International Journal of Science and Research (IJSR)
ISSN: 2319-7064
SJIF (2022): 7.942

Exploring the Integration of Azure LLMs, Serverless Computing, and DevOps: Advancing Paradigms in API Design and Automated Testing

Sri Rama Chandra Charan Teja Tadi

Lead Software Developer, Austin, Texas, USA

Abstract: ***The fusion of new technologies like Azure's language models, serverless computing, and DevOps practices is redefining how developers organize APIs and automate testing, delivering fresh solutions to old problems. Although large language models are in their early days, it increasingly appears that they will revolutionize the natural language processing abilities of APIs. This confluence of technologies, although not yet entirely there, can hopefully put an end to the age-old challenges of software engineering, including making the interfaces more intuitive and simpler to create. Serverless computing, which has garnered considerable attention in recent years, inherently fits into the bursty and compute-intensive nature of language model processing. Through serverless architectures, businesses can try out the use of language models within their applications without worrying about the complexity of elaborate infrastructure. This confluence not only maximizes the use of resources but also provides advanced language processing capability to smaller teams and startups as well as big corporations, at their fingertips. The integration of DevOps practices within this new tech trinity creates a turning point axis, both as a catalyst and a unifier. This type of alignment can be thought of as a "Continuous Adaptive Development" (CAD) process, wherein the classic DevOps cycle is supplemented with AI-driven intelligence and serverless responsiveness. Besides this, the CAD model also suggests that as serverless architecture and language models are developing, in the same way, the DevOps practices that form them should develop, forming an interdependent equation that urges constant improvement in the three concepts. This theoretical framework stipulates that organizations embracing the CAD approach will, besides realizing more development agility and system reliability, experience a paradigm shift in how they think and manage the software development lifecycle of an AI-fueled, serverless era.***

Keywords: language models, serverless computing, DevOps practices, Continuous Adaptive Development, API automation

1. Overview of Azure LLMs, Serverless Computing, and DevOps

Azure's continuously evolving set of tools in LLMs, serverless computing, and DevOps created an integrated platform designed to address the evolving demands of companies today. By reducing AI adoption barriers, streamlining infrastructure management, and encouraging agile development practices, Azure positioned itself as a driver of innovation.

1.1 Understanding Azure's Emerging Language Models

Azure's emphasis on large language models (LLMs) is a sign of a strategic move toward democratizing AI, making it more accessible and ubiquitous, and making advanced models available to developers and businesses. These models are integrated into Azure's platform, like natural language understanding, text generation, and conversational AI, as the foundation for much of what shows up in customer service, content creation, and data interpretation. As awareness of general AI was still growing, Azure's LLM solutions were designed to close the gap between the research capabilities of AI and real-world business applications.

The integration of Azure LLMs with Azure Machine Learning services streamlined processes so that developers could train, fine-tune, and deploy models effectively [4]. Since the use of AI was in its beginning, Azure focused on scaling and security so that businesses could integrate LLM functionality securely without making large infrastructure investments. The modularity of the Azure LLM framework enabled developers to customize models according to industry requirements, driving innovation without getting hampered by complexity.

1.2 The Rise of Serverless Computing

Serverless computing was a revolutionary method of performing cloud infrastructure without worrying about the underlying servers and concentrating on code execution only. The nature of the serverless model is event-driven, by which applications automatically scale with demand and are economical [6]. Azure Functions was the best example of such a revolution, and it facilitated seamless integration with other Azure services and allowed developers to create scalable microservices and APIs without any overhead of server management.

Serverless was driven by ease of use and flexibility. Compared to conventional cloud architecture, the serverless model enabled teams to develop and deploy applications at high speed, which was extremely useful in the new AI and data-driven application ecosystem [5]. While it had its merits, issues like cold starts and vendor lock-in needed to be considered heavily. Its accommodation of today's development styles, however, has made serverless a core part of most cloud-native solutions.

1.3 DevOps in Modern Software Development

DevOps then emerged as the central methodology of modern software development, focusing on cooperation between development and operational teams towards speeding up software delivery cycles. The deployment of DevOps within Azure's infrastructure enabled the automation of tests, efficient CI/CD pipelines, and better deployment models [4]. This end-to-end culture for software development came with

a culture of constant improvement as now teams could deploy features quicker and on a consistent basis.

Automation was one of the pillars of DevOps. Azure DevOps gave strong implementations of version control, build automation, and monitoring, such that code changes would be able to transition from development to production seamlessly. It was especially crucial in environments where microservices and serverless architectures were used, where iteration had to be quick and scalability essential.

The integration of AI-fueled insights within DevOps protocols gained mainstream popularity, with machine learning models forecasting potential deployment issues or detecting delays in the CI/CD pipeline. Integration through AI and its value in streamlining development cycles and minimizing downtime became evident. The combination of Azure's LLMs, serverless features, and DevOps protocols resulted in a robust system allowing developers to innovate without sacrificing agility and reliability.

2. Foundations of API Design in the Context of Language Models

2.1 Principles of Effective API Design

Designing an effective API, especially in the context of integrating language models (LLMs), requires a balance between usability, scalability, and flexibility. The goal is to create interfaces that are intuitive for developers while being robust enough to handle complex operations. Clear documentation, consistent endpoint structures, and predictable response formats are foundational principles that foster seamless adoption.

An effective API should also emphasize security and performance. Implementing rate limiting, authentication protocols, and efficient data handling ensures that APIs remain reliable under varying loads [9]. As the integration of LLMs becomes more prevalent, maintaining minimal latency becomes critical, given that real-time applications like chatbots and recommendation engines heavily depend on swift data exchanges.

Scalability is another cornerstone. APIs designed to integrate with LLMs must handle fluctuating demands without degrading performance. Using microservices architectures and containerization strategies can help achieve this flexibility. Designing APIs to be modular allows developers to extend functionalities without overhauling existing systems, enabling smoother updates and iterations.

2.2 Initial Considerations for Integrating Language Models

Integrating language models into APIs presents unique challenges and opportunities. One of the initial considerations involves data management - how input and output data flow through the API. Since LLMs process vast amounts of text data, ensuring efficient data preprocessing and postprocessing mechanisms is essential to maintain performance and accuracy.

Another critical factor is resource allocation. LLMs, even in their early iterations, require significant computational resources. Implementing adaptive resource management strategies ensures that APIs can dynamically allocate necessary resources based on demand. Cloud-native solutions, such as serverless computing, provide an ideal environment for scaling resources efficiently [5].

Security and privacy considerations also take center stage. When dealing with language data, sensitive information can inadvertently pass through APIs. Developers must implement strong encryption protocols and data anonymization techniques to safeguard user privacy and comply with data protection regulations.

2.3 Early Use Cases for Language Model-Enhanced APIs

Developers are constantly seeking various applications that take advantage of the generative and interpretive nature of LLMs. Customer service is one of the most prominent applications where LLMs drive chatbots and virtual assistants and allow them to answer sophisticated questions and offer human-like interactions.

Another common application is content creation. APIs allow developers to embed language models within applications for the automatic writing of reports, generation of marketing copy, and program creation, specifically lowering man-hours and maximizing efficiency. NLU facilitation tools grow stronger and drive applications such as sentiment analysis, summarization, and entity recognition.

Developers continuously improve search capabilities with LLMs. The natural language processing capabilities of APIs allow recommendation and search systems to automatically learn from user feedback and experience and provide more contextual, personalized, and relevant recommendations. [3]. In learning platforms, LLMs customize learning experiences, adapting content and exams according to individual learning patterns.

3. Serverless Computing as a Catalyst for Innovation

Serverless computing enables developers to innovate at speed by introducing layers of infrastructure management and nudging cost-effective, scalable solutions. Its integration with contemporary application architecture makes it an incredibly useful ingredient for bringing together complex technologies such as large language models. While there are challenges to overcome, intentional design and application-specific tuning enable developers to get the most out of serverless computing, enabling the next wave of AI-powered applications to emerge.

3.1 Overview of Serverless Architectures

Serverless architectures redefine application deployment and development by disconnecting server management from software developers. Developers no longer run servers but write code, while the cloud provider is responsible for scaling availability and infrastructure maintenance. This architecture has an event-driven model where functions run on the

occurrence of given triggers like HTTP requests or database updates.

One of the defining features of serverless computing is that it is stateless - every function call is independent, hence making it fault-tolerant and scalable. Cloud services such as Azure Functions and AWS Lambda provide rich environments that are compatible with a variety of programming languages as well as integration possibilities. The pay-as-you-use model is also popular since it enables developers to manage costs in accordance with actual usage instead of pre-allocated resources.

Serverless designs also map quite naturally to microservices-based designs. By splitting applications into small, independently deployable components, businesses can maintain greater levels of flexibility and shorter time to develop. Modular design reduces complexity and makes updates easier, encouraging agile development patterns [7].

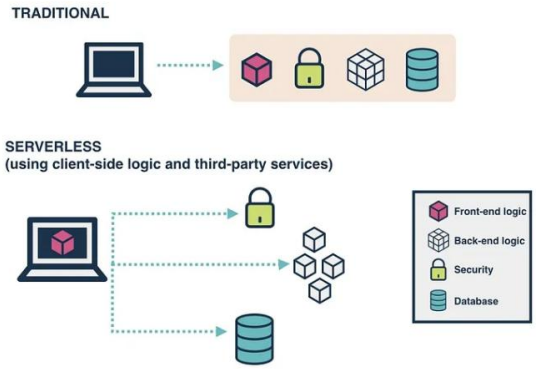

Figure 1: Traditional vs Serverless architecture
Source: Serverless Architecture and Applications

3.2 Benefits of Serverless Computing for Language Model Applications

The cost-saving and scalability of the serverless model render it especially fitting for application scenarios with big language models (LLMs). LLMs demand high computation during inference, yet applications might not always need processing. Serverless frameworks enable LLMs to scale on a dynamic level based on users' demand while using the resources efficiently and maintaining idle infrastructures without wastage.

Serverless platforms also facilitate the quick prototyping and iteration of LLM-based applications. Language model-based functions for text generation, sentiment analysis, or summarization can be run by developers without server-management overheads [6]. This speeds up the development of AI applications with low operating complexity.

Moreover, the event-driven nature of serverless computing suits the interactive workflow nature of LLM applications. For example, chatbots and virtual assistants can handle user queries via serverless functions with low-latency response at an economical cost. Serverless architecture also makes integrated workings with other cloud services simple, where data preprocessing and postprocessing workflows that are a common requirement in LLM operations become eased [8].

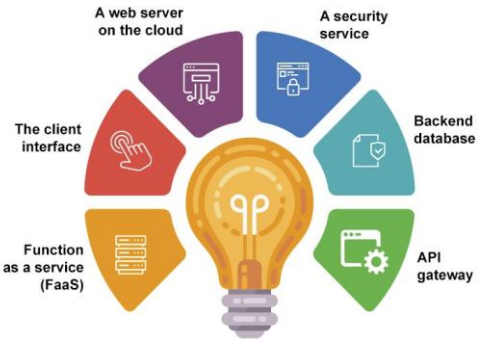

Figure 2: Key Components of Serverless Architecture
Source: What Is Serverless? Definition, Architecture, Examples, and Applications

3.3 Challenges and Considerations in Serverless Deployments

Though it has its benefits, serverless computing is not problem-free, particularly in scaling resource-intensive applications such as LLMs. Among its largest challenges is cold start latency - when a function takes seconds to be invoked for the first time following idleness. It affects performance in latency-critical applications, necessitating practices such as provisioned concurrency to help counter its effects [7].

Resource constraints in serverless environments are also stringent. Functions usually come with memory and execution time limits, which do not typically align with the demands of high LLM inferences. Developers need to optimize their models or employ hybrid strategies that blend serverless aspects with more capable compute resources where suitable [6].

Data security and privacy also come under the spotlight. As serverless functions tend to work on sensitive information, secure API gateways, encrypted data storage, and authentic authentication processes are essential. Security is not as effective with the volatile nature of serverless functions, so special ways of securing information must be used.

4. DevOps Practices Supporting Language Model Integration

4.1 Continuous Integration and Continuous Deployment (CI/CD) Fundamentals

Strong CI/CD pipelines are the keystone of software development today, particularly when applied to large language models (LLMs) within applications. Continuous Integration makes certain that code updates, such as those to LLM behaviors or model parameters, are tested and merged into the master codebase automatically. Continuous Deployment takes it a step further by deploying and updating features automatically, reducing touch and time-to-market.

In the event of LLM-driven applications, CI/CD pipelines have to handle model training, fine-tuning, and validation complexity. Workflows can initiate automatic retraining processes in the event of new data or model drift. Cloud

platforms such as Azure DevOps offer native support to handle such pipelines, making it simpler to deploy LLM updates without affecting code stability [4].

Security and compliance are also part of CI/CD for LLMs. Pipelines need to be automated with aggressive testing to avoid data leakage and make sure that model updates don't unwittingly introduce biases or vulnerabilities. Containerization tools such as Docker also introduce uniformity across development, test, and production environments [1].

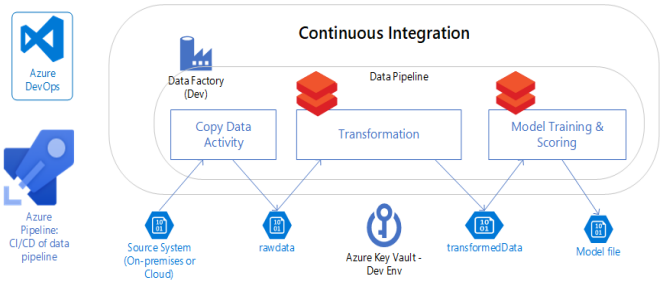

Figure 3: Azure continuous integration
Source: CI/CD data pipelines in Azure

4.2 Automating Workflows for Language Model-Driven Applications

Automation is at the core when developing with the dynamic character of LLM applications. Automated workflows, from data preprocessing to model deployment, minimize the potential for human error and speed up development cycles. Workflow automation platforms can automate complex processes such as data ingestion, feature extraction, and model evaluation so that each component operates in harmony within the pipeline.

Infrastructure-as-Code (IaC) is central to this context. Using Terraform or Azure Resource Manager, for instance, infrastructure configurations can be specified in code, allowing for reproducible and scalable deployment. IaC enables keeping development, staging, and production environments identical and making it simple to manage resources used for LLM training and inference [4].

Including monitoring instrumentation in these processes increases system resiliency. Monitoring actual resource utilization, model performance, and user behavior enables teams to tweak parameters in real time, resulting in higher end-user quality. Automation, beyond maintaining low operational overhead, also makes iteration cycles extremely fast and critical for AI-based applications.

4.3 Monitoring and Feedback Loops in Development

There ought to be continuous monitoring to preserve LLM-built application performance as well as trustworthiness. The effective monitoring strategies are the health of infrastructure, API latency, and proper output of the language model. Development teams are afforded the capability through monitoring key performance indicators to diagnose bottlenecks, sub-par performance, as well as any out-of-whack behavior upon detection.

Feedback loops complement this insight by allowing systems to learn from actual usage. User interaction creates useful information that can be used to retrain and improve models. For example, in chatbot use, user feedback can indicate where the model is having trouble correctly interpreting questions, which can be used to make targeted improvements.

Strong feedback is also helpful in detecting and removing bias. LLMs inherit biased elements in their training data, and constant user usage has the tendency to release such problems with time. The integration of ethical AI practices into monitoring allows teams to counteract bias in advance and maintain fairness in output from models.

With real-time analytics combined with adaptive feedback loops, there is a live development environment with models changing constantly based on user demand and system behavior. This cyclical approach is in line with DevOps, and continuous improvement and operational excellence are achievable.

5. Automated Testing Strategies for Language Model-Infused APIs

5.1 Importance of Testing in API Development

With the changing API development environment, the advent of large language models (LLMs) adds new dimensions of complexity. These have to be addressed with comprehensive testing approaches. Testing not only ensures APIs function as expected, but they also have to be consistent, effective, and secure across a range of usage cases. For language model-based APIs, testing assumes even greater importance as such systems will be processing dynamic, context-based inputs and producing accurate, coherent outputs.

Legacy testing practices - unit, integration, and functional tests - continue to be the basis. LLM-enhanced APIs, however, need extra layers of checks to examine model-specific behaviors. These include natural language understanding tests, response relevance tests, and tests to ensure outputs comply with ethical and business regulations. Performance testing is also important to ensure APIs react within tolerable latency boundaries, particularly in high-traffic applications [9].

Security testing is also among the pillars of sound API building. Given that LLMs handle such vast amounts of data, security measures against threats such as injection attacks or loss of data are critical. Security scans and penetration tests by automated software enable vulnerabilities to be identified with APIs following industry practices and data privacy laws.

5.2 Approaches to Testing Language Model Outputs

LLM output testing isn't just a case of confirming correct answers - it's ensuring fairness, accuracy, and quality in the output. A low-cost strategy is using test suites to address a wide range of inputs from common, edge-case, and adversarial examples. This will allow the model to reply to a wide number of requests without jeopardizing output integrity.

Metrics like BLEU, ROUGE, and perplexity give quantitative estimates of language generation quality. Qualitative metrics are still needed to detect nuances like coherence, sentiment matching, and contextuality. Human-in-the-loop (HITL) testing enables subject matter experts to inspect the model output, giving feedback to refine further.

Detection and reduction of bias are critical aspects of LLM assessment. It is possible for artificial intelligence to monitor model responses for biased text or offensive stereotypes to allow developers to modify training sets or model settings to support inclusivity and equity. Ongoing scrutiny of these phenomena ensures that ethics levels are high over the long term.

5.3 Tools and Frameworks for Automated Testing

There exist tools and frameworks that ease the automated testing of LLM-based APIs, from unit testing to intricate integration test cases. API performance is tested with top-grade tools such as Postman and LoadRunner, which allow developers to test heavy loads and validate system resilience. The tools aid in checking if APIs scale adequately based on changing user loads.

For language-specific testing, for instance, TensorFlow Extended (TFX) and Hugging Face's Transformers library provide native pipelines for model evaluation, deployment testing, and data verification. These facilitate end-to-end LLM integration's lifecycle from post-deployment to preprocessing.

CI/CD toolchains such as Azure DevOps and Jenkins further automate testing by integrating it directly into development. Automated test suites execute on each commit of the code, giving instant feedback and minimizing the chance of introducing bugs or performance regressions [4].

Security-oriented tools such as OWASP ZAP and Burp Suite allow comprehensive vulnerability scanning, protecting APIs from most attacks. Coupled with performance monitoring tools, these solutions offer end-to-end monitoring of API reliability and health.

6. Emerging Patterns and Best Practices

The deployment of large language models into high-performance, scalable, and secure environments calls for an attentive blend of security controls, optimization methods, and design patterns. With a microservices architecture, caching, load balancing, data privacy, and ethical operation in mind, development teams can develop strong and user-oriented solutions. Not only do these habits enhance system performance, but AI adoption is also made responsible and sustainable.

6.1 Design Patterns for Scalable Language Model Integration

Scalability is a core practice in effective large language model (LLM) implementations in contemporary applications. Utilizing established design patterns guarantees LLMs scale between loads without compromising consistency in performance. One very common practice adopted is the application of the microservices architecture, where the LLM feature is instantiated within a single service. It provides modularity for deployment, scaling, and upgrading without disrupting the entire system [1].

Event-driven architectures also improve scalability through the ability to support asynchronous processing. The isolation of user interactions from LLM computations enables applications to deal with different workloads effectively. For example, message queues and event streams enable several instances of an LLM to handle requests concurrently, enhancing throughput and robustness.

API Gateway patterns are also essential to scalable LLM integration. As the intermediaries between clients and backend services, API Gateways control request routing, load balancing, and security [9]. This pattern optimizes and controls LLM requests to avoid bottlenecks during high traffic.

6.2 Performance Optimization Techniques

Performance tuning in LLM-based systems is a matter of trading response times against computation requirements. Model distillation is a useful method wherein small, quick models are trained to mimic the operation of large LLMs. The technique minimizes inference latency at the expense of tolerable accuracy.

Caching facilities significantly improve performance by maintaining LLM answers to common requests. Applications can avoid repeated computations for common requests with the help of caching layers, which will render them faster and cheaper. Content Delivery Networks (CDNs) also possess the ability to cache static LLM outputs, further enhancing response times for users worldwide.

Load balancing is another key performance optimization feature. Sharing incoming requests between a cluster of LLM instances in a balanced way maximizes resource usage and avoids saturating particular nodes. Auto-scaling policies dynamically adjust the active instances count according to traffic patterns, keeping the system responsive across different loads [6].

6.3 Security and Privacy Considerations

Integrating LLMs into applications poses new security and privacy issues. With LLMs tending to work with sensitive user data, deploying strong encryption mechanisms is necessary. Encryption of both in-transit and at-rest data protects information from unauthorized access, maintaining industry regulations [1].

Authentication and authorisation controls enhance APIs with security. The utilization of OAuth 2.0 or JSON Web Tokens (JWT) makes certain only legitimate users and services have the capability of calling LLM endpoints. Auditing and continuous vulnerability testing forestall vulnerabilities before they take place, allowing real-time prevention.

Even the LLMs fall victim to privacy concerns. Differential privacy is utilized during the training of models to ensure that leakage of sensitive information is avoided. This is complemented with data anonymization to ensure that user-specific details are not inadvertently leaked out in response to models.

Biases can be identified, and ethical AI practices upheld to guarantee users' trust. LLM responses can be designed to automatically detect biased and offensive responses so that model tuning by developers is made feasible to uphold ethical practices. Handling an open data policy enhances users' confidence when LLMs are used with applications.

7. Future Directions and Opportunities

7.1 Potential Advancements in API Design Methodologies

API design trends will soon be revolutionized with the increasing need for intelligent and adaptive systems. The inclusion of AI-based decision-making within the API process can transform the way APIs manage intricate tasks. Adaptive APIs that can self-optimize according to usage patterns and environmental conditions could be the norm in the near future, easing the development process and user interactions.

The other advancement is semantic API evolution, beyond the conventional data exchange with additional contextual details. These ride on natural language processing (NLP) for enhanced intent capture and more natural, human-sounding interactions. This can vastly improve language model integrations, making APIs richer to cater to multiple user needs [9].

GraphQL and other adaptive query languages also provide avenues for API design optimization. These systems permit clients to ask for only the data they require, lowering payload sizes and enhancing efficiency. As more sophisticated LLMs emerge, such focused data retrieval systems will be more valuable.

7.2 The Evolution of Testing Paradigms for AI-Enhanced Systems

Testing practices are changing to address the specific needs of AI-augmented systems. Static testing methodologies cannot handle the dynamic nature of AI models, and hence, there is a move towards continuous and context-aware testing practices. AI-driven testing tools in the near future will likely create test cases autonomously based on learned usage patterns, simplifying the validation process and identifying edge cases more effectively.

Explainable AI (XAI) is also revolutionizing testing paradigms. As AI systems become more central to high-stakes decision-making, transparency and accountability are paramount. Testing frameworks that assess not only output accuracy but also the justification for AI choices will take center stage in building trust and regulatory adherence.

Simulation environments offer another avenue for advanced testing. Through virtual environments emulating real-world conditions, designers are able to test AI capabilities on difficult-to-replicate scenarios without the integrity of actual systems being compromised. Simulation environments support serious stress testing, which makes the system stronger and more reliable.

7.3 Preparing for a Shifting Technological Landscape

With technology scapes evolving, organizations have to employ future-oriented strategies to stay flexible and future-proof. Cross-disciplinary collaboration is one of them. The implementation of AI as a component of conventional software ecosystems requires more interdisciplinary collaboration between data scientists, software engineers, and domain experts. Such collaboration guarantees that deployments of LLM keep pace with technical capabilities and business objectives.

The proximity of LLMs to data sources allows for quicker data processing and quicker decision-making, which is necessary for the test automation of REST APIs, where real-time responsiveness and accuracy are paramount. [3]. This change toward a decentralized structure also introduces new concerns regarding data privacy and resource allocation that companies must address carefully.

Finally, ethical AI practices will increasingly shape how technology is designed in the future. As AI systems become more prevalent in aspects of our daily lives, developers will need to place a greater focus on transparency, fairness, and accountability. Establishing well-documented guidelines for data usage, incorporating robust bias detection mechanisms, and promoting diversity in AI development processes are essential steps in reaching sustainable and responsible innovation.

8. Conclusion

a) Scaling API Design Methodologies:

- The potential of API design is enormous, with scope for growth and innovation thanks to the rise of artificial intelligence, cloud computing, and edge technologies.
- API design methodologies will be more dynamic, modular, and context-specific to enable frictionless integration of sophisticated systems.
- Developers will use adaptive APIs that can self-adjust based on real-time feeds and changing user needs to design more natural and personalized user interfaces.

b) Shaping Testing Paradigms:

- Test patterns will extend beyond standard functional testing to incorporate tests for AI ethics, fairness, and explainability.
- Sophisticated simulation environments and AI-powered test agents will enable context-aware test cases to be generated, speeding up development cycles while maintaining system stability and equity.

c) Developing Resilient Technological Architectures:

- Technology development in the future will be based on resilient architectures that can handle rapidly evolving markets and emergent challenges.
- The use of decentralized technologies such as edge computing will decrease latency, increase data privacy, and enhance decision-making capabilities in real time.
- Those organizations that adopt such technologies will drive innovation, building future-proof, scalable, and responsible AI ecosystems that meet changing industry and user needs.

References

[1] E. Casalicchio and S. Iannucci, "The state-of-the-art in container technologies: Application, orchestration and security," *Journal of Grid Computing*, vol. 18, no. 3, pp. 401-424, 2020.

[2] R. Mahmud, K. Ramamohanarao, and R. Buyya, "Application management in fog computing environments: A taxonomy, review and future directions," *ACM Computing Surveys (CSUR)*, vol. 53, no. 4, pp. 1-43, 2020.

[3] M. Kim, Q. Xin, S. Sinha, and A. Orso, "Automated test generation for REST APIs: no time to rest yet," *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2022)*, Association for Computing Machinery, New York, NY, USA, pp. 289–301, 2022.

[4] Microsoft Azure, *Azure DevOps for CI/CD - Azure Machine Learning*, 2021.

[5] Datadog, *The State of Serverless 2022*, 2022.

[6] J. Schleier-Smith, V. Sreekanti, A. Khandelwal, J. Carreira, N. J. Yadwadkar, R. A. Popa, J. E. Gonzalez, I. Stoica, and D. A. Patterson, "What serverless computing is and should become: The next phase of cloud computing," *Communications of the ACM*, 2021.

[7] S. Eismann, J. Scheuner, E. van Eyk, M. Schwinger, J. Grohmann, C. L. Abad, and A. Iosup, "Serverless Applications: Why, When, and How?," *arXiv preprint arXiv:2009.08173*., 2020.

[8] H. B. Hassan, S. A. Barakat, and Q. I. Sarhan, "Survey on serverless computing," *Journal of Cloud Computing*, 2021.

[9] A. Tosato, M. Minerva, and E. Bartolesi, *Mastering Minimal APIs in ASP.NET Core: Build, test, and prototype web APIs quickly using .NET and C#*, Packt Publishing, 2022.