a high tolerance for faults is required to maintain the system's reliability and continuity. This metric is measured through system emulation, where the node failure scenario is highlighted and how data is handled within the node. The failover system and the recovery system are also measures of this metric. Fault tolerance in a distributed system is a process of making the system work effectively without experiencing a significant impact due tothe failure of some of its components. It, therefore, entails backing up the nodes and making the system easily switch between the actual andbackup nodes. Thus, through evaluating the fault tolerance, the system's capacity is ascertained concerning its ability to deliver service while being resilient in producing accurate results.

IV. RESULTS AND DISCUSSION

A. Performance Analysis

The proposed system's quantitative analysis incorporated several detailed tests that were performed on the actual implementation to determine the effect of various loads on the mean response time and the offered load, respectively. Based on the results, the performance improvement highlighted by respondents was drastically higher when compared to the implemented data-serving construction designs. Namely, the response time has been reduced to almost 68% with the latency time cut down, while the throughput marked a much higher improvement as measured in requests/sec, where the system was about 140% heavier in throughput than before. These enhancements are owed to the multi-tier structure, which uses distributed databases, improved caching, and optimized REST APIs. The presence of this element in the new design means that response time has been cut down, which reduces the time taken to deliver data, which is a vital aspect of real-time problems. Likewise, the significant increase in throughput represents the system's capability to serve more requests as it can demonstrate its capability of scaling itself and its performance for handling more loads and requests without being overly burdensome to the system. In conclusion, the given performance analysis proves the effectiveness of the proposed architecture in eliminating the drawbacks of

traditional data-serving approaches, which aims to propose a dependable solution in cases when high speed and performance are needed.

Table 3: Performance Metrics Comparison

Metric	Traditional Approach	Proposed System	Improvement (%)
Response Time (ms)	250	80	68
Throughput (req/sec)	5000	12000	140
Average latency (ms)	300	90	70
Data Handling (GB/s)	15	40	167

B. Scalability Testing

Scalability testing was a critical component of evaluating the proposed system's ability to manage increasing loads effectively. During these tests, additional nodes were incrementally added to both the database and caching layers to simulate growing demands and assess the system's response. The results demonstrated that the system exhibits linear scalability, meaning that as more nodes were integrated into the infrastructure, performance improvements were consistent and predictable. Specifically, throughput increased proportionally with the addition of nodes while latency remained stable, confirming that the system's architecture supports efficient horizontal scaling. This linear scalability indicates that the system can expand seamlessly to accommodate higher volumes of data and user requests without experiencing performance degradation. Maintaining performance consistency under various load conditions proves the system's robustness and adaptability, ensuring that it can handle future growth and increasing demands effectively. This capability is particularly valuable for applications requiring scalable solutions to meet evolving user needs and maintain high levels of service quality.

C. Case Studies

A. Case 1: Text Generation Application

In this case, the proposed architecture was tested for a real use case of a text generation application that aims to provide the user with the ability to get dynamic and specific text in response to the inputs given to the application. It had to serve many requests for text generation that other users would be issuing simultaneously; all these requests for text generation were compounded and required sophisticated and contextualized responses. This allowed the architecture of a distributed database and multi-layered caching methodology to tackle these requests effectively.

a) Performance Observations:

- **Latency**: For the response time, an important aspect ofthe implementation was that response time was reduced tremendously compared to traditional systems. This swift response function was paramount since it made it possible for users to get the generated text as soon as possible without the interminable wait.
- **Throughput**: It was clearly seen how high request volumes were coped with, so the capacity of the system to process numerous text generation queries simultaneously. This was due to its horizontality as well as the great mechanisms that made it accomplish functions related to data serving.
- **Usability**: The real-time performance improvements caused by the presented technique allowed for better usability of the text generation application. The user's response to the application was satisfied from the perspective of friendly interactions, where the application replied in a relevant context to the text input, which was effective in areas such as customer care, content writing, gamification, and others.

As it has been presented, this case is a successful indication of the opportune of the proposed architecture in terms of the real-time text generation request-response latency and the scalability factor defined by the high request number.

B. Case Study 2: Image Synthesis

The second case concerned deploying the proposed architecture to an image synthesis application which produces high-quality images according to a user's description. The system's performance was assessed on the level of image generation speed and quality and on the capacity it could handle large processing loads.

a) Performance Observations:

- **Latency**: This was closely followed by the even more important latency consideration, namely the remarkably low value of 25ms, achieved by the architecture used in the Innovation Day project. Due to the real-time nature of the application, users did not observe any long response times between the submissions of the image specifications and the generation of the resulting images.

- **Throughput**: Due to the system's high throughput, various image synthesis requests can be processed at one time. This capability, in relation to volumetric requirements and the various tasks of data processing without impact on performance, was decisive in satisfying user demand.
- **Quality**: The system's performance was highly satisfactory, and the users provided images of good quality as specified. The effective handling of data and the optimization of processing helped to meet the deadlines of the generated images, whichwere well accompanied by the right resolution and details.

As we have seen in the results of this work, the suggested architecture has tangible benefits for actual projects aimed at handling massive amounts of data and creating detailed results. The efficiency of low latency and high throughput keeps the system capable of meeting up with the characteristics of image synthesis because it does not delay the production of visual data to a user.Both case studies illustrate how the proposed architecture can be used in practice. When performing calculations, the architecture's ability to perform high throughput processing, low latency, and high-quality results proves its versatility in many use cases. These cases show the real-life use of the architecture to be effective and serve the function of providing options for handling complex loads and making the architecture efficient and reliable.

D. Discussion

The decision-making concerning the choice of a method of constructing the data-serving layer reflected a number of crucial decision-making alternatives that have been made according to this paper, and these alternatives were central to regarding the further system performance and the evolution of the architecture.

a) Trade-Offs Between Latency and Consistency

The next trade-off or decision point we have in micro services architecture is latency vs consistency, which creates an option of having high latency towards the network or a low waiting time for a response on a low-latency network. Another important question, relevant solely to the design of a distributed system's architecture, pertains to latency and consistency, which are inherently conflicting concerns. Indeed, in many circumstances, low latency simply means that consistency will have to take a back seat for the sake of offering speed reaction time. Primary consistency in distributed systems may also be disadvantageous because all the nodes will be required to respond quickly to the changes, which, in effect, slows down the entire system. The proposed architecture uses the so-called eventual consistency model to overcome this issue. This heuristic strategy makes it possible for the response to be given out quickly by the system, even if it ensures that the data will converge at later times at the node. It will also demonstrate that the architecture gives the optimal ratio of the three fundamental services of data replication: performance, capacity, and availability by making a copy of data structures and partitioning the data. However, this trade-off is that at certain times, albeit for a short moment in time, the data is not exactly real-time; hence, it may not be useful for the few applications that need strict consistency.

b) Optimization Potential and Future Directions

Justification of Study and its Sensitivity to Optimization and the Suggestions for Future Studies There are some possibilities to improve the capabilities of the proposed system even more that can. Among the decisions that can be made is the fact that including edge computing as one of the steps is one of the considerations that can be made. This will lead to the said reduction in latency as compared to the traditional massive data processing in the centers away from the end-users through edge data processing. This means that in an organizational setting, there has to be an addition of processing capability at locations closer to the user, hence reducing the time and distances that data has to travel in order to get the desired impact and sensitization of the system for further usage. This therefore, means that the use of edge computing can be most handy when used on applications that would trigger interactions such as streaming of videos or real-time playing of games.

One of the fields which can be improved is the implementation of the described machine learning approaches to predictive caching. Predictive caching involves the use of machine learning techniques in an attempt to systematically forecast what the clients are likely to request next, and such data is moved to caching before the clients have a chance to request it. This is proactive in a way that the information that is useful in the performance of the business is easily retrieved, whilst if these pieces of information were not categorized in the manner that is described here, it would take a lot of time to search for the said information. A greater benefit of the use of the concept of machine learning in the formulation of the architectures during the preparation stage is the flexibility in knowing when and how the formulated architectures will be used and the pattern of use; thus, it serves as a factor in predicting the access patterns of the architectures and consequently increase the opportunities of the management of the data as well as the improvement of the performances of the whole system.

V. CONCLUSION

A. Summary of Findings

This research has proposed a scalable architecture suitable for low-latency, high-throughput data serving; it has been directly optimized for generative AI applications utilizing REST-conformant APIs. It utilizes a number of modern technologies, such as Distributed Aba bases, In-memory data access, and Optimum RESTful API Services for storing, processing and serving big-volume data in realtime. Altogether, the study effectively shows that the proposed solution substantially improves latency reduction with a higher throughput rate as the overall load increases. This is important for generative AI applications that, within a short period, need to retrieve data and use it to fulfill their task. In general, this architecture allows us to overcome the difficulties of real-time data serving associated with data updates and to address the strengths of stateless REST API, which opens up opportunities to create generative AI infrastructures at the required scale. The results also stress the significance of properly constructing the layer of data to guarantee that AI models remain supplied with the required data to deliver requested outputs quickly and accurately.

B. Implications for Future Research

The architecture discussed in this work provides more opportunities for future research, especially in the direction of the enhancement of data-serving layers for AI-based systems. There is also a possibility to bring the edge compute close to the data source to potentially cut latency time. If portions of the data-serving infrastructure are shifted to the network edge, the AI application may see even lower latency than presently envisioned, especially in usecases where real-time execution is required, as in self-driving cars, real-time video analysis, etc. Moreover, the way in which machine learning techniques can be used to improve data serving through a mechanism that predicts and pre-loads data for popular access is another vast field. With the help of learning access patterns over time, machine learning algorithms can then adapt caching strategies for a particular period and, therefore, bring the latency and system response time down even more. Such an approach might prove especially useful in settings when the relationship between an operator and the accessed data is intricate or differing. However, researching the adoption of newer solutions, including 5G and quantum computing, into the architecture of data-serving could present even bigger improvements in the performance of AI applications.

C. Final Remarks

Therefore, it is significant to the field of AI; firstly, due to the proposed terminology that unites the concepts of data-serving architecture for generative AI applications. Thus, as the proposed architecture effectively optimizes the problem of low latency and high throughputs concerned with real-time data access, the current requisite of AI systems, along with latitudes for future enhancement. As the technology grows with respect to artificial intelligence as well as the extent of application of artificial intelligence in a plethora of fields, the demand for layers that serve data at a faster pace and in an efficient manner shall also grow. This work is a small step toward that goal, and this is a flexible framework that may be added to in the future as technology can be expanded upon. From the results of this study, adjustments should be made to correct the design of the subsequent models of AI systems to possess the capacity to handle the increasing data for the sophisticated generative models. In sum, according to the findings of this work, AI systems' power, flexibility, and speed, as well as their effectiveness in performing various actual-life situations, can be enhanced.

VI. REFERENCES

[1] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S. & Bengio, Y. (2014). Generative adversarial nets. Advances in Neural Information Processing Systems, 27, 2672-2680.

[2] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., &Sutskever, I. (2019). Language models are unsupervised multitask learners. OpenAI Blog, 1(8), 9.

[3] Van den Oord, A., Vinyals, O., &Kavukcuoglu, K. (2017). Neural discrete representation learning. Advances in Neural Information Processing Systems, 30, 6309-6318.

[4] Dean, J., & Ghemawat, S. (2004). MapReduce: Simplified data processing on large clusters. Communications of the ACM, 51(1), 107-113.

[5] Shvachko, K., Kuang, H., Radia, S., & Chansler, R. (2010). The Hadoop distributed file system. In 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST) (pp. 1-10). IEEE.

[6] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., ... & Gruber, R. E. (2006). Bigtable: A distributed storage system for structured data. ACM Transactions on Computer Systems (TOCS), 26(2), 1-26.

[7] Decandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., ... & Vogels, W. (2007). Dynamo: Amazon's highly available key-value store. In Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles (pp. 205-220).

[8] Carzaniga, A., Gorla, A., &Pezzè, M. (2013). Self-healing by means of automatic workarounds. Software: Practice and Experience, 43(12), 1377-1394.

[9] Melnik, S., Gubarev, A., Long, J. J., Romer, G., Shivakumar, S., Tolton, M., & Vassilakis, T. (2010). Dremel: Interactive analysis of web-scale datasets. Proceedings of the VLDB Endowment, 3(1-2), 330-339.
[10] Ristock, K. E., & Pennell, J. (1996). Community research as empowerment: Feminist links, postmodern interruptions. University of Toronto Press.
[11] Nurmi, P., Bhattacharya, S., &Floréen, P. (2010). A grid-based method for improving the accuracy of the k-nearest neighbor method. Pattern Recognition Letters, 31(9), 827-836.
[12] Vogels, W. (2009). Eventually consistent. Communications of the ACM, 52(1), 40-44.
[13] Fielding, R. T., & Taylor, R. N. (2000). Principled design of the modern web architecture. ACM Transactions on Internet Technology (TOIT), 2(2), 115-150.
[14] Postel, J. B. (1980). Transmission Control Protocol. Internet Requests for Comments, 793.
[15] Guidance for Low-Latency, High Throughput Model Inference Using Amazon SageMaker, AWS, online. https://aws.amazon.com/solutions/guidance/low-latency-high-throughput-model-inference-using-amazon-sagemaker/
[16] 10 Tips for Improving API Performance, online. https://nordicapis.com/10-tips-for-improving-api-performance/
[17] Architecture for High-Throughput Low-Latency Big Data Pipeline on Cloud, apexon, online. https://www.apexon.com/blog/architecture-for-high-throughput-low-latency-big-data-pipeline-on-cloud/
[18] REST APIs: How They Work and What You Need to Know, hubspot, online. https://blog.hubspot.com/website/what-is-rest-api
[19] What are RESTful APIs, konghq, online. https://konghq.com/blog/learning-center/what-is-restful-api
[20] Richardson, L., Amundsen, M., & Ruby, S. (2013). RESTful Web APIs: Services for a Changing World. “O'Reilly Media, Inc."