International Journal of Computer Trends and Technology
ISSN: 2231–2803 / https://doi.org/10.14445/22312803/IJCTT-V72I11P123

Volume 72 Issue 11, 220-227, November 2024

Original Article

Modern API Design: AI-First Architecture, Event-Driven Patterns, and Zero-Trust Security

Ganapathy Subramanian Ramachandran

Professional in Cloud Computing, Networking, and Security, California, USA.

Corresponding Author : ganapathys.ramachandran@gmail.com

Received: 09 October 2024 Revised: 10 November 2024 Accepted: 26 November 2024 Published: 30 November 2024

Abstract *- This paper presents a comprehensive framework for modern API design that addresses the challenges of integrating machine learning operations into traditional and contemporary API architectures. The proposed approach combines three key elements: an AI-first architecture design for efficient vector operations and model serving capabilities alongside traditional data operations, event-driven patterns that enhance ML workflows and standard request-response interactions, and zero-trust security principles adaptable to both ML workloads and conventional API usage. The research demonstrates how these architectural patterns can be effectively implemented to create APIs that support traditional web services and modern computational workload operations while maintaining system scalability, security, and performance.*

Keywords *- Event-driven architecture, Zero-trust security, AI integration, Feature store, Model serving, Vector operations, Distributed systems, Scalable systems, Edge computing, Real-time processing, Batch processing, Stream processing, Continuous authentication, Context-aware security, ML model security, AI-first design, Edge AI, Federated learning, Scalability, High throughput, Low latency.*

1. Introduction

Over the years, developers have relied heavily on traditional API approaches, such as REST, GraphQL, and RPC, to power web services. However, as applications become more data-intensive and complex, conventional methods show their limitations [1]. Three main challenges have become particularly apparent when building modern applications: First, complex data operations and high-performance computing present unique challenges that traditional APIs struggle to address [2]. The computational complexity of similarity searches across high-dimensional vectors and requirements for efficient data processing demand specialized processing pipelines beyond simple data retrieval and storage [3,4]. Second, real-time data processing and analysis pose significant architectural challenges. Traditional APIs, designed for simple data retrieval and storage, need more infrastructure for complex computations and real-time processing [5].

The complexity of modern computational workflows introduces new challenges in infrastructure design and scaling [6]. Third, streaming data processing capabilities in traditional APIs are limited by their request-response nature [7]. Modern distributed applications require continuous data streams for real-time processing and analytics, a challenge that traditional request-response patterns need help to address efficiently [8]. Transitioning from synchronous to asynchronous patterns has become crucial for high-performance systems, particularly for handling continuous data streams and long-running computations [9]. This paper presents a comprehensive framework that addresses these challenges through three principles across different API styles: (1) a next-generation architecture design optimized for high-dimensional data operations and efficient processing, (2) event-driven patterns that enable efficient real-time processing and state management, and (3) a zero-trust security framework for modern distributed systems. The proposed approach integrates these elements to create a robust foundation for building secure, scalable, and efficient APIs.

2. Literature Review and Industry State

API architectures have evolved significantly over time. While REST APIs, first described in Fielding's dissertation [10], created the basis for modern web services, they face new limitations as AI workloads grow. Leading tech companies have tackled these challenges through innovative solutions: Google developed gRPC to enhance streaming capabilities through Protocol Buffers. At the same time, Netflix created Falcor to optimize data fetching with its JSON Graph model.

2.1. Current Industry Solutions

2.1.1. Vector Operation Handling

Adopting HNSW (Hierarchical et al.) graphs [11] has transformed how database services handle vector operations.

These graphs deliver impressive performance metrics - they can search through millions of vectors in less than 10 milliseconds, representing a 90% improvement over conventional brute-force search methods. MongoDB's Atlas Vector Search [12] merges traditional database queries with vector search capabilities, enabling sophisticated applications such as recommendation systems

2.1.2. Event-Driven Architectures

Kafka's implementation demonstrates the capabilities of modern event architecture at scale [13]. Production deployments have shown that event-driven systems can successfully maintain data consistency while expanding horizontally to meet growing demands.

AWS EventBridge [14] takes this concept further, illustrating how serverless event routing can significantly reduce operational complexity in distributed systems.

2.1.3. AI-First Architecture

The emergence of large-scale AI platforms has revealed innovative approaches to managing substantial inference workloads [15]. Systems like TensorFlow [16] highlight a crucial architectural insight - organizations can achieve more efficient resource utilization and improved performance by separating AI-serving infrastructure from conventional API operations.

2.1.4. Security Implementations

The effectiveness of context-aware security over conventional methods has been demonstrated through Google's BeyondCorp implementation [17]. In a parallel development, Azure's API Management for ML [18] has established practical methods for safeguarding AI models while maintaining robust input validation protocols.

2.2. Emergency Trends

Three key trends stand out in current implementations:

2.2.1. Hybrid Architectures

Organizations are succeeding with hybrid systems that merge traditional and vector operations in unified API frameworks. These setups optimize resource usage across mixed workloads [19].

2.2.2. Automated Scaling

New autoscaling systems use ML to predict resource needs, offering more sophisticated solutions than basic threshold-based scaling [20].

2.2.3. Edge Processing

Edge computing shows promise for latency-sensitive operations. Cloudflare's work demonstrates how edge deployment can significantly reduce bandwidth needs for AI-enabled APIs [21].

3. AI-First Architecture Implementation

Building effective AI-driven systems requires fundamentally different architectural choices than traditional APIs. Let us examine the core components and patterns that enable high-performance ML operations.

3.1. Core Components

3.1.1. Feature Store Architecture

The feature store architecture incorporates two distinct components: an online store that enables rapid feature retrieval during inference operations and an offline store that maintains historical feature values essential for model training. These components are unified through a synchronization layer, which ensures data consistency between stores. Real-world production implementations have demonstrated that this architectural approach delivers substantial improvements in both feature computation performance and the overall effectiveness of model training processes.

3.1.2. Model Serving Pipeline

The model-serving pipeline is another crucial component of the AI-first architecture. This pipeline must handle the complex requirements of model deployment, versioning, and inference with high reliability and performance.

The pipeline follows a multi-stage approach:

1) Model Registration: New models are registered with metadata, version information, and runtime requirements.
2) Version Control: A sophisticated versioning/rollback system manages model versions and their dependencies.
3) Deployment Management: Automated deployment processes (blue-green approach) handle model distribution and resource allocation.
4) Serving Layer: A high-performance serving layer manages model inference with load balancing and monitoring.

This architecture (Figure 1) follows patterns like TensorFlow Serving's design principles for model serving and scalability.

3.2. Integration Patterns

The effectiveness of an AI-first architecture depends heavily on how well its components integrate with existing systems. Two critical integration patterns emerge as essential: vector operations and batch processing.

3.2.1. Vector Operation Integration

Vector operations demand specialized endpoints. Key implementations include:

- Dynamic index selection based on dimensions and query patterns
- Automatic timeout handling
- Memory management for large vectors

Fig. 1 Model serving pipeline architecture

These optimizations consistently achieve sub-100ms response times for million-scale vector datasets [23].

3.2.2. Batch Processing Integration

Batch processing capabilities are essential for handling high throughput scenarios efficiently. Sophisticated batch processing patterns balance throughput with resource utilization and system stability. The batch processor balances throughput with system stability through:

- Intelligent chunking of large batches
- Backpressure mechanisms
- Comprehensive result aggregation
- Timeout handling for extended operations

Organizations using this approach report up to 300% improvement in processing throughput while maintaining stability under heavy loads [24].

4. Event-Driven Architecture Patterns

4.1. Core Components

The event-driven architecture (Figure 2) is organized into three distinct layers: input, processing, and storage. Event sources in the input layer generate a continuous stream of data that requires immediate processing. These events flow through the processing layer, where specific business logic transforms and analyzes the incoming data. Finally, the storage layer systematically preserves the processed information for future use and reference.

4.1.1. Input Layer

- Event Sources generate continuous data streams
- Event Bus implements message routing and distribution mechanisms
- Publisher-subscriber patterns enable loose coupling between components

4.1.2. Processing Layer

- Event Processors handle core business logic
- ML Pipeline executes model inference and feature computation
- Vector Processor manages similarity searches and embedding operations

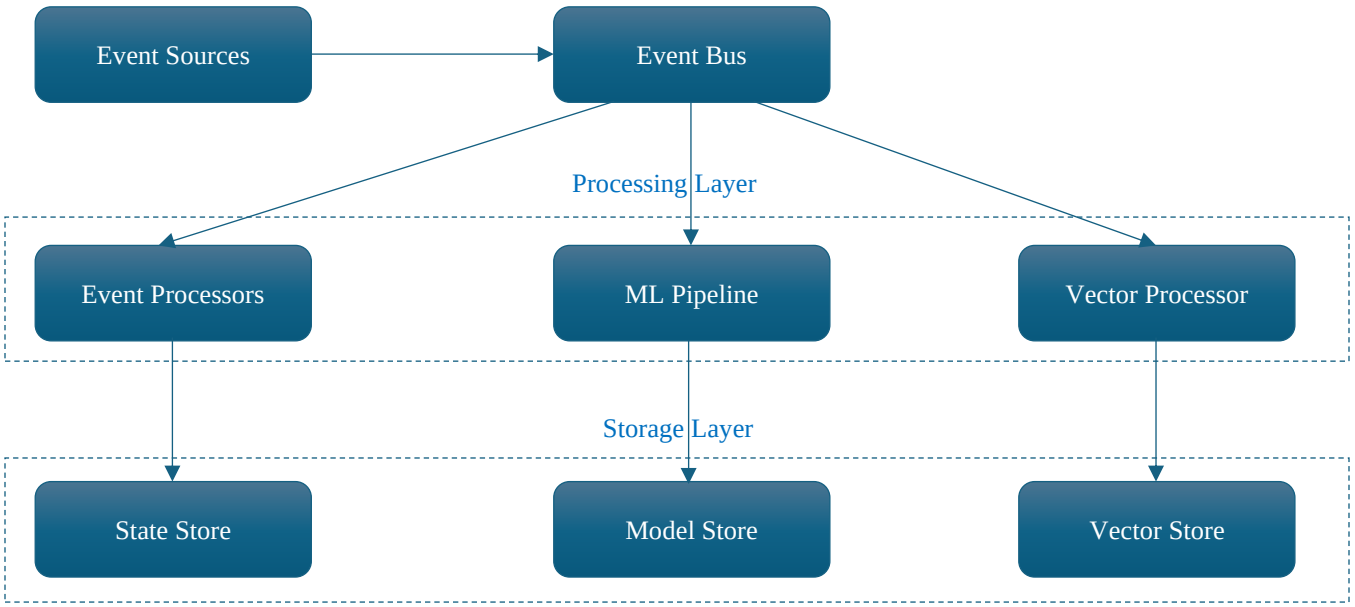

Fig. 2 Event-driven architecture components

4.1.3. Storage Layer

- State Store maintains system state through event logs
- Model Store manages ML artifacts and versioning
- Vector Store provides specialized storage for high-dimensional vectors

4.2. Event Processing Implementation

The Event Processor orchestrates comprehensive event lifecycles through multiple mechanisms. It begins with event enrichment and validation processes, followed by dynamic handler registration capabilities.

To ensure system resilience, it implements circuit breaker patterns for effective failure management [25]. The system's reliability is further enhanced through retry policies that utilize exponential back-off strategies for handling temporary disruptions.

4.3. State Management and Consistency

The implementation builds upon event sourcing as its core architectural principle. At its foundation lies an append-only event log that functions as the authoritative system of record. Materialized views provide streamlined access to current system states while snapshotting mechanisms optimize storage utilization and processing performance. The architecture maintains data consistency through clearly defined transaction boundaries throughout the system.

4.4. System Benefits

This architecture enables:

- Independent scaling of components
- Loose coupling for improved resilience
- Optimized handling of different data types
- Real-time processing capabilities

5. Zero-Trust Security Framework

5.1. Key Principles

The zero-trust security framework is built upon three fundamental principles: assume breach, least privilege access, and continuous authentication and authorization [26].

- Assume Breach: Every access attempt requires verification regardless of the source.
- Least Privilege Access: Access rights are minimized to required functionality.
- Continuous Authentication and Authorization: Ongoing verification of identity and context throughout sessions.

5.2. Framework Components

5.2.1 Identity and Access Management (IAM)

The IAM component manages authentication using multi-factor protocols and implements role-based access control with attribute-based policies [27]. Device attestation verifies hardware and software configurations before granting access.

5.2.2. Data Protection

This component implements AES-256 encryption for data at rest and TLS 1.3 for transit security [28]. Fine-grained permissions control data access while comprehensive auditing tracks usage patterns.

5.2.3. Threat Detection

Real-time monitoring uses both rule-based and ML-based detection methods. The system analyzes security telemetry from application logs, network traffic, and user behavior to identify potential threats [29].

5.3. ML-Specific Security Considerations

5.3.1. Model Security

Digital signatures verify model integrity, while input validation protects against adversarial attacks [30]. The framework includes detection mechanisms for identifying malicious input patterns.

5.3.2. Data Security

The framework implements data encryption and secure access patterns to prevent unauthorized inference [31]. Federated learning enables collaborative model training while preserving data privacy [32].

5.3.3. Model Governance

Risk assessment and bias detection tools ensure responsible model deployment [33]. Role-based workflows control model updates while continuous monitoring tracks model behavior.

5.4. Security Integration

The security framework is integral to the system, ensuring robust protection and compliance across various components. It integrates seamlessly with the API gateway, enabling authentication mechanisms that safeguard access.

Additionally, it works in conjunction with the event processing layer to perform telemetry analysis, identifying potential threats in real-time. The framework supports the storage layer to protect data through encryption and access controls. Moreover, the framework incorporates security measures within the machine learning (ML) pipeline to protect models against vulnerabilities such as adversarial attacks.

6. Proposed System Architecture

6.1. Architecture Overview

The architecture presented (Figure 3) addresses fundamental limitations in contemporary API design through three key principles: AI-first integration, advanced event processing, and contextual security validation. Unlike traditional approaches that treat machine learning capabilities as external services, this architecture incorporates vector operations and models as primary architectural components.

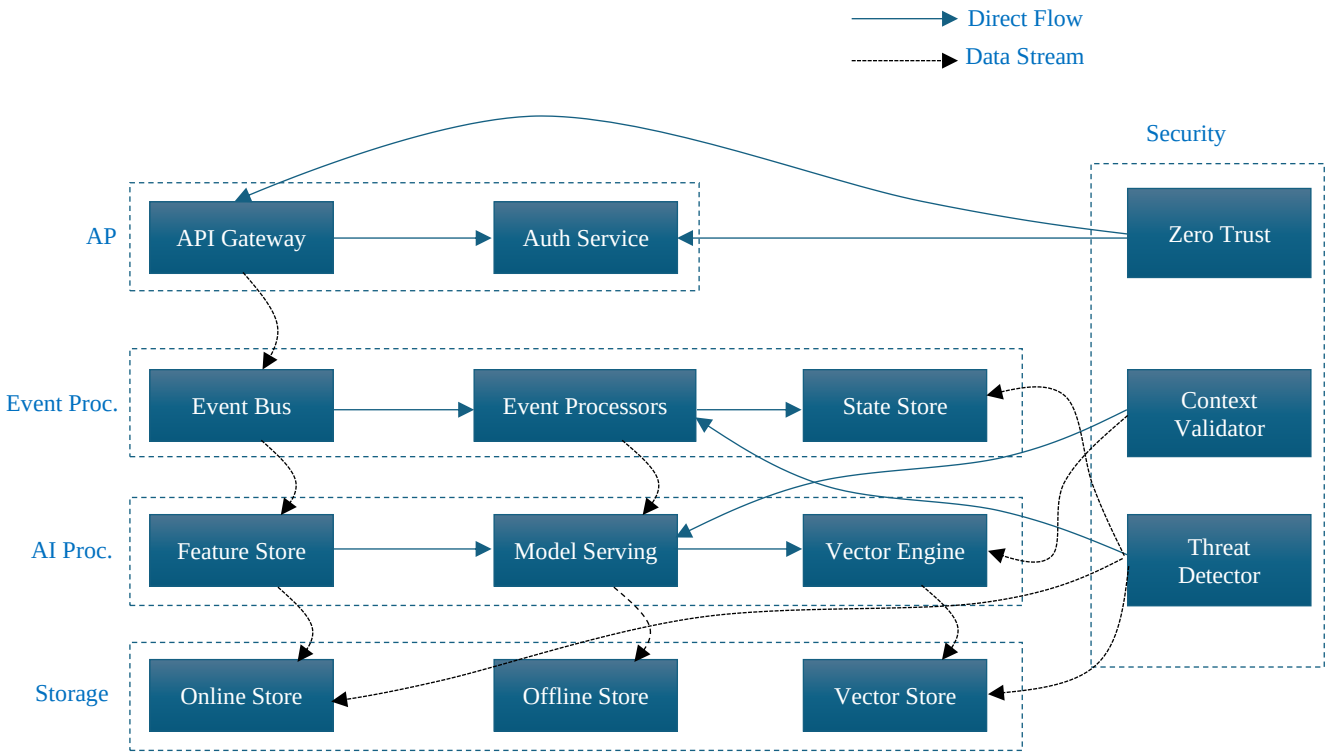

Fig. 3 Complete system architecture

6.1.1. AI-First Integration Framework

The architecture adopts an AI-first approach, seamlessly integrating with traditional data operations to enhance system functionality. It includes an integrated feature store that leverages dual storage capabilities to support both online and offline data access. Additionally, the architecture provides a unified Machine Learning (ML) model serving pipeline, ensuring efficient deployment and management of models. Moreover, it facilitates cohesive workflow management, enabling streamlined coordination between ML processes and standard API operations.

6.1.2. Advanced Event-Driven Processing Model

The event processing system transcends conventional publish/subscribe (pub/sub) patterns by incorporating advanced capabilities tailored for Machine Learning (ML) workflows. It enables real-time feature computation while ensuring rigorous validation to maintain data quality. The system also supports stateful processing, allowing it to effectively manage the complex requirements of ML workflows. Additionally, it implements adaptive resource allocation strategies to optimize performance and scalability under varying workloads.

6.1.3. Security Framework

The security implementation applies zero-trust principles, tailored explicitly for Machine Learning (ML) workloads, to ensure robust protection across the system. It incorporates model validation and integrity verification processes to safeguard models from tampering or corruption. Furthermore, context-aware access control mechanisms are employed to regulate permissions dynamically based on the specific requirements of each interaction.

The framework also includes systematic ML-specific threat detection capabilities, enabling proactive identification and mitigation of vulnerabilities unique to ML systems.

6.2. Implementation Framework

The architecture implements its AI-first approach through three distinct but interconnected layers: the API, Event Processing, and AI Processing Layer. Each layer is designed to support traditional API operations and ML workloads seamlessly.

6.2.1 API Layer

The API Layer is the primary interface facilitating external interactions within the system. It integrates routing mechanisms to support traditional and machine learning (ML) endpoints, ensuring seamless communication.

Additionally, the layer is designed for ML-aware request handling and validation, improving the system's reliability. It offers compatibility with multiple protocols, including REST, GraphQL, and gRPC, broadening its applicability across various platforms. Furthermore, the API Layer enables real-time feature validation and computation, enhancing the system's responsiveness and accuracy.

6.2.2 Event Processing Layer

The Event Processing Layer manages asynchronous operations within machine learning systems. It achieves this by implementing ML-specific event patterns and workflows, enabling the efficient processing of real-time feature computation pipelines. Additionally, it supports continuous monitoring and validation of models to ensure reliability and performance. Furthermore, the layer facilitates effective state management for various machine-learning operations.

6.2.3. AI Processing Layer

AI Processing Layer serves as the system's central intelligence and comprises three main components. First, the Feature Store utilizes a dual storage architecture to enable online and offline access, ensures feature consistency, and facilitates real-time feature computation. Second, the Model Serving component supports blue-green deployment strategies, manages model versioning and lifecycle processes, and enables A/B testing to compare model performance. Finally, the Vector Engine is designed for high-dimensional vector operations, optimizing similarity search tasks and ensuring efficient index management.

6.2.4. Storage Layer

The storage implementation leverages specialized data stores, each designed to accommodate specific data types and usage scenarios. The online store is optimized for managing current operational data, ensuring low-latency access for real-time applications. In contrast, the offline store is dedicated to handling historical and training data, supporting analytical workloads, and machine learning (ML) model development. A vector store is also employed to efficiently store high-dimensional vectors, which are essential for similarity search and other ML-driven operations.

6.2.5. Security Layer

The security implementation adheres to zero-trust principles, ensuring a comprehensive and proactive approach to safeguarding Machine Learning (ML) workflows. It emphasizes the continuous validation of ML operations to maintain system reliability and prevent unauthorized activities.

Context-aware access control mechanisms dynamically regulate permissions based on the specific context of each interaction. The implementation also includes ML-specific threat detection to identify and mitigate vulnerabilities unique to ML environments. Additionally, model integrity verification is employed to protect against tampering and ensure the trustworthiness of deployed models.

7. Analysis

The proposed architecture demonstrates several significant advantages in handling modern API requirements, particularly in four key areas: flexibility, extensibility, scalability, and security.

7.1. Operational Benefits

The proposed architecture delivers key operational advantages through its integrated approach to AI operations. The system's flexibility enables adaptation to diverse ML workloads through dynamic resource allocation and support for synchronous and asynchronous operations. This flexibility extends beyond essential resource management, including intelligent workload distribution and adaptive processing patterns that accommodate varying computational demands. Extensibility is achieved through the architecture's modular design, which enables incremental addition of capabilities without disrupting existing operations. The standardized interfaces between components facilitate the integration of new technologies as they emerge while allowing the independent evolution of individual components. This approach ensures the architecture can adapt to advancing ML technologies and changing operational requirements. Security integration represents a fundamental advancement over traditional approaches. The architecture ensures comprehensive protection of data and models by implementing fine-grained access control at the operation level and maintaining continuous validation of ML workflows. The security framework adapts zero-trust principles specifically for ML operations, providing threat detection mechanisms tailored to AI workloads. The architecture's scalability characteristics are implemented across multiple dimensions. Horizontal scaling capabilities enable independent scaling of ML processing units and distributed feature computation. Resource management is handled through dynamic allocation mechanisms that respond to ML workload demands, with automated scaling triggers and intelligent load balancing across processing units. Data distribution is managed through sharded vector storage, distributed feature stores, and replicated model-serving, ensuring efficient data access patterns at scale.

7.2. Comparative Analysis

The architecture provides a theoretical framework for future implementation and empirical evaluation.

Table 1. Comparative analysis

Feature	Traditional Solutions	Proposed Architecture
AI Operations	Separate ML services	Integrated processing
Event Processing	Basic event handling	ML-aware events
Security	Generic API security	ML-specific security
Scalability	Component level scaling	Unified scaling across layers
Data Distribution	Single store focused	Multi-store optimization
Resource Management	Static allocation	Dynamic ML aware allocation

8. Conclusion

This research presents an architectural framework that addresses the growing complexity of integrating AI operations into modern API systems. The proposed architecture advances the field through three key innovations: an AI-first integration approach that treats ML operations as primary architectural components, an event-driven processing framework optimized for ML workflows, and a comprehensive security model adapted for AI workloads. The architecture's primary contribution lies in its unified approach to handling traditional API operations and ML workflows within a single cohesive framework. This integration eliminates the complexity and overhead typically associated with maintaining separate systems for ML operations and standard API functionality. The event-driven processing framework enables sophisticated ML workflows while maintaining system flexibility and extensibility. The security framework adapts zero-trust principles specifically for ML operations, providing comprehensive protection without compromising system performance. The architecture provides a foundation for building scalable, secure, and maintainable AI-enabled systems through its layered approach, encompassing API, Event Processing, AI Processing, Storage, and Security layers.

The multi-store data architecture and dynamic resource allocation mechanisms enable efficient handling of diverse workloads, while the modular design facilitates system evolution as ML technologies advance. Future research directions include empirical evaluation of the architecture's performance characteristics, investigation of advanced scaling mechanisms for specific ML workloads, and development of additional security measures for emerging ML threats. The framework presented here is a theoretical foundation for implementing robust, AI-enabled API systems that can evolve with advancing technology while maintaining operational efficiency and security.

References

[1] Andrei Paleyes, Raoul-Gabriel Urma, and Neil D. Lawrence, "Challenges in Deploying Machine Learning: A Survey of Case Studies," *ACM Computing Surveys*, vol. 55, no. 6, pp. 1-29, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[2] J. Johnson, M. Douze and H. Jégou, "Billion-Scale Similarity Search with GPUs," in *IEEE Transactions on Big Data*, vol. 7, no. 3, pp. 535-547, 1 July 2021, doi: 10.1109/TBDATA.2019.2921572.

[3] Kim Hazelwood et al., "Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective," *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Vienna, Austria, pp. 620-629, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[4] Daniel Crankshaw et al., "Clipper: A Low-Latency Online Prediction Serving System," *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017. [Google Scholar] [Publisher Link]

[5] Chaoyun Zhang, Paul Patras, and Hamed Haddadi, "Deep Learning in Mobile and Wireless Networking: A Survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2224-2287, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[6] Samyam Rajbhandari et al., "ZeRO: Memory Optimizations Toward Training Trillion Parameter Models," *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, Atlanta, GA, USA, pp. 1-16, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[7] Paris Carbone et al., "State Management in Apache Flink®: Consistent Stateful Distributed Stream Processing," *Proceedings of the VLDB Endowment*, vol. 10, no. 12, pp. 1718–1729, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[8] Shivaram Venkataraman et al., "Drizzle: Fast and Adaptable Stream Processing at Scale," *SOSP '17: Proceedings of the 26th Symposium on Operating Systems Principles*, Shanghai China, pp. 374–389, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[9] Chen Li et al., "The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-scale, Unbounded, Out-of-order Data Processing," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1792–1803, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[10] Roy Thomas Fielding, "*Architectural Styles and the Design of Network-based Software Architectures*," Ph.D. dissertation, Department of Information Computer Science, University of California, Irvine, CA, USA, 2000. [Google Scholar] [Publisher Link]

[11] Yu A. Malkov, and D.A. Yashunin, "Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 4, pp. 824-836, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[12] MongoDB, MongoDB Atlas Vector Search. [Online]. Available: https://www.mongodb.com/products/platform/atlas-vector-search

[13] Kafka, Documentation. [Online]. Available: https://kafka.apache.org/documentation/

[14] Amazon Web Services, Amazon EventBridge Developer Guide. [Online]. Available: https://aws.amazon.com/eventbridge/

[15] OpenAI, OpenAI API Documentation, 2020. [Online]. Available: https://openai.com/blog/openai-api/

[16] Google LLC, "TensorFlow Serving Documentation," 2021. [Online]. Available: https://www.tensorflow.org/tfx/guide/serving

[17] Google Cloud, BeyondCorp: A New Approach to Enterprise Security. [Online]. Available: https://cloud.google.com/beyondcorp

[18] Microsoft Corporation, Azure API Management Documentation, Microsoft Technical Documentation. [Online]. Available: https://azure.microsoft.com/en-us/services/api-management/

[19] Kabir Nagrecha, and Arun Kumar, "Hydra: A System for Large Multi-Model Deep Learning," *arXiv*, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[20] Alexey Ilyushkin et al., "An Experimental Performance Evaluation of Autoscaling Policies for Complex Workflows," *ICPE '17: Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*, L'Aquila Italy, pp. 75-86, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[21] Cloudflare Workers Documentation. [Online]. Available: https://workers.cloudflare.com/

[22] J. Hermann, and M. Del Balso, Meet Michelangelo: Uber's Machine Learning Platform, Uber Engineering Blog, 2017. [Online]. Available: https://eng.uber.com/michelangelo

[23] Herve Jégou, Matthijs Douze, and Cordelia Schmid, "Product Quantization for Nearest Neighbor Search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 1, pp. 117-128, 2011. [CrossRef] [Google Scholar] [Publisher Link]

[24] Matei Zaharia et al., "Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing," *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI'12)*, 2012. [Google Scholar] [Publisher Link]

[25] Fabrizio Montesi, and Janine Weber, "Circuit Breakers, Discovery, and API Gateways in Microservices," *arXiv*, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[26] Scott Rose et al., "Zero Trust Architecture," *NIST Special Publication 800-207, National Institute of Standards and Technology*, 2020. [CrossRef] [Publisher Link]

[27] Dick ardt, "The OAuth 2.0 Authorization Framework," *Internet Engineering Task Force (IETF)*, 2012. [Google Scholar]

[28] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," *Internet Engineering Task Force (IETF)*, 2018. [Google Scholar] [Publisher Link]

[29] Robin Sommer, and Vern Paxson, "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection," *2010 IEEE Symposium on Security and Privacy*, Oakland, CA, USA, pp. 305-316, 2010. [CrossRef] [Google Scholar] [Publisher Link]

[30] Maria-Irina Nicolae et al., "Adversarial Robustness Toolbox v1.0.0," *arXiv*, pp. 1-34, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[31] Haokun Fang, and Quan Qian, "Privacy Preserving Machine Learning with Homomorphic Encryption and Federated Learning," *Future Internet*, vol. 13, no. 4, pp. 1-20, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[32] Qiang Yang et al., "Federated Machine Learning: Concept and Applications," *ACM Transactions on Intelligent Systems and Technology*, vol. 10, no. 2, pp. 1-19, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[33] Solon Barocas, Moritz Hardt, and Arvind Narayanan, *Fairness and Machine Learning*, 2017. [Publisher Link]