Towards Message Brokers for Generative AI: Survey, Challenges, and Opportunities

ALAA SALEH, Center for Ubiquitous Computing, Oulu, Finland
ROBERTO MORABITO, Department of Communication Systems, Eurecom, Biot, France
SCHAHRAM DUSTDAR, Distributed Systems Group, TU Wien, Vienna, Austria
SASU TARKOMA, Computer Science and Engineering, Helsinki, Finland
SUSANNA PIRTTIKANGAS, Center for Ubiquitous Computing, Oulu, Finland
LAURI LOVÉN[*], Center for Ubiquitous Computing, Oulu, Finland

In today's digital world, GenAI is becoming increasingly prevalent by enabling unparalleled content generation capabilities for a wide range of advanced applications. This surge in adoption has sparked a significant increase in demand for data-centric GenAI models spanning the distributed edge-cloud continuum, placing increasing demands on communication infrastructures, highlighting the necessity for robust communication solutions. Central to this need are message brokers, which serve as essential channels for data transfer within various system components. This survey aims to delve into a comprehensive analysis of traditional and modern message brokers based on a variety of criteria, highlighting their critical role in enabling efficient data exchange in distributed AI systems. Furthermore, we explore the intrinsic constraints that the design and operation of each message broker might impose, highlighting their impact on real-world applicability. Finally, this study explores the enhancement of message broker mechanisms tailored to GenAI environments. It considers key factors such as scalability, semantic communication, and distributed inference that can guide future innovations and infrastructure advancements in the realm of GenAI data communication.

CCS Concepts: • **Computing methodologies** → **Semantic networks**; **Self-organization**; **Multi-agent systems**; **Intelligent agents**; **Cooperation and coordination**; *Information extraction*; • **Networks** → **Application layer protocols**.

Additional Key Words and Phrases: Generative AI, Message Brokers, Publish/Subscribe Paradigm, Brokerless, Edge Computing, Large Language Models.

1 Introduction

In the burgeoning field of Generative Artificial Intelligence (GenAI), the computing continuum faces unprecedented challenges in efficiently managing data flows and computational resources. GenAI has predominantly targeted consumer applications, offering applications such as the ChatGPT [1], a conversational AI agent based on a large language model (LLMs), a type of machine learning (ML) model with a deep neural network. However, a surge in machine-to-machine (M2M) use cases, coupled with the increasing possibility of relying more and more on decentralized, distributed, and edge-based Large Language Models (LLMs), beckons a reevaluation of

[*]Corresponding Author

Authors' Contact Information: Alaa Saleh, Center for Ubiquitous Computing, Oulu, Finland; e-mail: alaa.saleh@oulu.fi; Roberto Morabito, Department of Communication Systems, Eurecom, Biot, France; e-mail: roberto.morabito@eurecom.fr; Schahram Dustdar, Distributed Systems Group, TU Wien, Vienna, Austria; e-mail: dustdar@dsg.tuwien.ac.at; Sasu Tarkoma, Computer Science and Engineering, Helsinki, Uusimaa, Finland; e-mail: sasu.tarkoma@helsinki.fi; Susanna Pirttikangas, Center for Ubiquitous Computing, Oulu, Pohjois-Pohjanmaa, Finland; e-mail: susanna.pirttikangas@oulu.fi; Lauri Lovén, Center for Ubiquitous Computing, Oulu, Pohjois-Pohjanmaa, Finland; e-mail: lauri.loven@oulu.fi.

ACM 1557-7341/2025/6-ART
https://doi.org/10.1145/3742891

the supporting communication infrastructure. This reevaluation is necessitated by the evolving demands for higher bandwidth, lower latency, and more robust data processing capabilities that these advanced applications require [2, 3]. This survey paper delves into the role of publish/subscribe (pub/sub) message broker systems, considering in particular their emerging role for seamless and scalable data exchange in GenAI applications. We scrutinize contemporary message brokers for their adaptability and efficiency in GenAI contexts, outline existing challenges, and chart promising research avenues for future development.

In more detail, while the focus of GenAI systems has been on consumer-oriented applications, there is an increased interest towards M2M use cases. GenAI has been proposed to be used in, for example, for networking, wireless communication, and compression [4].

As a result, current computing continuum platforms, spanning the networks and computational resources from user devices to cloud [5, 6], face new challenges. These platforms provide support AI models, offering interconnect between their data sources and sinks, and optimising their use of resources in the computing continuum. As GenAI models require and generate ever larger amounts of data [7], all the while consuming computational resources varying from moderate to massive [5], the computing continuum must offer a dynamic and scalable communication and computation substrate to ensure timely data dissemination and efficient use of resources [8, 9].

Pub/sub approach is equally useful alongside other paradigms across the computing continuum for a wide range of AI applications, including smart cities [10], healthcare [11], and many other domains. This approach decouples data producers from their consumers, allowing applications to develop components independently, and enhancing system robustness and adaptability [12]. Furthermore, pub/sub makes system design more flexible by increasing the independence between system components with a reliable interconnect.

Pub/sub systems are based on the exchange of data between clients (e.g., services or application components) through a message broker. Publishers submit content to the broker, which then allows subscribers to access that content without knowing its source [13, 14]. By managing, filtering, and routing communication between publishers and subscribers, the message broker acts as an intermediary layer [15], routing and distributing messages efficiently, accurately, and in a timely manner, based on the interests expressed by subscribers. Within the brokers, message queues can temporarily store the messages, protecting the system from overflows or outages. Moreover, brokers also often provide other essential features such as persistent storage, monitoring, and authentication.

As GenAI continues to evolve, parallels can be drawn with the historical trajectory of IoT systems, where the emergence of robust and adaptive message brokers marked a significant evolutionary step. These message brokers became a de facto paradigm, primarily because they addressed critical challenges associated with scalability, real-time data processing, and the integration of heterogeneous devices and platforms [16]. Similarly, in the context of GenAI, the anticipation of an analogous development is not without merit. The complexity and volume of data that GenAI applications demand, coupled with the necessity for high-quality service monitoring and data exchange processes, suggest that a transition towards more sophisticated message brokering solutions may be inevitable. Such solutions would not only have to manage the large data throughput but also ensure adaptability, reliability, and efficiency in dynamic GenAI ecosystems. Reflecting on the IoT evolution, the motivations for this shift include the need to support scalable communication, facilitate interoperability among diverse systems, and uphold stringent Quality of Service (QoS) standards, which are likely to be paralleled in the GenAI domain [17].

However, it is critical to acknowledge that the evolution towards more sophisticated message brokering solutions, specifically tailored to accommodate GenAI application needs, does not come without its challenges.

Bearing all this in mind, the primary contributions of this survey are summarized as follows:

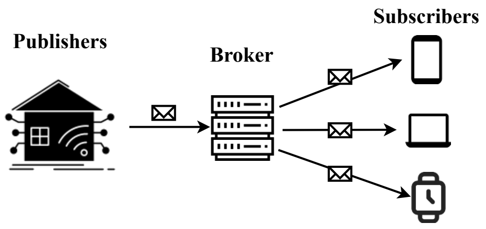

Fig. 1. The Publish/Subscribe paradigm.

- We provide a comprehensive review of recent message brokers, evaluating the brokers by their suitability for GenAI systems. We aim to guide the development of a compatible brokering framework in consideration of the evolving requirements for GenAI systems in the future.
- We summarize the challenges of message brokers and highlight the need for a robust and efficient data communications substrate based on an increase in GenAI applications.
- We discuss central research topics and their potential focus areas for making message brokers suitable for GenAI applications. We also describe promising algorithms for implementing such brokers.

This paper's remaining sections are organized as follows. Section 2 provides a general definition of pub/sub paradigm and highlights the advantages and disadvantages of both broker-based and brokerless messaging architectures. Section 3 presents existing message brokers with their features and cons. Section 4 examines possible ways to make message brokers suitable for GenAI applications. The paper concludes with Section 5.

2 The Pub/Sub Paradigm

Pub/sub is a messaging paradigm where publishers send messages without indicating specific recipients. Remaining oblivious to the original publishers, subscribers receive relevant messages according to their interests. At its core, pub/sub thus decouples message delivery from the senders and recipients. This enhances the system's adaptability and robustness, as it allows publishers and subscribers to operate independently [12]. Furthermore, subscribers can flexibly choose topics based on their interests, enabling them to find content relevant to their preferences. As a result of pub/sub, real-time messaging can be sent to a wide range of subscribers, enabling scalable and timely dissemination of information [14, 18, 19].

As part of the pub/sub communication model, a message broker functions as an intermediary layer, managing the flow of messages from publishers to subscribers. By ensuring that messages are accurately routed to subscribers, based on their expressed interests or specific topics, the broker ensures that messages are received by subscribers precisely as they have been requested. By providing a layer of abstraction between publishers and subscribers, the broker goes beyond simply facilitating message transmission. Therefore, neither party needs to be aware of the other's operations or presence. A key strength of the broker is its reliability [20], as it has mechanisms for guaranteeing delivery of messages even when subscribers are temporarily offline or have connectivity difficulties.

Moreover, message queues are general-purpose components of the broker that temporarily store messages from publishers until they can be delivered to subscribers as part of the broker process [21]. By orchestrating the overall flow of messages, the message broker ensures that the appropriate subscribers receive the messages based on their subscriptions, while a message queue ensures that these messages are held and dispatched in an orderly manner, ensuring that publishers and subscribers are able to communicate in an asynchronously and decoupled ways as shown in Fig. 1.

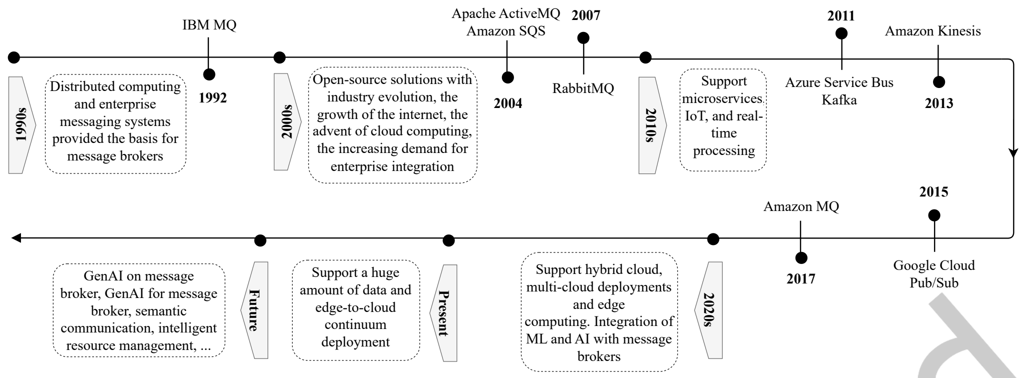
1990s
Distributed computing and enterprise messaging systems provided the basis for message brokers
IBM MQ
1992
2000s
Open-source solutions with industry evolution, the growth of the internet, the advent of cloud computing, the increasing demand for enterprise integration
Apache ActiveMQ
Amazon SQS
2004
2007
RabbitMQ
2010s
Support microservices, IoT, and real-time processing
2011
Azure Service Bus
Kafka
Amazon Kinesis
2013
2015
Google Cloud Pub/Sub
Amazon MQ
2017
2020s
Support hybrid cloud, multi-cloud deployments and edge computing. Integration of ML and AI with message brokers
Present
Support a huge amount of data and edge-to-cloud continuum deployment
Future
GenAI on message broker, GenAI for message broker, semantic communication, intelligent resource management, ...

Fig. 2. The timeline of message broker evolution from 1990 to present.

2.1 Message broker development

Over the past 30 years, message broker technology has evolved and innovated significantly as shown in Fig. 2. With the development of message-oriented middleware (MOM) and the Java Message Service (JMS), the technology began to gain prominence between 1980 and 1999, as organizations began to require greater integration and communication [22, 23]. During the period 2000-2009, the technology underwent significant advancements, influenced by the implementation of service-oriented architecture (SOA), the increased internet usage, and the emergence of cloud computing [24]. Among these developments was the implementation of web services standards and open-source alternatives, laying the foundation for future advances in cloud-based message brokers. During the following decade, from 2010 to 2019, the technology adapted to new demands such as real-time data processing, cloud computing, IoT, and microservices architectures [21]. Increasing demands for Internet of Things and real-time data processing have led to the rise of containerization platforms such as Docker and Kubernetes.

With the advent of cloud-native architectures, microservices, edge computing, and IoT demands, message broker technology continued to evolve unabated between 2020 and 2023 [25]. The integration of AI and ML was particularly important for optimizing message routing, anomaly detection, and auto-scaling, addressing the complexities of growing data volumes. In the future, message broker technology will continue to evolve, leveraging advances in computing and communication. As 5G/6G technologies advance, cross-platform interoperability, and decentralized architectures are developed, a number of trends are set to shape its future. These include edge computing, quantum computing, enhanced security, serverless architectures, and advancements in the 5G/6G technologies.

2.2 Broker vs. Brokerless Messaging Architecture

A broker is the main unit for managing and monitoring data of pub/sub systems [13], offering scalability, balanced load distribution, and optimal resource utilization, among others. Additionally, the message broker ensures that messages are reliably transmitted, preventing any loss of data.

Despite their benefits, certain challenges accompany message brokers, particularly concerning scalability and efficiency. As data volume grows, brokers face increased complexity, undermining scalability – a common issue also observed in brokerless pub/sub systems known for their simplicity, quick access, and improved efficiency. In such systems, publishers and subscribers interact directly, making discovery, management, and availability crucial factors. However, the absence of a central unit for overseeing message flows complicates supervision and control. Furthermore, this setup does not inherently guarantee reliable message delivery [26].

3 Survey of Message Brokers

In the past decade, numerous message brokers have been developed both in proprietary and open-source sectors. Each broker possesses unique features and pitfalls, influenced by their respective vendors and intended applications. This section focuses on the most commonly used message brokers, with their features and challenges. To clarify the methodology used for the comparative analysis, we utilized data sourced from the official websites of the brokers under study. This data included technical documentation, feature descriptions, code explanations, and vendor-provided whitepapers. The comparative analysis framework is structured in Tables 1 to 4. We categorize these message brokers based on their open-source availability and the priority-based delivery of messages (built-in priority-support) that ensures messages are delivered in priority order, with high-priority messages processed first.

3.1 Open Source Message Brokers

We found 30 message brokers that were available as open source. Out of these, 17 supported priority messages, while 13 did not. Each is discussed in more detail in below subsections.

3.1.1 Priority Support. *Apache ActiveMQ* [27] is a Java-based message broker licensed under the Apache 2.0 license. Through the use of double layers of SSL/TLS security layers, it provides dual security levels. A distribution of Apache ActiveMQ provided by FuseSource, *Fuse Message Broker* [28] supports J2EE integration capabilities such as Java Database Connectivity (JDBC), J2EE Connector Architecture (JCA), and Enterprise JavaBeans (EJB). *Apache Qpid* [29], on the other hand, provides cloud-based messaging capabilities and supports queuing for structured message exchange, making it essential for distributed applications.

RabbitMQ [30] was developed by Rabbit Technologies Ltd in 2006 using the Erlang programming language and released under the Mozilla Public License. It supports multiple protocols, including AMQP, STOMP, and MQTT. *HornetQ* [31], a Java application based on JBoss, provides a distributed messaging platform for enterprise-level applications using STOMP and AMQP protocols.

Red Hat AMQ [32] is a messaging protocol based on Java for large-scale Internet business applications with no administrative costs, installation, or configuration required. *Celery* [33] is written in Python and supports multiple message brokers, including RabbitMQ and Redis. Using Java Messaging Service (JMS) API, *JBoss Messaging* [34] is a messaging broker provided by JBoss, a division of Red Hat for facilitating communication between different components or applications in a distributed system.

OpenMQ [35] is implemented in Java and was developed by Oracle as an open source protocol. *Beanstalk* [36] creates queues automatically with pure Python. *Gearman* [37] is an optimized server written in C/C++ with a simple interface that provides low application overhead. *Enduro/X* [38] is written in C and offers native APIs for C/C++. For enhanced interprocess communication, it utilizes in-memory POSIX kernel queues. As part of the WSO2 Integration platform, *WSO2 Message Broker* [39] is a message-based communication component.

HiveMQ [40] is compatible with MQTTv3.1 and all subsequent versions. The Eclipse Public License (EPL) and Eclipse Distribution License (EDL) cover this implementation. *Redis* [41] is BSD-licensed, used by companies such as Uber, Instagram, and AirBNB for caching and messaging queues. With 100 million concurrent connections per cluster and sub-millisecond latency, *EMQX* [42] can efficiently and reliably connect massive amounts of IoT devices. EMQX nodes can be bridged by other MQTT servers and cloud services to send messages across platforms. Additionally, it deploys and operates on all public cloud platforms. *Apache Pulsar* [43] is an open-source distributed messaging system developed as a queuing system, but it recently added event streaming features. It combines many Kafka and RabbitMQ features.

3.1.2 No Priority Support. *Apache Kafka* [44] was developed by LinkedIn as a distributed streaming platform, supporting multiple data formats, including JSON, Avro, and XML. Furthermore, Java, Python, and Go are the official client libraries and several cloud platforms are supported, including Amazon Web Services, Microsoft Azure, and Google Cloud Platform. It provides a variety of tools for managing and monitoring Kafka clusters, such as Kafka Manager, Kafka Monitor, and Kafka Connect.

Apache RocketMQ [45] is a cloud-native platform that operates across distributed systems, facilitating real-time data processing. With support for versions 5.0, 3.1.1, and 3.1, *Eclipse Mosquitto* [46] implements the MQTT protocol. *ZeroMQ* [47] is supported by a large and active open source community, and utilizes a broker-less pub/sub pattern.

Apache NiFi [48], developed by the Apache Software Foundation, automates data exchange between software systems, and facilitates the conversion of data formats in real time. *Ably Realtime* [49] is built on Ably's Data Stream Network, which includes a cloud network and realtime messaging fabric. Additionally, over 40 Client Library SDKs are available, as well as native support for six real-time protocols. *Apache SamZa* [50] is a streaming framework based on Apache Kafka and Apache Hadoop developed by LinkedIn and now part of the Apache Software Foundation. It processes real-time data streams generated by Apache Kafka, Amazon Kinesis, and Azure Event Hub.

VerneMQ [51] was launched in 2014 by Erlio GmbH. It supports MQTT messages in LevelDB, and uses a clustering architecture based on Plumtree, however it isn't actively developed and lacks features. *NServiceBus* [52] is designed with simplicity. With a number of retry strategies, a message which fails processing can automatically be forwarded to an error queue for manual investigation. *Kestrel* [53] is a JVM-based distributed message queue inspired by Blaine Cook's "Starling".

In *NSQ* [54], distributed and decentralized topologies are promoted, allowing fault tolerance and high availability as well as reliable delivery by replicating every message across multiple nodes within the cluster. *NATS* [55] was originally released in 2011 and was written in Go. *KubeMQ* [56] is a modern and innovative message queue and broker that facilitates communication across cloud platforms, on-premise environments, and edge deployments.

3.2 Proprietary Message Brokers

We found 16 proprietary message brokers, out of which 10 supported priority messages while 6 did not. Each is discussed in more detail in below subsections.

3.2.1 Priority Support. *IBM MQ* [57] supports data exchange between applications, systems, services, and files via messaging queues, serving as a crucial communication layer for message flow management. It offers flexibility in deployment options, whether in virtual machines or containers, including Docker, Kubernetes/Cri-O and Red Hat OpenShift. Moreover, it is ideal for applications demanding high reliability and zero message loss. *Amazon Simple Queue Service* [58] is operated by Amazon, so it can handle a lot of traffic with providing authentication using the Amazon API key and secret. However, requests are sent to the SQS web service via HTTP, which is susceptible to latency issues.

Microsoft Message Queue [59] is a messaging infrastructure created by Microsoft and built into the Windows Operating System. It serves as a queue manager and allows two or more applications to communicate without immediately knowing each other's responses.

As a Java-based message broker, *Oracle GlassFish Server Message Queue* [60] provides message brokering services to popular message queue systems such as AQ, IBM MQ Series, and TIBCO Rendezvous. It provides a consistent, open, JMS-compliant API for these message queuing systems. Additionally, OMB supports both durable and non-durable subscribers, as well as the JMS standard pub/sub, topic-based routing.

TIBCO Rendezvous [61] is a peer-to-peer architecture for high-speed data distribution. *TIBCO Enterprise Message Service* [62] is a message oriented middleware that supports a wide range of message protocols and technologies, including the Java Message Service (JMS) standard using Java and J2EE, Microsoft .NET, TIBCO FTL, TIBCO Rendezvous and C and COBOL on the Mainframe. Besides supporting up to 10 MB payloads in XML, JSON, CSV, HTML and plain text formats, *Anypoint MQ* [63] also has easy connectivity to Mule applications or non-Mule applications.

Azure Service Bus [64] from Microsoft is a cloud-based message broker that only supports AMQP and STOMP protocols. As a fundamental part of *SAP NW PI* [65] architecture, an Integration Broker facilitates communication between different enterprise applications, both SAP-based and non-SAP.

Solace Message Broker [66], also known as Solace PubSub+, is an advanced event broker that facilitates the efficient exchange of information between applications, IoT devices, and users through several messaging paradigms, including pub/sub, queue, request/reply, and streaming.

3.2.2 No Priority Support. *Google Cloud Pub/Sub* [67] is a messaging service offered by Google Cloud. In addition to native integration with other Google Cloud services, including Cloud Functions, Dataflow, and BigQuer, as well as a variety of development tools like Cloud Shell, Cloud Code, and Cloud Build, it supports real-time data processing for ML applications with Google Cloud AI Platform and other ML services. Aside from the Stackdriver Logging and Stackdriver Monitoring tools, it also provides SDKs for Java, Python, Node.js, and Go.

Amazon MQ [68] developed for ActiveMQ based on Java with support for MQTT, AMQP, STOMP, and WebSocket. *Intel MPI Library* [69] provides a cloud support for Amazon Web Services, Microsoft Azure, and Google Cloud Platform. *Amazon Kinesis* [70] is a real-time streaming data service with a scalable and durable architecture that can capture and store GBs or TBs of data per second from multiple sources for up to 24 hours. It provides various developer tools and integrations with AWS services, such as SDKs, templates, and integrations with AWS CloudFormation.

Azure Storage Queue [71] provides cloud messaging that enhances communication in the cloud, on desktops, on-premises, and on mobile devices. *IronMQ* [72] runs on public clouds as well as on-premise with providing client libraries in a wide variety of programming languages, including Python, Ruby, Java, PHP, and NET.

3.3 Summary on Message Brokers

Message brokers play a major role in streamlining communication between distributed systems by ensuring messages are properly routed. As highlighted in Tables 1 to 4, key strengths of message brokers include reliability, achieved through guaranteed message delivery that ensures no data loss during transmission, and flexibility, provided by pub/sub mechanisms that decouple publishers and subscribers. Additionally, they often provide mechanisms for message persistence so that they do not lose a single message in the event of a system failure. Their support for multiple messaging patterns, many different protocols, programming languages, and data styles, meets the needs of various types of communication. Moreover, many brokers come with an array of features. These features are critical elements, each contributing significantly to creating a robust message broker capable of managing communications in complex systems, ensuring efficiency. Further details of these features are provided below.

- *Clustering support* [27] enables scaling the message service to accommodate more clients or connections, effectively handling large message volumes and numerous clients.
- *Monitoring* [128] tools are crucial for tracking a message broker's performance and health, allowing for proactive management, early problem detection, and reliable operation.
- *Pub/sub support* [103] enables separation of publishers from subscribers, increasing system flexibility.
- *Parallel processing support* [44] allows the message broker to handle multiple messages simultaneously, improving throughput and efficiency.
- *Pull and push support* [44] allow flexible and timely message delivery.
- *Reliable delivery support* [129] ensures messages are not lost during transit, typically through acknowledgments, retries, or temporary storage until successful delivery.
- *Persistence support* [130] safeguards messages against loss during broker restarts or storage message failures.

Table 1. A summary of open source and priority-supporting message brokers.

Message Brokers and Queues	Clustering Support	Monitoring Support	Pub/Sub Support	Parallel Processing	Pull & Push Support	Reliable Delivery	Persistent	Authentication	Scalable	Distributed	Fault Tolerance	Shortcomings	Features
Apache ActiveMQ [27]	✓	✓	✓	✓	Both	✓	✓	✓	✓	✓	✓	Message delivery guarantees are limited [27]. Memory per queue is limited, the default number of messages is 400 [27]. Installation is complex [27]. Scaling is challenging [27].	Multi-protocols & multi-languages support [27]. Efficient management and resource allocation [27]. Supports flow control and message expiration [27]. Provides message groups as well as virtual and combined queues [27]. Works on small and medium-scale applications [27].
Fuse Message Broker [28]	✓	✓	✓	✓	Both	✓	✓	✓	✓	✓	✓	Limited monitoring tools[73].	Written in Java [73]. Supports JMS 1.1 & J2EE 1.4 integration-related components [73]. Supports loosely couple applications [73]. Supports multi-languages including C/C++, Java, .NET, Ruby, Perl, PHP, Pike,& Python [73]. Supports message compression [73].
Apache Qpid [29]	✓	✓	✓	✓	Both	✓	✓	✓	✓	✓	✓	Compatibility issues between versions [29]. Message size is limited to 100MB for AMQP protocols 0-8, 0-9, or 0-91 [29]	Implements AMQP Protocol [29]. Easy to use [74]. Detects failures and assigns messages to different brokers [29]. Low latency [29]. Supports multiple authentication schemes [29]. Active connections can be limited to protect client processes from malicious activity [29].
RabbitMQ [30]	✓	✓	✓	✓	Push	✓	✓	✓	✓	✓	✓	Written in Erlang, which is unfamiliar to many developers [30]. Queues with large numbers of messages are memory-intensive and strain brokers [30]. Redundant message broker communication [14]. Clustering has few features and is complicated [30]. Message size is limited to 512MB [14, 30].	Runs on all major operating systems [30]. Has good documentation [30]. Works with C, C++, .NET, and Python [30]. Supports asynchronous cluster-to-cluster message routing [75]. Supports multiple messaging protocols [30]. Offers several built-in exchange types [30]. Supports flow control for balancing workloads and avoiding rapid messages flooding [30].
HornetQ [31]	✓	✓	✓	✓	Both	✓	✓	✓	✓	✓	✓	Data loss may occur [14]. Delay may occur with large messages (up to 100KB) due to split message into multiple packages [76].	Supports AMQP and STOMP protocols [77]. Provides better performance and stability when combined with ActiveMQ [14].
Red Hat AMQ [32]	✓	✓	✓	✓	Both	✓	✓	✓	✓	✓	✓	Queue access is limited by special characters [78]. Non-persistent messages are lost when brokers stop [78].	Enables real-time integration [32]. Supports multi-message patterns for real-time messaging [32]. Supports multi-languages, including Java, C, C++, Python, Ruby, & .Net [32]. Supports mission-critical applications [32].
Celery [33]	✓	✓	✓	✓	Push	✓	✓	✗	✓	✓	✓	Compatibility and integration with other brokers can be complicated [33]. Overall complexity [33]. Monitoring and management are challenging [33]. Number of connections is limited by 10 connections [33].	Enables operations to manage and maintain distributed task queues ,such as starting, stopping, and restarting worker processes [33]. Functions as a task queue [33]. Focuses on real-time processing [33]. Supports task scheduling [33]. Supports multi-message brokers [33]. Integrates with multi-web frameworks [33]. Supports automatic retry in the event of connection loss or failure [33].
JBoss Messaging [34]	✓	✓	✓	✓	Push	✓	✓	✓	✓	✓	✓	Delay may occur with large messages (up to 100KB) due to split message into multiple packages[79].	Supports AMQP, MQTT, STOMP message protocols [79]. Supports transactions [79]. Provides management processes related to deployments, configuration, and access control [79]. Easy integration with other JBoss and Java EE components [79].
OpenMQ [35]	✓	✓	✓	✓	Both	✓	✓	✓	✓	✓	✓	Setup is complex [35]. High Latency [14].	Loosely-coupled architecture [35].
Beanstalk [36]	✓	✓	✗	✓	Pull	✓	✓	✗	✓	✗	✗	Lacks authentication [36]. Message size is limited to 64 KB [80].	Unprocessed messages are automatically returned to the queue [80]. Supports Ruby, Rails, Java, JavaScript, Haskell, and PHP [81].
Gearman [37]	✓	✓	✗	✓	Both	✗	✓	✗	✓	✓	✓	Does not have authentication and SSL support [37]. Manual configuration [37]. Monitoring tools are limited [37].	Used by LiveJournal, Yahoo!, and Digg [37]. Multi-languages support [37]. No single point of failure [37]. No limits on message size [37]. Supports load balancing [37].
Enduro/X [38]	✓	✓	✓	✓	Both	✓	✓	✗	✓	✓	✓	Message size is limited to max 10 MB [82]. Buffer size is limited to max 64KB [82]. Cluster nodes number is limited to max 32 nodes [82]. Resource managers numbers with single transaction are limited to max 32 [82]. Versions compatibility depends on the date of release [38]. Limitations on availability of the operations that can be executed within the callback [82].	Distributed transaction processing [82]. Works on multi-platforms [38].
WSO2 [39]	✓	✓	✓	✓	Both	✓	✓	✓	✓	✓	✓	Heap memory size allocation is limited to max 4GB [83].	Supports widely used protocols such as HTTP/S, JMS, VFS, UDP, TCP, MQTT, MSMQ, and MailTo [84]. Supports message filtering [85]. Integrates easily with other WSO2 products and third-party systems [39].
HiveMQ [40]	✓	✓	✓	✓	Push	✓	✓	✓	✓	✓	✓	Number of characters broker accepts in an Client ID is limited between 1 and 65535 [86]. Number of characters broker accepts in a topic string is limited between 1 and 65535 [86]. Resource intensive for maintenance [86].	Is a client-based MQTT broker for M2M communication [40]. Suitable for mission-critical applications [40]. Supports real-time monitoring of device data & integration with existing systems [40].
Redis [41]	✓	✓	✓	✓	Push	✓	✓	✓	✓	✓	✓	Uses a memory dump which leads to slow performance [41]. Has only basic security options [87].	Supports multiple data types [41]. In-memory data storage [41].
EMQX [42]	✓	✓	✓	✓	Both	✓	✓	✓	✓	✓	✓	Setup, configuration and management are complex [88].	Supports MQTT bridging [42]. Supports data integration [42].
Apache Pulsar [43]	✓	✓	✓	✓	Push	✓	✓	✓	✓	✓	✓	Complex configuration and deployment [43]. Complex architecture, based on four components (Pulsar servers, Apache BookKeeper, Apache ZooKeeper, and the RocksDB database) that need to be configured and managed [43].	Supports event streaming [43]. An index-based storage system [43]. Low latency [43]. Supports messaging, streaming, and queuing [43].

Table 2. A summary of open source and non-priority-supporting message brokers.

Message Brokers and Queues	Clustering Support	Monitoring Support	Pub/Sub Support	Parallel Processing	Pull & Push Support	Reliable Delivery	Persistent	Authentication	Scalable	Distributed	Fault Tolerance	Shortcomings	Features
Apache Kafka [44]	✓	✓	✓	✓	Pull	✓	✓	✓	✓	✓	✓	Resource intensive [44]. Provides a data backlog [44]. Complex [44].	Supports topic (log) compaction & distributed event streaming [44]. Supports data integration [44]. Language support [44]. Supports multiple data formats [44]. Supports permanent storage & the management of data flow and consumer groups [44]. Supports replication and partitioning of data [44]. Supports deployment in different environments [44].
Apache RocketMQ [45]	✓	✓	✓	✓	Both	✓	✓	✓	✓	✓	✓	Message size is limited to max 4MB [45]. Message sending retries is limited to max 3 times [45].	Supports message broadcasting, tracking, filtering, and retrying [45]. Low latency [45]. Maintains the order of messages [45]. Supports multi-protocols [45]. Supports multiple programming languages [45].
Eclipse Mosquitto [46]	✗	✓	✓	✗	Both	✓	✓	✓	✓	✓	✓	Limited security [46]. No built-in clustering [89]. Unsuitable for large-scale deployments [89]. Deployment is challenging in a cloud environment [89]. Message size is limited to max 256MB [46].	Low resource usage [90]. QoS support [46]. Topic-based message filtering [46]. Supports logging and debugging [46]. Supports functioning as a bridge [46]. Supports dynamic restart configuration [46]. Suitable for low-power machines [90].
ZeroMQ [47]	✓	✓	✓	✓	Both	✓	✗	✓	✓	✓	✓	High load of local control modules [14]. Fails to manage relationships between all network components [14]. Limited security [91]. Scaling is challenging [92]. Delivery is not guaranteed [92].	Brokerless messaging platform [47]. Multi-languages and platforms support [47]. Carries messages across IPC, TCP, TPIC, and multicast [47]. Low latency [47].
Apache NiFi [48]	✓	✓	✓	✓	Both	✓	✓	✓	✓	✓	✓	Data extraction is difficult when a node is separated from a cluster [48]. Under certain conditions, data is automatically deleted [48]. Complex configuration [48].	Provides a data flow framework [48]. Provides data compression using a user-specified algorithm to reduce data size [48]. Prevents data loss by controlling data flow and stopping the production of more data than a queue can handle [48]. Supports buffering of all queued data [48]. Integrates and processes multiple data sources [48].
Ably Realtime [49]	✓	✓	✓	✓	Both	✓	✓	✓	✓	✓	✓	Caps the number of channels per connection [49]. Peak connections number limited between 200 and 240 [49]. Message size limited to 16KB [49]. Number of queues limited to 5 [49]. Queue length limited to 10,000 [49].	Addresses challenging real-time requirements [49]. Supports streaming data [49]. Supports multiple protocols [49].
Apache SamZa [50]	✓	✓	✓	✓	Pull	✓	✓	✗	✓	✓	✓	Only supports JVM languages [93]. Configuration is complex [50]. Resources intensive with large data volumes [94].	Stream processing framework [50]. Supports message storage, routing, and processing management [50]. Supports at-least once data processing [50]. Real-time data processing with low latency [50]. Easy to integrate [50].
VerneMQ [51]	✓	✓	✓	✓	Both	✓	✓	✓	✓	✓	✓	Lack of security [14]. Clustering architecture is unproofed [95]. Limited enterprise features [95]. Not under active development [95]. Limited support for MQTT integration [95]. Lacks management and monitoring features [95]. No cloud-based service [51].	Master-less clustered messaging protocol [51]. Supports flow control [14]. Low latency [51].
NServiceBus [52]	✓	✓	✓	✓	-	✓	✓	✗	✓	✓	✓	Scalability is limited due to use centralized resource [96]. Monitoring tools are limited [97]. Debugging is complex with huge stream of messages [96].	Ensures message processing [52]. Supports transactions and recovery is built-in [96]. Messages can be retried at regular intervals [96].
Kestrel [53]	✗	✓	✗	✓	Pull	✓	✓	✗	✗	✗	✗	Low support of security [53]. Low clustering capabilities [53]. Memory size is limited to max 128MB [53]. Number of items in the queue is limited to 500 [53]. Data size of each item in the queue is limited to max 32bytes [98].	Written in Scala [53]. Each server handles ordered MQs, with no cross communication, resulting in a cluster of k-ordered queues [53].
NSQ [54]	✓	✓	✓	✓	Push	✓	✓	✗	✓	✓	✓	Data loss with server crash [54]. Messages are unordered [54]. Limited persistence [54]. No message recovery [54]. Lacks replication [54]. Messages are delivered at least once, which may duplicate messages [54].	Load-balanced message delivery [54]. Efficient handling of high-volume& real-time data streams [54].
NATS [55]	✓	✓	✓	✓	Both	✓	✓	✓	✓	✓	✓	Message size is limited to max 64MB [99].	Suitable for real-time communication [99]. Easy to use [99]. Minimal resource consumption [99]. Offers persistence with "at-least-once" and "exactly-once" [99].
KubeMQ [56]	✓	✓	✓	✓	Both	✓	✓	✓	✓	✓	✓	Unsuitable to all use cases due to it's designed for dynamic microservice environments [100].	Builds a hybrid infrastructure across clouds, on-prem, and at the edge to allow microservices from multi-environments to communicate [56]. Support for pub/sub, microservices, multistage pipeline, and tasks queue use cases [56]. Runs in Kubernetes and connects natively to the K8S cloud-native ecosystem [56]. Simple deployment in Kubernetes [56]. Easy to use [56]. Low latency [56].

Table 3. Summary of proprietary and priority-supporting message brokers.

Message Brokers and Queues	Clustering Support	Monitoring Support	Pub/Sub Support	Parallel Processing	Pull & Push Support	Reliable Delivery	Persistent	Authentication	Scalable	Distributed	Fault Tolerance	Shortcomings	Features
IBM MQ [57]	✓	✓	✓	✓	Push	✓	✓	✓	✓	✓	✓	High costs [101]. Problems with message prioritization due to unordered way of allocating messages [102]. Does not always integrate with the newest forms of messaging [102]. Messages are unordered [102].	QoS support [103]. Provides robust monitoring & tracing of all messages [103]. Controls undelivered messages [103]. Multi-APIs support [103]. Allows applications to be decoupled [103]. Easy to deploy on various platforms [103].
Amazon SQS [58]	✗	✓	✓	✓	Pull	✓	✓	✓	✓	✓	✓	High scale-up cost [58]. Message size is limited between 1KB and 256 KB [104]. Message ordering is not guaranteed [58]. Message retention before deletion is limited between 1 minute and 14 days [58].	Cloud-based web service [58]. Supports decoupling microservices, distributed systems, and serverless applications [58]. Transmits, stores, and receives messages across software components using SQS at any volume [58]. Messages are delivered at least once [58].
Microsoft MQ (MSMQ) [59]	✓	✓	✓	✓	Push	✓	✓	✓	✓	✓	✓	May experience resource failure [59]. Limitation on message size [59]. Drops all MSMQ messages if the appropriate server is not deployed [105]. Open queue failure error prevents data transfer [105].	Multi-protocols support [59]. Tracks and deletes expired messages [59]. Manages distributed brokers [59]. Supports remote access [59]. Effective routing [59]. Provides guaranteed message delivery [59]. Provides a store and forward mechanism [59]. Supports transactions [59].
Oracle GlassFish Server Message Queue [106]	✓	✓	✓	✓	Both	✓	✓	✓	✓	✓	✓	Resource intensive as the number of messages increases [106]. High latency as connections number to the broker increases [106]. Size of message is limited to max 70MB [106].	Supports transactions [106]. Supports JMS 1.1, STOMP, and HTTP protocols[106]. Well-known standards-based messaging support [106].
TIBCO Enterprise Message Service [62]	✓	✓	✓	✓	Both	✓	✓	✓	✓	✓	✓	Excludes fault tolerance of the server [107]. Recourse intensive as message size increased up to 512MB [108].	Supports load balancing [107]. Manages the real-time flow of information [62]. Supports multiple message protocols and technologies [107]. Easy integration with TIBCO eFTL™ software expands broker to web and mobile applications [62]. Loosely coupled design [62]. Supports integration for heterogeneous platforms [62].
TIBCO Rendezvous [61]	✗	✓	✓	✓	Both	✓	✓	✓	✓	✓	✓	Expensive [109]. Queue size limited to max 500 [110].	Supports C, C++, Java , & .NET programming language [111]. Easy to use & setup [111]. Has a distributed architecture to eliminate failure [111].
Anypoint MQ [63]	✓	✓	✓	✓	Pull	✓	✓	✓	✓	✓	✓	Expensive [112]. Payload size is limited to max 10 MB [63]. Converts the payload format, leading to an increase in payload size. [63].	Supports data integration [112]. Stores messages in a queue [63]. Provides intelligent message routing [63].
Azure Service Bus [64]	✓	✓	✓	✓	Both	✓	✓	✓	✓	✓	✓	Is a cloud-only service [64]. Message size is limited between 256 KB and 100 MB [64]. Number of queues is to max 10,000 [113]. Number of subscriptions per topic is limited to max 2,000 [64].	Provides duplicate detection, duplicate messages will not be stored in the queue [64]. Guarantees ordering [113]. Offers scheduling [114, 115]. Integrates well with other Azure products [115]. Supports multi-protocols [64]. Provides delivery guarantee (at-least-once, at-most-once) [113].
SAP NW PI [65]	✓	✓	✓	✓	Both	✓	✓	✓	✓	✓	✓	Message size is limited to max 350 MB [116]. Performance is directly affected by the message size [116].	Supports various integration patterns [65]. Supports message transformation [65].
Solace PubSub [66]	✓	✓	✓	✓	Both	✓	✓	✓	✓	✓	✓	Message size is limited to 64 MB [117]. Broker connections is limited to max 1000, some functions are not available with the default 100 connections [117]. Guaranteed messaging is not supported over connections [117].	Provides dynamic message routing [118]. High availability & high-performance [118]. Provides distributed tracing [66]. Event-driven architecture [118]. Supports multi-protocols [118].

- *Authentication support* [130] is vital for permitting only authorized users and systems to publish or subscribe to messages, especially in systems dealing with sensitive data.
- *Scalability* [131] refers to a message broker's ability to handle increasing loads from a large number of concurrently connected clients.

Table 4. Summary of proprietary and non-priority-supporting message brokers.

Message Brokers and Queues	Clustering Support	Monitoring Support	Pub/Sub Support	Parallel Processing	Pull & Push Support	Reliable Delivery	Persistent	Authentication	Scalable	Distributed	Fault Tolerance	Shortcomings	Features
Google Cloud Pub/Sub [67]	✗	✓	✓	✓	Both	✓	✓	✓	✓	✓	✓	Unsuitable for large-scale deployments due to limitations on resources [67]. Message size is limited to max 10MB [67].	Provides real-time stream analytic [67]. Handles the underlying infrastructure, including provisioning servers, monitoring, scaling, backups, and security updates [119]. Provides service maintenance feature that suitable for Google's most fundamental products to serve all customers effectively [67]. Provides system maintenance feature to detect any issues with releases by continuously-running tests it is before used by customers and by monitoring [67]. Integrates with other Google Cloud services [67]. Supports automatic retries and message ordering [67].
Azure Storage Queue [71]	✗	✓	✓	✓	Pull	✓	✓	✓	✓	✓	✓	Orders messages randomly [113]. Message size is limited to max 64 KB [71].	Maximum number of queues is unlimited [113]. Activity monitoring support [71]. Supports storing large numbers of messages [71]. Messages are delivered at-least-once [113]. Does not provide duplicate detection [113].
Amazon MQ [68]	✓	✓	✓	✓	Both	✓	✓	✓	✓	✓	✓	Number of broker connections is limited to 1,000, or 100 for micro brokers [120].	Supports Standard Java Message Service (JMS) features [120]. Performs maintenance to the hardware, operating system,& the engine software a message broker [120]. Integrates with other AWS services and applications [68]. Supports distributed transactions [120]. Multi-protocols support [120].
Intel MPI Library [69]	✓	✓	✗	✓	Both	✓	✓	✓	✓	✓	✓	Expensive [121]. Compatibility issues with some systems [69, 122]. Other processor architectures are not supported [69].	Uses OpenFabrics Interface (OFI) to handle all communications [69]. Establishes the connection only when needed, which reduces the memory footprint [69]. Chooses the fastest transport available [69]. Supports multi-cloud platforms [69]. Supports multi-cluster interconnects [69].
Amazon Kinesis [70]	✓	✓	✓	✓	Push	✓	✓	✓	✓	✓	✓	Permission issues [123]. Costly as data volume increases [70]. Data payload size is limited to max 1MB), data read rate to to 1MB/s, and number of consumers for each data stream to max 20 [124]. Operations are rate-limited [124]. Limitation on number of data streams [124]. Each record is added to a buffer with a deadline [124].	Provides buffering and processing of real-time data streaming [70]. Serverless streaming data service [70]. Provides reliable data processing and delivery with checkpointing and error-handling [70, 125]. Offers monitoring and management [70]. Offers various developer tools [70].
IronMQ [72]	✓	✓	✓	✓	Both	✓	✓	✓	✓	✓	✓	Expensive [126]. Limited control [127].	Meets the needs of both small businesses and large enterprises [127]. Supports multiple programming languages [127]. Uses REST API [72]. Easy to install [127]. Handles load buffering, synchronicity, and database offloading issues [127]. No limitation on the number of queues [72].

- Message brokers can operate in *distributed environments* [103], allowing them to work on multiple devices or locations, optimizing workload distribution.
- *Fault tolerance support* [103] enables the message broker to recover from failures and continue operating smoothly.

The evolution of message brokers as illustrated in Tables 1 to 4 highlights significant advancements in their features and capabilities, driven by by advancements in computing architectures, application demands, and integration with emerging technologies. In the early stages, clustering support was either absent or limited, posing challenges to scalability. This prompted brokers such as Apache Kafka and Intel MPI Library to provide robust clustering capabilities, enabling horizontal scaling and efficient multi-cluster interconnects to handle large message volumes effectively. Initially, monitoring tools were sparse and often relied on third-party integrations. Today, platforms like Google Cloud Pub/Sub and IBM MQ offer integrated monitoring features, facilitating proactive issue detection and performance optimization.

Current systems have moved beyond basic routing to support advanced topic-based filtering, as seen in Eclipse Mosquitto, and support multiple protocols and programming languages, such as Apache ActiveMQ. Reliable delivery mechanisms have also evolved from minimal guarantees like "at-most-once" delivery to robust mechanisms "at-least-once" delivery through acknowledgments, retries, and persistent queues, as exemplified by NATS, Azure Service Bus, and Azure Storage Queue. Furthermore, fault tolerance has seen significant progress, with features such as automatic retries helping to prevent data loss and downtime, as demonstrated by Celery and Google Cloud Pub/Sub.

With these capabilities, they are able to serve as the backbone for a wide variety of event-driven architectures and asynchronous communication patterns by serving as the basis for these architectures. Each feature reflects a critical capability necessary for brokers to meet the demands of modern, dynamic, and data-intensive applications, like GenAI applications. For example, these applications require several operational requirements, including scalability to meet dynamic demand, efficient data exchange management, real-time responsiveness, reliability, and robust security.

With scaling support, message brokers can distribute workloads across multiple nodes efficiently and meet GenAI applications' growing demands. Additionally, pub/sub support capabilities enable seamless asynchronous communication between multiple components in distributed environments. The message broker with parallel processing capabilities can efficiently handle multiple messages simultaneously, enabling high throughput and low latency (real-time responsiveness), which are key to GenAI applications' real-time requirements. Additionally, push and pull support ensures timely processing of high-priority or time-sensitive data while facilitating efficient data stream management.

The message broker equipped with monitoring tools is crucial, since real-time monitoring allows for detection and seamless recovery of issues such as latency, bottlenecks, and failures before they affect system performance. This capability improves reliability, for example latency-sensitive GenAI tasks, such as real-time translation and conversational AI. GenAI applications also require guaranteed delivery mechanisms, such as acknowledgments and retries, to ensure the integrity of critical data. This is particularly important in scenarios where data loss can lead to system failure. Also, the sensitive nature of GenAI systems' data-such as personal or medical information-demands robust authentication mechanisms to prevent unauthorized access and ensure data security.

Despite their inherent strength, message brokers have several limitations that can impact their performance. Among these limitations, for instance, message size, message sending retries, security features, queue length, monitoring tools, memory size limitations and the number of allowable broker connections. These limitations further highlight the challenges associated with these systems. For instance, brokers face in complex setup

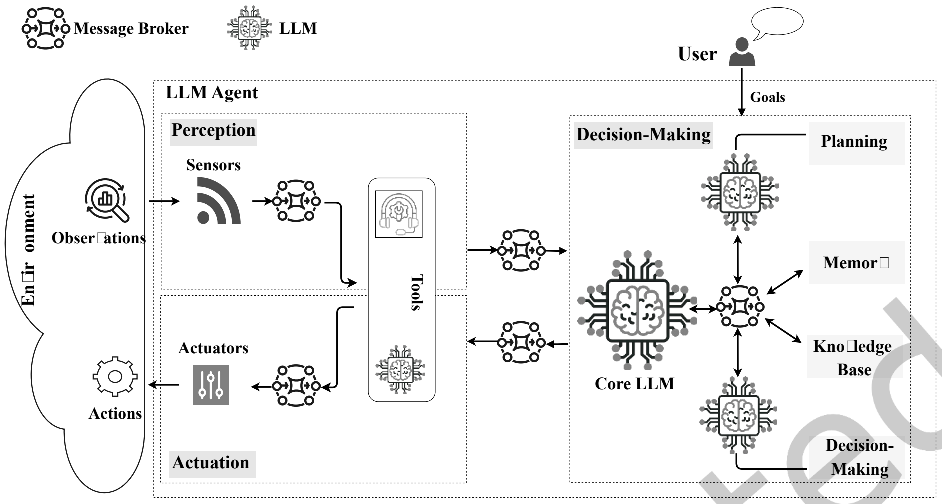

Fig. 3. The overall architecture of a GenAI agent, with possible integration points with message brokers.

processes and real-time processing demands, such as latency issues. Additionally, maintenance, monitoring, integration, large-scale deployments, and management can pose significant difficulties that contribute to considerable resource consumption.

Furthermore, the limited ability of some brokers to handle large volumes of data require the use of methods such as GenAI tools to enhance their capabilities and meet the growing processing and performance demands of GenAI applications. These applications require a dynamic and adaptive communication infrastructure capable of efficiently managing large-scale data generation. In the following section, we will explore the role of GenAI models in enhancing message brokers, the contribution of message brokers to GenAI, and advanced techniques designed to optimize their functionality within the GenAI context.

4 Message brokers and GenAI

The exploration of pub/sub communication patterns from the perspective of GenAI opens up a vision for the future, suggesting a range of benefits that could potentially enhance content delivery, personalization, and user engagement. Leveraging AI models such as GPT-3 [132] and its successors holds the promise of delivering customized content in real-time, tailored to the unique preferences of individual subscribers. This could be achieved by automatically generating human-like text that aligns with each subscriber's interests [7]. While the full realization of these benefits remains a subject for further research and development, the integration of advanced AI technologies with pub/sub systems offers a promising opportunities into the possibilities for more dynamic and personalized communication strategies. In this and following sections, we will attempt to shed light on this vision with practical examples and discuss how emerging enabling technologies can play a key role in this respect.

Language-based GenAI systems offers the capability to process subscriber queries and feedback efficiently using natural language understanding. These systems can power chatbots and virtual assistants, enabling users to communicate both interactively and intelligently with each other. Furthermore, their ability to summarize content, translate it into different languages, and moderate it significantly enhances the quality and accessibility of information [133]. Consequently, such systems enable the design of proactive information delivery mechanisms, including automated reporting, anomaly detection, and predictive analysis [134–136]. For instance, LLMs can learn from historical data and trends to autonomously analyze security logs and reports, generating comprehensive summaries that detail threat sources and attack paths while providing early warnings about potential threats [137, 138]. Additionally, within network environments, LLMs can predict peak usage periods by analyzing historical trends in user activity, workload demands, and performance metrics, allowing for proactive scaling of computing resources [139]. With these capabilities, LLMs can contribute to reliability and resource optimization.

These advancements in language processing and interaction capabilities signify a critical opportunity in the evolution of computing architectures, especially as we address the rising processing and performance demands of GenAI applications. The complexity and sophistication of these applications necessitate a robust architectural framework that is not only capable of supporting the intricate dynamics of GenAI operations but also adaptable to the novel types of data generated by GenAI applications. This architecture should be specifically tailored to leverage the unique advantages that GenAI offers, such as enhanced content personalization and user engagement, while still aligning with traditional scenarios where message broker technologies are pivotal. For instance, the architecture can integrate LLMs and Large Multi-modal Models (LMM) to enhance the capabilities of actuators and sensors, enabling them to extract more semantic information from data and identify combinational patterns among them [140]. This approach ensures that the system can dynamically adapt and respond to the evolving landscape of GenAI-driven communication, making it possible to abstract richer, more meaningful insights and foster synergistic interactions within the pub/sub ecosystem.

In this regards, central to our discussion is the conceptualization of the *GenAI agent model*, which exemplifies the architecture required to harness the full potential of GenAI applications Fig. 3. This model, segmented into *environment* interaction, *perception*, *decision-making*, and *actuation components*, serves as the backbone for integrating GenAI capabilities with pub/sub systems. It encapsulates the essence of GenAI's interaction with its surroundings, leveraging advanced computational engines like LLMs for processing and decision-making tasks.

The **perception component** perceives the environment through observations. It is equipped with sensing elements which may include physical sensors to gather diverse data from the surrounding environment, software-based tools, or both, along with a message broker designed to facilitate communication between these elements. These physical sensors can capture multi-modal observations, including visual, auditory, and textual data, as well as other modalities that can help the GenAI agent to understand its situation. The software-based tools such as LLMs, interface with abstract data streams. They read, analyze, and transform the gathered data into a comprehensible format for the agent's brain. These tools includes such as Multimodal-GPT [141], Flamingo [142], HuggingGPT [143], AudioGPT [144], GPT-4 [145], Visual ChatGPT [146], etc. Using sensors with LLMs enhances their capability to understand and react to changes in the environment, leading to more effective decision-making in dynamic situations.

The agent's **decision-making component**, or brain, executes memorizing, thinking, analyzing and decision-making tasks, supporting long and short-term memory, knowledge, and planning [147]. Short-term memory records recent tasks and actions, while long-term memory acts as an external database, enhancing the agent's ability to recall past conversations and pertinent details. Utilizing subgoal decomposition [148] and a chain-of-thought approach [149], the agent breaks down large tasks into multiple manageable subgoals that are processed by a group of LLMs models. Through self-critics and reflection [150, 151], the agent can learn from its errors, enhancing its capabilities iteratively.

The agent uses actual physical actuators, LLMs-based tools, or both to execute tasks in the **actuation component**. These elements allow agents to interact with and respond to their environments. The LLMs-based tools include text generation through text-based tools such as ChatGPT [152]. Moreover, agents' workspaces have been expanded with embodied actions to support their integration and interaction with the physical world. LM-Nav [153] analyzes input commands and the environment, aiming to identify the optimal walk based on a topological graph that is constructed internally. EmbodiedGPT [154] enables robots to comprehend and perform motion sequences in physical settings through multimodal visual understanding. Using these non-textual output tools extend the functionality of language models and agent scenarios.

Crucially, GenAI agents are also able to generate novel tools. With frameworks like CREATOR [155], agents can generate executable programs or merge existing tools into more robust ones. Furthermore, with frameworks such as Self-Debugging [156], agents can iteratively improve the quality of the generated tools, autonomously learning from past experience, self-correcting and adapting, enhancing their tool-generation capabilities.

Following milestone studies in edge intelligence [157–159], the interaction between brokers and GenAI can be separated into two different categories: the benefits that GenAI can bring to message brokers, as well as the benefits that message brokers can provide to GenAI, are illustrated in Table 5 and Table 6.

4.1 GenAI for message brokers

GenAI has the potential to complement and enhance the intelligence and efficiency of message brokers, particularly by supporting the prediction of routing decisions to avoid busy routes. LLM can analyze network status, traffic, routing data to evaluate network performance [135]. Based on this information, LLMs can automatically adjust routing strategies and allocate traffic to optimal paths, reducing latency. For instance, during times of network congestion, LLM can dynamically reroute some traffic to less congested paths, reducing latency and enhancing service quality [139].

Furthermore, GenAI's ability to identify patterns that may induce errors [135, 160–162] offers a promising avenue for augmenting message brokers with automatic corrective measures, potentially increasing system reliability and reducing downtime.GenAI's could assist in proactively distributing workloads or services across multiple nodes and determining the optimal time for scaling computing resources by analyzing historical usage patterns, workload demands, and performance metrics [139]. This potentially can minimize access times and enhance the efficient utilization of resources.

By leveraging GenAI's advanced capabilities in understanding, interpreting, and generating text [163], there is an opportunity to improve topic matching accuracy by analyzing the content of messages, enhancing the precision with which messages are delivered to their intended recipients. Additionally, GenAI's capacity for learning from ongoing interactions and its explainability could create a continuous feedback loop [164] that, when used alongside existing machine learning models, refines system performance over time. Table 5 highlights the integration of GenAI in message broker systems and addresses key issues, their impact of message broker systems, the GenAI solutions that can be applied to enhance traditional AI capabilities within a message broker system, and the associated challenges.

To concretely illustrate the enhancements that GenAI brings to message brokers, we present a representative use-case scenario involving a smart city infrastructure. Consider an IoT ecosystem with a message broker responsible for managing communications between thousands of devices across a smart city infrastructure. This scenario is visualized in Figure 4, which illustrates how a message broker facilitates communication between IoT devices, AI modules, and system operators. Traditional AI modules within the broker analyze sensor data to facilitate basic routing and load balancing. However, with the integration of GenAI, the system gains a significantly broader and deeper analytical capability—in particular, the ability to process and interpret large volumes of unstructured log data and textual feedback from devices and users in real-time, which traditional AI modules are not typically equipped to handle. For instance, GenAI models, trained on large and diverse datasets, can analyze historical trends and real-time environmental signals to predict traffic congestion on network pathways [165]. This semantic-level understanding and forecasting ability enables the message broker to dynamically reroute data flows, reducing latency and avoiding congested network nodes in ways that go beyond traditional rule-based or statistical methods. In this respect, and still referring to the example shown in Figure 4, GenAI modules—such as LLMs or foundation models fine-tuned for system log understanding—can interpret user

Table 5. GenAI for Message Broker.

Issues	Impact on Message Broker	GenAI Solution	Challenge
Routing & Filtering & Matching	- Accurate routing decisions for efficient message delivery. - High accuracy for filtering and matching of messages.	- Prediction of routing decisions to avoid busy routes - Filtering of messages according to specific criteria. - High matching accuracy by analyzing the content of messages	- Handling large-scale dynamic message flow. - Maintaining low latency for real-time systems.
Load Balancing	- Bottlenecks due to uneven workload distribution. - Coordination across multiple nodes.	- Proactive distribution of workloads or services across various nodes. - Minimizing access times.	- Adapting to fluctuating workloads. - Ensuring fair distribution of computational resources.
Failure	- Downtime impacts system reliability and user trust. - Potential for failures in distributed systems.	- Increasing system reliability and reducing downtime. - Identify patterns that may induce errors or security breaches. - Automatic corrective measures.	- Predicting failures in highly dynamic and unpredictable environments. - Addressing failures before they impact other systems. - Ensuring system recovery.
Scalability	- Systems must handle growing data flow while maintaining efficiency. - Poor scalability leads to service degradation.	- Determining the optimal times for scaling resources. - Ensure efficient operation under varying loads. - Efficient resources utilization.	- Balancing cost-efficiency with performance.
Continuous Improvment	- Continuous adaptation to evolving requirements. - Learning from system interactions to improve over time.	- Learning from ongoing interactions. - Improving system performance over time.	- Integrating learning mechanisms without disrupting ongoing operations.

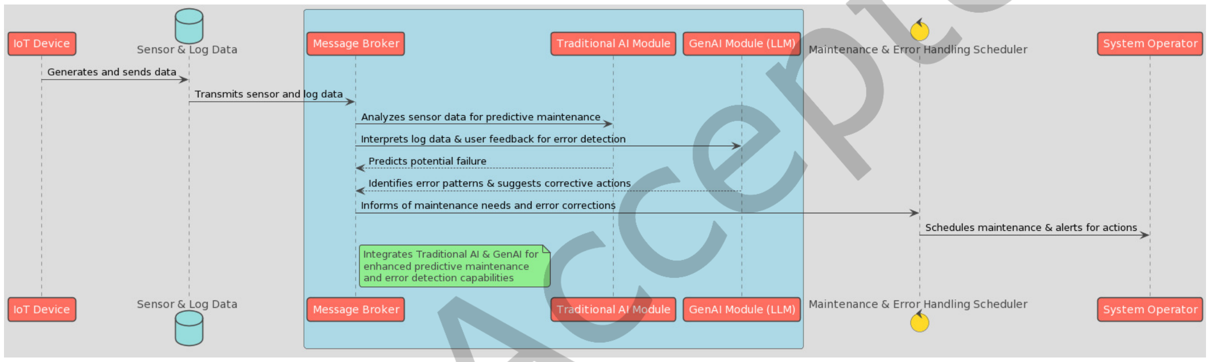
IoT Device
Sensor & Log Data
Message Broker
Traditional AI Module
GenAI Module (LLM)
Maintenance & Error Handling Scheduler
System Operator
Generates and sends data
Transmits sensor and log data
Analyzes sensor data for predictive maintenance
Interprets log data & user feedback for error detection
Predicts potential failure
Identifies error patterns & suggests corrective actions
Informs of maintenance needs and error corrections
Schedules maintenance & alerts for actions
Integrates Traditional AI & GenAI for enhanced predictive maintenance and error detection capabilities

Fig. 4. GenAI for Message Brokers in a Smart City Context. This diagram illustrates how GenAI modules enhance traditional AI capabilities within a message broker system deployed in a smart city infrastructure. The integration enables more intelligent routing decisions, semantic error detection, and adaptive resource scaling by leveraging real-time sensor data, user feedback, and predictive insights.

feedback and error reports, detect semantic anomalies, and suggest proactive corrective actions [136–138]. These modules complement traditional AI by handling more abstract and unstructured data inputs [166, 167], ultimately enabling more adaptive and context-aware broker decisions.

Moreover, GenAI's pattern recognition capabilities [168] extend to identifying subtle signs of potential system failures or security breaches before they escalate. By analyzing error logs and user reports, GenAI can pinpoint anomalies that traditional systems might overlook, enabling preemptive maintenance and strengthening the network's security layout.

In the context of resource scaling, LLMs can be used for time series analysis [169, 170] to intercept trends in data traffic and device engagement to forecast demand spikes [171, 172]. This foresight enables the system to scale resources up or down efficiently, ensuring optimal performance without wastage of bandwidth or computing power. The continuous learning aspect of GenAI [173], fueled by an ongoing feedback loop, ensures that the message broker's performance and decision-making processes improve over time, adapting to the evolving needs of the smart city infrastructure.

Table 6. GenAI on Message Broker.

Issues	Impact on GenAI	Message Broker Solution	Challenge
Scalability	High volume of data streams across distributed systems impacts flexibility.	- Dynamic load balancing - Horizontal scaling - Decoupling.	Ever-growing data volumes & connections overwhelm traditional brokers.
Fault Tolerance	Message loss during failures impacts reliability.	- Persistent message storage - Acknowledgments - Retries.	Complexity, especially in resource-constrained environments.
Low Latency	Delays disrupt real-time applications like chatbots.	- Optimized routing - Real-time queuing mechanisms.	Advanced methodologies (e.g., distilled models, computing & networking-aware orchestration, resource management techniques).
Heterogeneity	Difficulty integrating various sub-components.	- Seamless interoperability - Balanced offloading of tasks.	Ensuring consistent communication between heterogeneous components.
Dynamic Workloads	Resource bottlenecks due to unpredictable spikes.	- Manage the real-time data streams efficiently.	GenAI workloads are unpredictable & vary significantly over time.
Energy Efficiency	High energy consumption due to large-scale workloads.	- Energy-efficient message routing - Optimized resource utilization.	Balancing performance and energy efficiency in large-scale deployments.
Ethical & Privacy	Inaccurate or misleading outputs can disrupt critical applications.	- Monitor and detect anomalies.	Real-time validation, continuous monitoring, robust security measures, fine-tuning.

To conclude, we reaffirm that the integration of GenAI into message brokers opens new possibilities for enhancing both the intelligence and adaptability of broker systems. As showcased through the smart city example, GenAI modules can help overcome some of the limitations that are inherent to traditional AI approaches, especially when dealing with unstructured inputs and dynamic environments. Overall, this approach aims to enable brokers to support more context-aware decision-making, thereby improving service quality in complex, heterogeneous, and large-scale deployments.

4.2 GenAI on message brokers

GenAI introduces unparalleled content generation capabilities for advanced applications across diverse domains [7]. Future GenAI systems are expected to experience a data explosion due to the integration of multimodal systems [174]. This heterogeneous data explosion will span across a distributed edge-cloud continuum, placing increasing demands on current communication infrastructures [7].

As GenAI applications scale, managing large datasets across edge, fog, and cloud layers will require optimized distribution strategies that minimize inference latency [17]. Additionally, coordinating and managing the distributed inference process across distributed nodes is crucial for ensuring responsiveness in reactive applications [17], while also ensuring the integrity of AI-generated content. This requires communication infrastructure that can meet the requirements of GenAI applications. Table.6 highlights the critical issues affecting GenAI, emphasizes the importance of message brokers in addressing these issues, and outlines the associated challenges.

Message brokers, leveraging asynchronous communication capabilities [103], can operate to enhance GenAI task processing efficiency. This approach facilitates interactions that do not necessitate real-time communication, streamlining the processing of real-time data streams. By decoupling the sending and receiving processes, message brokers can offer a flexible and scalable solution for managing the complex data workflows associated with GenAI applications.

Central to message brokers is their robust mechanism to prevent message loss [130], which is key in conserving computational resources and safeguarding critical data for GenAI applications. These systems are equipped with tools designed to monitor and regulate message flow effectively. This functionality is fundamental in preserving the performance and operational integrity of GenAI by ensuring that data is processed efficiently and reliably, mitigating the risk of bottlenecks and data overflow.

Furthermore, message brokers facilitate service decoupling [103], enabling GenAI to manage growing workloads more effectively. The broker's ability to distribute tasks to different nodes allows GenAI to achieve a balanced load distribution [103], with different components running on different nodes. This is particularly important for GenAI deployments in the computing continuum, providing them with access to local environments with limited computational capacity. In such cases, a message broker can act as an intermediary, bridging sensors with the perception module and conveying actions to the actuators and responses to the user [3, 17].

Within the critical *brain component*, message brokers play a vital role in linking various sub-components, each offering distinct features or possessing heterogeneous computational resources and functionalities. By promoting interoperability and collaboration among diverse LLMs agents that contribute to decision-making, message brokers can significantly boost the performance of agents on complex tasks. This collaborative framework also facilitates the balanced offloading of tasks, optimizing the utilization of computing, communication, and storage resources across the network.
This highlights using agent frameworks such as AutoGen [175] and LangChain [176]. These frameworks enable building autonomous LLM agents configured to perform diverse tasks while collaborating within a coordination layer with minimal human intervention. These entities have the ability to integrate with external data sources, such as APIs and knowledge bases, enhancing their capability to access, process, and utilize information dynamically. Incorporating message brokers within the coordination layer contribute to manage the flow of information between agents, as well as enable decoupling of LLM agents (producers and consumers), allowing LLM agents and external tools or APIs to independently publish or consume messages without direct interdependencies. Additionally, message brokers ability to support the integration of agents or tools is beneficial in enabling parallel processing for complex workflows involving multiple agents. Furthermore, the asynchronous communication is essential for managing tasks with varying execution times or priorities, ensuring the system responsiveness. Message brokers can manage retries, storage, and error handling, ensuring the stability and reliability of these frameworks.
For instance, integrating Kafka within the AutoGen framework [177] established a responsive infrastructure that efficiently routed multimedia data to subscribing agents or tools based on topics. This integration allowed the system to manage multiple data streams and distribute them among agents without bottlenecks, enabling seamless interaction between LLM agents and external data sources.

However, integrating GenAI into message broker requires careful planning and customization to mitigate the risks of biased, misleading, or delayed outcomes. Addressing hallucination problems [178], inference latency [7], privacy [134], and ethical considerations [179] is paramount to align GenAI implementations with specific system objectives [134]. For further details, as illustrated in the Table 6, one prominent challenge is model hallucination, where the AI generates inaccurate content [178]. This issue can significantly impact message brokers by propagating misinformation, thereby lack of user trust. Enhancing message broker through implementing robust validation mechanisms to verify model outputs and fine-tuning the model with domain-specific data is essential strategies to address this issue [180].

Another key concern is inference latency, which refers to the delay between a request to the model and the generation of its response. High latency disrupts time-sensitive applications, such as real-time communication, creating bottlenecks and reducing the overall throughput of the message broker. Consequently, message brokers must adopt advanced methodologies to manage the extensive data volumes generated and consumed by GenAI applications effectively and minimize computational demands in this context [181]. As example for these methodologies, implementing computing- and networking-aware orchestration, resource management techniques [182], and deploying lightweight versions of the model or small language models, such as distilled models. These methodologies are not only essential for enhancing connectivity but also play a crucial role in reducing the computational demands and latency that are often associated with GenAI tasks. Additionally, ethical and privacy concerns present significant challenges, as GenAI may generate biased, harmful, or malicious content, compromising user trust [134, 179]. Addressing these issues requires integration broker with continuous monitoring, robust security measures [183].

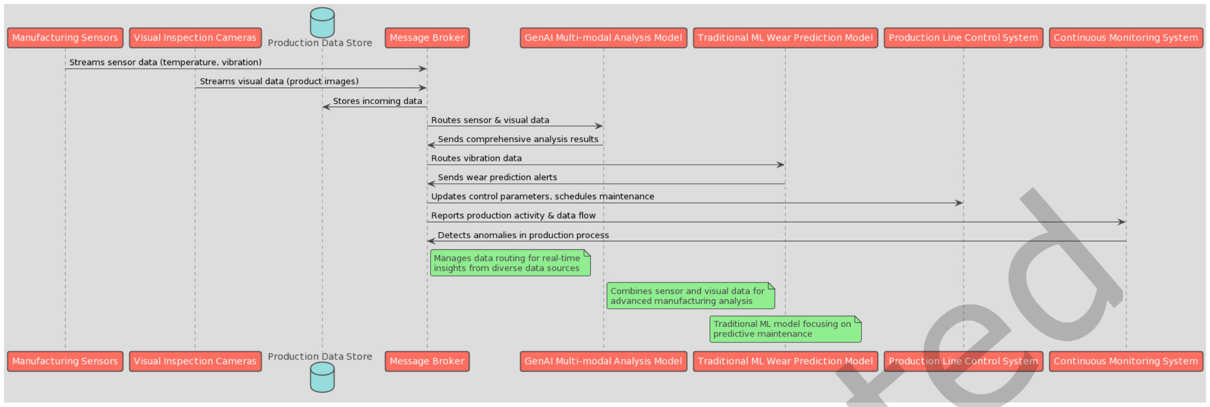
Manufacturing Sensors
Visual Inspection Cameras
Production Data Store
Message Broker
GenAI Multi-modal Analysis Model
Traditional ML Wear Prediction Model
Production Line Control System
Continuous Monitoring System
Streams sensor data (temperature, vibration)
Streams visual data (product images)
Stores incoming data
Routes sensor & visual data
Sends comprehensive analysis results
Routes vibration data
Sends wear prediction alerts
Updates control parameters, schedules maintenance
Reports production activity & data flow
Detects anomalies in production process
Manages data routing for real-time insights from diverse data sources
Combines sensor and visual data for advanced manufacturing analysis
Traditional ML model focusing on predictive maintenance

Fig. 5. Message brokers for GenAI. This diagram illustrates a smart manufacturing system where a message broker facilitates the flow of multimodal data from sensors and cameras to a GenAI model and a traditional ML model. The GenAI model performs comprehensive analysis by combining diverse data types, while the ML model focuses on predictive maintenance tasks.

In practical terms, there are several technical aspects contribute to enhancing the suitability of message brokers for GenAI applications. Among these technical aspects are model compression and model training acceleration strategies. The integration of these strategies within message broker can significantly mitigate GenAI's computational requirements and delays, addressing one of the major challenges in deploying these advanced systems efficiently. Furthermore, message broker with intelligent resource management algorithm can distribute workloads based on each node's capacity significantly, enhancing the performance of the distributed system and ensuring optimal utilization of computational and storage resources while maintaining smooth operation, even under varying loads.

Embedding semantic communication techniques within message brokers may contribute in management and real-time analysis of vast data volumes generated by GenAI applications. This include integrate techniques to transform GenAI input and output data into priority-based smart data, facilitating more timely and effective processing. Embedding a prioritization mechanism similar to the QoS levels defined in MQTT [184] serves as a suitable approach, ensuring that critical tasks are processed with the urgency they require. This adaptation enhances the responsiveness of GenAI systems by ensuring that high-priority data is attended to promptly, mirroring the efficiency and reliability seen in established communication protocols.

Additionally, integrate techniques to select the right models and their continuous adaptation in response to evolving data landscapes. This is crucial for maintaining the accuracy and relevance of GenAI outputs. In this context, incorporating methods for model fine-tuning [185], continual and in-context learning within message brokers can enable dynamic adjustments to the models based on real-time data, ensuring that the GenAI system remains effective and up-to-date. This requirement underscores the necessity for message brokers to support not just the routing and handling of messages, but also the intelligent adaptation of GenAI models to changing conditions, thereby maximizing the potential of GenAI applications in diverse environments.

In this respect, embedding a continuous monitoring system within the message broker to promptly detect anomalies and data loss becomes fundamental. This system enables corrections and adaptations in real time,

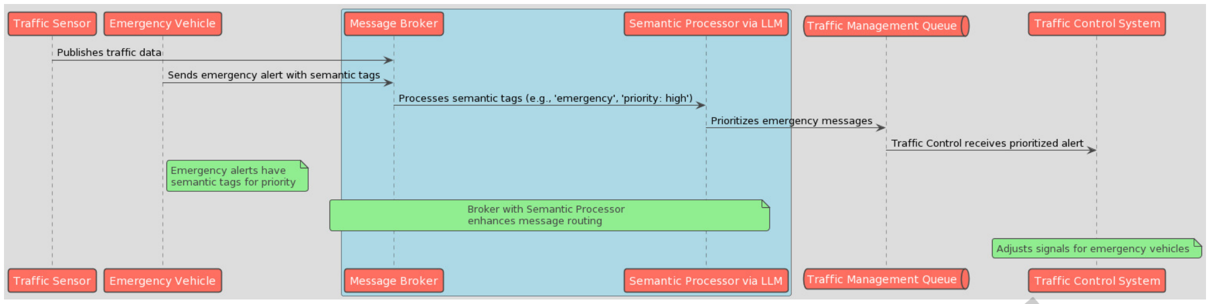
Traffic Sensor
Emergency Vehicle
Message Broker
Semantic Processor via LLM
Traffic Management Queue
Traffic Control System
Publishes traffic data
Sends emergency alert with semantic tags
Processes semantic tags (e.g., 'emergency', 'priority: high')
Prioritizes emergency messages
Traffic Control receives prioritized alert
Emergency alerts have semantic tags for priority
Broker with Semantic Processor enhances message routing
Adjusts signals for emergency vehicles

Fig. 6. GenAI-enabled and Semantic Communication. This diagram illustrates a traffic management system where a message broker, enhanced with semantic processing and LLMs capabilities, prioritizes and analyzes traffic data and emergency alerts.

fostering the necessary model adaptation and ensuring that the GenAI systems remain both robust and responsive to the dynamic nature of real-world data and application demands.

We have outlined several key aspects on how message brokers can facilitate the integration and management of GenAI applications within various domains. For instance, Figure 5 demonstrates a practical application in smart manufacturing, where a message broker orchestrates the flow of multimodal data to both GenAI and traditional ML models. This setup enhances real-time decision-making and operational efficiency, showcasing the dynamic capabilities of message brokers in supporting advanced analytics.

As we noticed, the integration message broker within GenAI require advanced message broker that leverage distributed architectures to schedule tasks, ensure scalability, and maintain correctness in dynamic environments. However, the central challenge is the broker's need to adopt new methods and techniques to address scalability, semantic communication, and distributed inference challenges. In the following sections, we will further discuss advanced methods designed to enhance the functionality of message brokers within the GenAI context, overcoming deployment challenges [3, 17]. By tackling these issues, we aim for the seamless integration and optimal performance of GenAI applications. This exploration will include discussions on how specific platforms and frameworks can be leveraged to enable our envisioned approach with insights into the integration of existing and advanced methods and techniques within specific brokers, highlighting their potential adaptations in the analyzed context.

4.3 Semantic Communication

Effective management and real-time analysis of vast data volumes are crucial for the development of GenAI applications. This necessitates a flexible system capable of accommodating a variety of data types with precision. Traditional message broker systems, while efficient in basic data routing, often struggle to cope with the complexity and volume of data typical in GenAI environments, limiting their effectiveness in scenarios requiring nuanced understanding and processing of data content. To address these limitations and reduce the strain on communication networks, embedding efficient communication mechanisms, such as semantic communication [186, 187], within message brokers is essential. This integration, particularly leveraging the capabilities of LLMs, can enable intelligent, automated feature selection that aligns with subscriber needs, enhancing the broker's ability to manage data-rich content more effectively [188]. While this approach may not directly minimize latency due to the inference time required by LLMs, it significantly improves the efficiency of data search, match, and mapping processes, thereby optimizing overall system performance in handling and distributing relevant information.

Following this, message brokers equipped with dynamic prioritization capabilities can intelligently identify and route high-priority messages by incorporating semantic communication technology. This prioritization allows for the handling of messages based not just on the criteria within the message header, but also on the content itself, enhancing the relevance and timeliness of information delivery. Although integrating semantic communication with a broker introduces demands for high scalability, processing power, and memory to manage large datasets effectively, it is a crucial step towards mitigating network congestion and optimizing the use of network resources. Moreover, this sophisticated processing capability must be balanced with robust security features to ensure sensitive data is handled securely, highlighting the need for a comprehensive approach to upgrade message broker systems for the GenAI era.

In delay-sensitive applications like healthcare, this integration is vital to assigning priorities based on its deep understanding of data patterns and sensitivities and its subscribers' specific needs. For example, in the healthcare scenario, GenAI can analyze large amounts of medical data to identify urgent cases, alerting healthcare professionals of critical patient needs or alarming health trends [189]. In a similar fashion, the effective synergy of LLMs with semantic processing capabilities within message brokers can also be observed in smart city traffic management (Fig.6). Here, the combination of semantic tags from emergency vehicles and real-time traffic sensor data, when processed through an advanced LLM, enables the system to prioritize and analyze critical information promptly. This GenAI-enhanced approach not only interprets the urgency and context of incoming data but also predicts and optimizes traffic flow in response to dynamic urban conditions. By doing so, it ensures that emergency responses are effective, minimizing delays and improving public safety. Building upon the foundation laid by traditional ML techniques, GenAI complements these approaches by incorporating advanced natural language understanding and context-aware processing capabilities. This allows for a more peculiar analysis and interpretation of complex data sets, enhancing the system's ability to make informed decisions rapidly and accurately.

For instance, embedding semantic ontology model in Apache Kafka [190] can enhance its routing and delivery capabilities, enabling the broker to understand the meaning and context of each message. Additionally, Eclipse Mosquitto [46], with its support for topic-based message filtering, can be enhanced through semantic communication by embedding ontologies, semantic tags, or metadata [191] within topics. This enhancement can enable intelligent and context-aware filtering, allowing the broker to understand and process messages based on their meaning and relevance. Solace PubSub provides dynamic message routing [118], which can be further enhanced by embedding semantic communication. By incorporating semantic annotations into messages, the system can enable context-aware routing, ensuring that messages are directed to the most relevant subscribers based on their content, meaning, or priority, improving the efficiency of message delivery.

Furthermore, Apache RocketMQ, with its support for message broadcasting and filtering [45], has the potential to offer advanced message routing and filtering capabilities by leveraging Natural Language Processing (NLP) for interpreting messages' semantic content, allowing for context-sensitive handling [17]. Finally, Anypoint MQ's intelligent routing [63] can be enhanced by integrating a knowledge graph that represents relationships between entities such as topics and subscriptions [192]. This integration enables dynamic routing decisions based on both content and context.

However, implementing this approach presents several challenges. Some semantic communication techniques are computationally intensive to meet real-time inference demands [193]. This highlight the need for novel, distributed, and intelligent resource management methods to enable real-time responsiveness in various applications. Additionally, semantic communication mechanisms enhance the broker's ability to process and interpret content contextually, enabling selective filtering of sensitive data before transmission. Nevertheless, processing sensitive data requires robust security measures to ensure its protection during processing and transmission while maintaining user trust.

4.4 Dynamic Data and Model Management

GenAI-based applications require a data and model management system that not only simplifies the construction of AI models but also optimizes efficiency and effectiveness. Such a system should minimize the need for human intervention in selecting ML models, enhancing real-time responsiveness with high model's accuracy, and improving real-time inference capabilities when integrated with message brokers. LLMs can expedite the model selection process and boost the deployment efficiency and precision of AI solutions [194, 195]. Moreover, integrating GenAI models with message brokers involves managing and directing the related data flows.

For instance, Apache Pulsar [43] with its distributed streaming capabilities, can be enhanced by integrating it with LLMs to enable autoscaling for efficiently managing large datasets required for training and inference in GenAI applications.

GenAI models, such as LLMs, frequently process sensitive or personal data, robust security measures are critical. For example, Google Cloud Pub/Sub [67] incorporates multiple integrated security measures to protect confidential data being transmitted, including authentication, encryption using Google-managed keys, and advanced Data Encryption Key (DEK) technology [67]. However, the security of the broker can be further enhanced by employing GenAI tools to enhance security protocols and improve the automation of key cybersecurity processes [134, 196]. These improvements may include automated reporting, threat intelligence analysis, and malware detection [196], ensuring further resilience of Google Cloud Pub/Sub security framework.

Amazon Kinesis [70] provides data transmission capabilities through synchronous data replication, checkpointing mechanisms, and error-handling strategies [70, 125]. However, to further enhance its reliability in GenAI applications, several improvements can be introduced, such as the integration of AI-driven anomaly detection to proactively identify and mitigate transmission failures. Additionally, implementing self-healing mechanisms can enable automatic detection and rerouting of failed data streams. These improvement would make Amazon Kinesis more robust solution for real-time data transmission for GenAI applications.

Further, Apache Kafka's [44] robust architecture, equipped with a stream processing feature enables it to efficiently handle parallel processing tasks for GenAI applications. However, it can be improved by integrating with predictive analytics and AI-driven methods for dynamically scaling brokers, efficiently partitioning topics, and determining the optimal number of partitions.

Since processing GenAI models is computationally intensive, with large amounts of real-time data, it needs to be scalable and distributed to cope with the varying workloads. Running these models locally on edge devices is often impractical due to hardware limitations. This challenge is particularly significant in real-time applications where high-performance inference is essential. Such integration also must also consider the handling of streaming data.

Moreover, this integration requires seamless coordination between clients and brokers to exchange information, such as model architectures. Furthermore, Efficient and intelligent coordination is essential to prevent delays or data mismatches. Misalignment in understanding the model's structure between the client and broker can lead to incorrect deployments, repeated requests, or errors, which can degrade the system's performance and reliability.

4.5 Training Acceleration

The training of GenAI models requires a significant amount of computation and time. By incorporating training acceleration methods [197] into a message broker system, training time can be reduced, computational resources saved, and models deployed more rapidly. Message brokers play a pivotal role in this process by enabling the distribution of training tasks across multiple nodes, which allows for parallel processing of data and computations. This parallelization accelerates the training process by breaking down complex tasks into smaller, manageable units that can be processed simultaneously, reducing the overall time required to train and deploy large GenAI models. It also facilitates faster and more efficient inference for LLMs. Moreover, message brokers monitors resource utilization and reallocates tasks dynamically in real time to balance computational demands across nodes.

The most common technique of training acceleration is sequence parallelism (SP), in which the prompt sequence is divided into smaller sub-sequences and processed in parallel [198, 199]. Another method involves selective activation re-computation, in which only the necessary parts of the GenAI model are recalculated during training, rather than the entire model [200]. A fine-tuning technique involves adjusting the parameters of an existing model rather than training a new one from scratch. As a result, training data and computations can be reduced [201].

Furthermore, tensor parallelism works by splitting the model across multiple GPUs and processing different parts of the model in parallel [202]. The mixed-precision training technique is one of the most effective ways to enable fast and efficient LLM inference on GPUs [203]. In this technique, the amount of memory required during training is reduced by using lower-precision data types.

The interaction between GenAI training systems and message brokers can significantly accelerate the training process by utilizing distributed computing and dynamic resource management. However, this approach faces challenges related to load balancing, data dependencies, and hardware compatibility. In a distributed training, tasks must be distributed across multiple and heterogeneous nodes and dynamically reallocating tasks to maximize resource utilization [204]. Intelligent task scheduling algorithms are essential to dynamically distribute tasks while accounting for node capabilities and workload balance.

In addition, training GenAI models often involves handling large datasets that are divided into chunks and distributed across nodes. These chunks may have dependencies that need to be maintained to ensure accurate model training [198, 199]. Message brokers need robust mechanisms to track and manage data dependencies, ensuring proper sequencing and maintaining data integrity to prevent errors caused by incomplete data handling.

For instance, Apache ActiveMQ provides efficient management and resource allocation capabilities [27], which can be enhanced through LLMs to analyze historical usage patterns and predict optimal task distribution dynamically. Similarly, Google Cloud Pub/Sub [67] ensures scalable and reliable message delivery [119], a capability that can be further augmented by LLMs to predict workload fluctuations and proactively scale resources, ensuring optimal performance during large-scale model training and inference. Moreover, Apache RocketMQ offers capabilities for maintaining message order and supporting message tracking [45], which are essential for tracking and maintaining the dependencies among distributed data chunks during GenAI model training. Ensuring the correct sequencing of these chunks is critical to maintaining data integrity and preventing errors that arise from incomplete data processing. To further enhance these capabilities, LLM-powered mechanisms can be integrated to intelligently manage sequencing and real-time data integrity validation, ensuring accurate model training.

4.6 Dynamic Model Compression

Integrating GenAI models within message brokers can enhance analytical and predictive capabilities in event processing (routing, filtering, matching and prioritizing) based on learned patterns and context in real time, enabling faster decision-making. However, resource-constrained environments demand efficient resource utilization and low-latency responses. Integrating GenAI models on resource-constrained nodes in the computing continuum often requires model compression, especially with high-dimensional models, strenuous computational tasks, and low latency requirements [4]. In this context, lightweight versions of GenAI models can improve functionality and performance effectively while minimizing computational requirements [205]. Their ability to be deployed on edge devices reduces energy consumption can make them suitable for real-time applications and improving user experience in interactive systems [206, 207]. Achieving this involves adopting innovative model compression techniques in message broker.

Message brokers can provide essential capabilities such as scheduling and distributing compression tasks across multiple nodes, enabling parallel processing. This is particularly beneficial for handling computationally intensive compression methods that may not be suitable for resource-limited devices. By offloading and distributing these tasks efficiently, message brokers optimize resource utilization while ensuring minimal latency. Moreover, brokers continuously monitor the performance of compressed models to ensure consistent reliability and accuracy in responses.

Among the compression strategies, pruning is prominent, involving the removal of superfluous elements from a model to decrease its size and complexity [208] without a significant loss in performance. For GenAI models, this can involve techniques such as removing weights with smaller gradients or magnitudes, reducing parameters, and other optimization methods [209, 210]. Another technique is Knowledge Distillation, where a smaller 'student' model learns to replicate the functionality of a larger 'teacher' model [211–213]. This technique can reduce GenAI models into smaller, distilled versions, enhancing the performance of the student model and increasing inference speed [214, 215].

Furthermore, quantization methods are utilized to reduce the precision of model parameters, significantly lowering memory usage and computational needs substantially. This leads to a smaller model size with faster inference speeds [216]. Another technique is low-rank factorization, which simplifies weight matrices with lower-rank approximations to reduce model size and computational demands [217].

However, this integration introduces challenges, particularly related to computational intensity and coordination. Techniques such as pruning, quantization, and knowledge distillation are computationally demanding [218]. Performing these processes locally on resource-limited devices is often impractical due to constraints in processing power, memory, and storage. Additionally, this process becomes increasingly complex when multiple clients with diverse requirements interact with the broker simultaneously. These challenges highlight the need for novel, distributed, and intelligent resource management methods.

For instance, Celery with its support for task scheduling [33], can be enhanced to optimize compression task distribution across multiple nodes. By implementing an adaptive scheduling mechanism that dynamically considers resource availability and workload, Celery can optimize compression performance in real time. Similarly, Azure Storage Queue with its activity monitoring support [71], can be enhanced through AI-based predictive analytics for monitoring the performance of compressed models and detect any change in the accuracy. IronMQ [72] provides support for workload offloading issues [127]. This capability can be further enhanced by integrating LLMs tools to proactively analyze workloads and determine which tasks should be offloaded, ensuring distribution of computationally intensive processes.

4.7 Dynamic Orchestration

The integration of brokers with GenAI requires effective resource management to address the substantial computational demands of GenAI models, which can significantly influence the broker's operational efficiency. The broker must allocate and manage resources such as CPU, memory, and storage to handle computational requirements. GenAI models can be computationally intensive during inference or when processing large volumes of real-time events. To mitigate potential bottlenecks, brokers must ensure an even distribution of computational tasks across available nodes [3]. Furthermore, GenAI-enabled brokers can intelligently and dynamically allocate resources based on workload demands and the priority of critical GenAI-related tasks, thereby enhancing system responsiveness. However, the process of efficient resource usage and allocation requires parallel processing capability.

Some brokers are equipped with features that can be optimized to support GenAI tasks, significantly improving their efficiency in resource management. For instance, Celery provides capabilities for managing, maintaining, and scheduling distributed tasks across multiple nodes, thereby enhancing the efficiency of GenAI agents [33].

To further optimize its functionality for GenAI workloads, Celery can be enhanced with AI-driven dynamic task scheduling. This approach would enable intelligent workload distribution based on real-time system performance metrics, including GPU/TPU availability, memory utilization, and computational demand. Apache ActiveMQ provides efficient resource allocation [27]. Thus, optimizing Apache ActiveMQ through mechanisms such as elastic scaling method could be critical to providing intelligent resource allocation to ensure balanced workloads across different nodes. Amazon SQS provides a reliable queue service for handling messages that facilitates microservices and distributed systems decoupling [58]. Incorporating ML techniques for intelligent workload distribution across multiple queues can optimize resource allocation and reduce bottlenecks in large-scale GenAI pipelines.

Further, the robust architecture of Apache Kafka, which uses clusters of brokers to handle data distribution and partition topics for scalability, makes it excellent for supporting distributed event streaming [44]. To further enhance its performance, AI-driven algorithms can be integrated to dynamically adjust partitioning strategies based on real-time workload distribution. This approach would improve resource utilization and make Kafka more efficient for high-performance GenAI applications. Additionally, Azure Service Bus provides advanced scheduling features and message orchestration in distributed environments [114, 115]. These functionalities can be further optimized to support GenAI workloads by integrating AI-driven task scheduling that intelligently prioritizes GenAI workloads and allocates computational resources based on real-time system performance.

However, the integration of brokers with GenAI requires effective resource management to address the substantial computational demands of GenAI models. Therefore, there is need for novel, highly distributed, efficient, and secure resource orchestration methods to support the integration of GenAI within brokers. Such methods must employ advanced algorithms capable of dynamically monitoring, allocating, and optimizing resource usage, enabling proactive adjustments to mitigate potential bottlenecks. This may require development of a self-organizing multi-agent systems that can autonomously adjust their structure in response to changing workloads, along with semantic-based orchestration techniques for efficient coordination [17]. Furthermore, robust security mechanisms are essential to ensure secure resource orchestration, particularly in multi-tenant environments where multiple clients or applications share resources, to prevent misuse and maintain operational integrity.

4.8 AIOps/MLOps and Monitoring

MLOps, merging DevOps principles with ML, is central to the advancement of ML and AI, streamlining the lifecycle of GenAI models from development to deployment. A critical feature within this field is the monitoring of deployed models, crucial for the uninterrupted and reliable operation of message broker systems. This practice enables real-time insights into model metrics, resource usage, and system irregularities, creating a proactive environment for identifying and addressing issues promptly. Furthermore, MLOps facilitates the setting up of automated alerts and triggers, enhancing responsiveness to anomalies and minimizing downtime [17, 219].

A Continuous Diagnostics and Mitigation (CDM) program plays a vital role in network security by analyzing network behavior and thwarting unauthorized access, thereby enabling prompt responses and maintaining network integrity. Beyond autonomous configuration management and monitoring device availability, CDM programs conduct continuous health assessments of devices and evaluate their environmental footprint. This continuous surveillance helps identify potential threats, bolstering processes to enhance security measures. Furthermore, CDM ensures the protection of sensitive information against unauthorized access or breaches [9].

KubeMQ, which supports multi-stage pipelines [56], can be enhanced through the integration of CDM real-time performance monitoring to improve data pipeline management within MLOps frameworks. Moreover, HiveMQ and Amazon Kinesis can contribute to MLOps integration by facilitating real-time monitoring and alert systems integration, since they support real-time device monitoring [40, 70]. By integrating CDM-driven analysis, these

Table 7. Opportunities & Challenges for enhancing message brokers' functionality within GenAI

Opportunities	Corresp-ondong Section	Challenges
Semantic Communication to reduce communication networks strain and streamline the data-rich content transmission.	Section 4.3	Computationally intensive to do locally. Processing sensitive data requires robust security measures.
Dynamic Data and Model Management to minimize the need for human intervention in selecting ML models and enhance real-time responsiveness accuracy.	Section 4.4	Computationally intensive to do locally. May require coordination between clients and broker to exchange information on, e.g., model architectures.
Training Acceleration to reduce training time, save computational resources, and rapidly deploy models.	Section 4.5	Requires ability to manage load balancing & data dependencies. Compatibility with hardware configuration.
Dynamic Model Compression to save resources and improve response time.	Section 4.6	Computationally intensive to do locally. May require Coordination between clients and broker to exchange information on, e.g., model architectures.
Dynamic Orchestration to optimize use of resources.	Section 4.7	Requires novel, highly distributed, efficient, and secure resource orchestration methods.
AIOps/MLOps and Monitoring to enhance responsiveness to anomalies and minimize downtime.	Section 4.8	Computationally intensive to do locally. Efficient and dynamic prioritization between monitoring and regular tasks.

brokers can facilitate real-time anomaly detection and predictive maintenance, ensuring that GenAI workloads remain resilient and secure.

Finally, tracking GenAI model performance can be effectively enhanced through integration CDM with IBM MQ, which has robust monitoring and tracing capabilities [57]. Solace PubSub could potentially assist in identifying and resolving issues related to message routing, delivery, and processing [118]. Integrating CDM can further enhance these functionalities, facilitating control and modeling activities in MLOps environments.

However, frameworks such as MLOps and CDM with are resource-hungry [219]. Integrating them with message brokers requires careful consideration of, for example, the computation capacity available locally, as well as the distribution of the related tasks, to avoid the starvation of regular operations. This requires striking an optimal balance between monitoring and regular tasks to gain performance optimization, lower cost, and energy efficiency. Additionally, distributed architectures and parallel processing are essential to manage high-intensity tasks effectively and in a timely manner.

4.9 Summary of message broker enhancement methods

While we have thoroughly explored the possible interplay between existing message broker technologies and their specific features to meet GenAI requirements, it is important to emphasize that our assessment of the suitability of a certain technology for specific tasks is indicative of their potential in the given context. Selecting any technology must be informed by a comprehensive analysis of the application and infrastructure topology requirements, data handling needs, and the particular features of these tools that align with the envisioned objectives. Adopting this approach can guarantee that the chosen solution not only fulfills immediate operational demands but also possesses the necessary scalability and flexibility for future growth and increased complexity. This consideration is crucial as we move towards the conclusion of our discussion, underscoring the importance of strategic technology selection in the dynamic landscape of GenAI-enhanced communication systems. The challenges, opportunities, and our strategic view on utilizing these technologies, along with the related subsections, are summarized in Table 7.

4.10 Sustainability Considerations for GenAI in Message Broker Systems

Before concluding this paper, it is important to discuss a critical and increasingly visible concern: the environmental impact of GenAI. The training of LLMs, which serve as the foundation for many GenAI systems, demands significant computational resources and energy. For example, the training of Meta's LLaMA models required over two thousand GPUs running for several months, consuming an estimated 2.6 million kWh of electricity and emitting more than 1,000 tonnes of CO_2 equivalent—comparable to the annual footprint of dozens of individuals [220]. Such figures underscore the substantial carbon footprint associated with foundation model development, particularly as models scale toward hundreds of billions of parameters [221].

In light of this, the integration of GenAI into message broker systems should not disregard sustainability. Instead of relying solely on large, general-purpose models, future research and system design should increasingly consider smaller, task-specific language models that are fine-tuned opportunistically for dedicated broker functionalities—such as semantic topic matching, prioritization, anomaly detection, or adaptive routing [222, 223]. These lightweight models require fewer resources to train and deploy [215], but can also offer faster inference and reduced latency, which is particularly important in real-time messaging infrastructures.

Complementing this model-level optimization, message brokers themselves can play an instrumental role in orchestrating GenAI workloads more sustainably. As intermediaries in distributed systems, brokers are well-positioned to support resource-aware scheduling, energy-efficient routing, and selective activation of GenAI modules. For instance, brokers could choose when to trigger a lightweight local model—for example, a specialized LLM fine-tuned for semantic topic classification or anomaly detection—versus delegating more complex or ambiguous tasks to a centralized, general-purpose LLM. This selective invocation and dynamic inference offloading can reduce redundant computation and help ensure that high-energy GenAI operations are reserved for the most impactful use cases, such as multi-modal reasoning or open-ended instruction following.

Such strategies align with broader trends toward energy transparency and sustainability in AI, where leading organizations have begun reporting the environmental footprint of their models and advocating for lifecycle-aware metrics [224–226].

In summary, addressing sustainability in GenAI-enabled message brokering systems requires both architectural and operational awareness. On one hand, deploying smaller, specialized models tailored to broker-specific tasks can help reduce training and inference costs. On the other, intelligent brokers can act as orchestrators of sustainable AI usage, ensuring that GenAI resources are leveraged effectively, efficiently, and responsibly.

5 Conclusion

In this paper, we have provided a comprehensive overview of contemporary message brokers, delineating their features, capabilities, and limitations with an eye toward their application within GenAI frameworks. Our analysis spanned a broad spectrum of criteria and we delved into the inherent limitations of existing message brokers when confronted with the demands of GenAI applications, prompting a reflection on the essential attributes of an ideal message broker framework designed to seamlessly integrate with GenAI technologies. In addressing these challenges, we analyzed several requirements to be satisfied in order to bolstering the efficacy of message brokers in facilitating the rapid evolution and deployment of GenAI applications.

Through a comprehensive analysis of the current state, challenges, and forward-looking strategies for message brokers, this study lays the groundwork for the development of more adaptable and efficient GenAI-enabled communication systems. Such systems are envisioned to not only distribute data with increasing efficiency but also to ensure the delivery of high-quality service, manage resources with greater intelligence, and satisfy the increasing demands of GenAI applications.

Finally, our exploration underscores the critical need for message brokers to evolve in tandem with technological advancements and GenAI requirements. By identifying opportunities for improvement, this paper aims to boost further research and development efforts focused on creating message broker frameworks that are not only robust and scalable but also closely aligned to the peculiarities of GenAI-driven data communication.

Acknowledgement

This research is supported by the Research Council of Finland (former Academy of Finland) 6G Flagship Program (Grant Number: 346208), and by Business Finland through the Neural pub/sub research project (diary number 8754/31/2022) and the Digital Twinning of Personal Area Networks for Optimized Sensing and Communication research project (diary number 8782/31/2022).

References

[1] OpenAI. ChatGPT. https://chat.openai.com/. ([n. d.]). Last accessed: June 4, 2025.

[2] Yifei Shen, Jiawei Shao, Xinjie Zhang, Zehong Lin, Hao Pan, Dongsheng Li, Jun Zhang, and Khaled B Letaief. 2024. Large language models empowered autonomous edge ai for connected intelligence. *IEEE Communications Magazine* (2024).

[3] Sasu Tarkoma, Roberto Morabito, and Jaakko Sauvola. 2023. AI-native Interconnect Framework for Integration of Large Language Model Technologies in 6G Systems. *arXiv preprint arXiv:2311.05842* (2023).

[4] Lina Bariah, Qiyang Zhao, Hang Zou, Yu Tian, Faouzi Bader, and Merouane Debbah. 2023. Large Language Models for Telecom: The Next Big Thing? *arXiv preprint arXiv:2306.10249* (2023).

[5] Henna Kokkonen, Lauri Lovén, Naser Hossein Motlagh, Abhishek Kumar, Juha Partala, Tri Nguyen, Víctor Casamayor Pujol, Panos Kostakos, Teemu Leppänen, Alfonso González-Gil, et al. 2022. Autonomy and intelligence in the computing continuum: Challenges, enablers, and future directions for orchestration. *arXiv preprint arXiv:2205.01423* (2022).

[6] Naser Hossein Motlagh, Lauri Lovén, Jacky Cao, Xiaoli Liu, Petteri Nurmi, Schahram Dustdar, Sasu Tarkoma, and Xiang Su. 2022. Edge computing: The computing infrastructure for the smart megacities of the future. *Computer* 55, 12 (2022), 54–64.

[7] Yun-Cheng Wang, Jintang Xue, Chengwei Wei, and C-C Jay Kuo. 2023. An Overview on Generative AI at Scale with Edge-Cloud Computing. (2023).

[8] Schahram Dustdar, Victor Casamayor Pujol, and Praveen Kumar Donta. 2022. On distributed computing continuum systems. *IEEE Transactions on Knowledge and Data Engineering* 35, 4 (2022), 4092–4105.

[9] Donta Praveen Kumar, Murturi Ilir, Casamayor Pujol Victor, Sedlak Boris, and Dustdar Schahram. 2023. Exploring the Potential of Distributed Computing Continuum Systems. *Computers* (2023).

[10] Fulya Ozturk and Ayse Meliha Ozdemir. 2019. Content-Based Publish/Subscribe Communication Model between IoT Devices in Smart City Environment. In *2019 7th International Istanbul Smart Grids and Cities Congress and Fair (ICSG)*. 189–193. DOI: http://dx.doi.org/10.1109/SGCF.2019.8782370

[11] Rakshit Wadhwa, Apurv Mehra, Pushpendra Singh, and Meenu Singh. 2015. A pub/sub based architecture to support public healthcare data exchange. In *2015 7th International Conference on Communication Systems and Networks (COMSNETS)*. 1–6. DOI: http://dx.doi.org/10.1109/COMSNETS.2015.7098706

[12] Patrick Th Eugster, Pascal A Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. 2003. The many faces of publish/subscribe. *ACM computing surveys (CSUR)* 35, 2 (2003), 114–131.

[13] Sasu Tarkoma. 2012. *Publish/subscribe systems: design and principles.* John Wiley & Sons.

[14] Praveen Kumar Donta, Satish Narayana Srirama, Tarachand Amgoth, and Chandra Sekhara Rao Annavarapu. 2022. Survey on recent advances in IoT application layer protocols and machine learning scope for research directions. *Digital Communications and Networks* 8, 5 (2022), 727–744.

[15] Jonathan Hasenburg, Florian Stanek, Florian Tschorsch, and David Bermbach. 2020. Managing latency and excess data dissemination in fog-based publish/subscribe systems. In *2020 IEEE international conference on fog computing (ICFC)*. IEEE, 9–16.

[16] Alessandro EC Redondi, Andrés Arcia-Moret, and Pietro Manzoni. 2019. Towards a scaled IoT pub/sub architecture for 5G networks: The case of multiaccess edge computing. In *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*. IEEE, 436–441.

[17] Lauri Lovén, Roberto Morabito, Abhishek Kumar, Susanna Pirttikangas, Jukka Riekki, and Sasu Tarkoma. 2023. How Can AI be Distributed in the Computing Continuum? Introducing the Neural Pub/Sub Paradigm. *arXiv preprint arXiv:2309.02058* (2023).

[18] Fatima Zahra Chafi, Youssef Fakhri, and Fatima Zahrae Ait Hamou Aadi. 2022. Introduction to Internet of Things' Communication Protocols. In *Advanced Intelligent Systems for Sustainable Development (AI2SD'2020) Volume 2*. Springer, 142–150.

[19] Vittorio Maniezzo, Marco A Boschetti, and Pietro Manzoni. 2023. Self-adaptive Publish/Subscribe Network Design. In *Metaheuristics: 14th International Conference, MIC 2022, Syracuse, Italy, July 11–14, 2022, Proceedings*. Springer, 478–484.

[20] Filipa Pedrosa and Luís Rodrigues. 2021. Reducing the subscription latency in reliable causal publish-subscribe systems. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing*. 203–212.

[21] Vineet John and Xia Liu. 2017. A survey of distributed message broker queues. *arXiv preprint arXiv:1704.00411* (2017).

[22] David S Linthicum. 2000. *Enterprise application integration.* Addison-Wesley Professional.
[23] Mark Perry, Christophe Delporte, Federico Demi, Animesh Ghosh, and Marc Luong. 2001. *MQSeries publish/subscribe applications.* IBM Redbooks.
[24] Gregor Hohpe and Bobby Woolf. 2004. *Enterprise integration patterns: Designing, building, and deploying messaging solutions.* Addison-Wesley Professional.
[25] Kasun Indrasiri and Sriskandarajah Suhothayan. 2021. *Design Patterns for Cloud Native Applications.* " O'Reilly Media, Inc.".
[26] Oleg Iakushkin and Valery Grishkin. 2014. Messaging middleware for cloud applications: Extending brokerless approach. In *2014 2nd 2014 2nd International Conference on Emission Electronics (ICEE).* IEEE, 1–4.
[27] The Apache Software Foundation. Apache ActiveMQ. https://activemq.apache.org/. ([n. d.]). Last accessed: June 4, 2025.
[28] Red Hat. Fuse Message Broker. https://access.redhat.com/taxonomy/products/fuse-message-broker. ([n. d.]). Last accessed: June 4, 2025.
[29] Apache Software Foundation. Apache Qpid. https://qpid.apache.org/. ([n. d.]). Last accessed: June 4, 2025.
[30] Rabbit Technologies. RabbitMQ. https://www.rabbitmq.com/. ([n. d.]). Last accessed: June 4, 2025.
[31] Red Hat. HornetQ. https://hornetq.jboss.org/. ([n. d.]). Last accessed: June 4, 2025.
[32] Inc Red Hat. Red Hat AMQ. https://www.redhat.com/en/technologies/jboss-middleware/amq. ([n. d.]). Last accessed: June 4, 2025.
[33] Celery software. Celery. https://docs.celeryq.dev/. ([n. d.]). Last accessed: June 4, 2025.
[34] Red Hat. JBoss Messaging. https://jbossmessaging.jboss.org/. ([n. d.]). Last accessed: June 4, 2025.
[35] Oracle. OpenMQ. https://javaee.github.io/openmq/. ([n. d.]). Last accessed: June 4, 2025.
[36] Inc Philotic. Beanstalk. https://beanstalkd.github.io/. ([n. d.]). Last accessed: June 4, 2025.
[37] Gearman. Gearman. http://gearman.org/. ([n. d.]). Last accessed: June 4, 2025.
[38] Mavimax. Enduro/X. https://www.endurox.org/. ([n. d.]). Last accessed: June 4, 2025.
[39] Paul Fremantle. WSO2. https://wso2.com/. ([n. d.]). Last accessed: June 4, 2025.
[40] HiveMQ. HiveMQ. https://www.hivemq.com/. ([n. d.]). Last accessed: June 4, 2025.
[41] Redis Labs. Redis. https://redis.io/. ([n. d.]). Last accessed: June 4, 2025.
[42] EMQX Technologies. EMQX. https://www.emqx.io/. ([n. d.]). Last accessed: June 4, 2025.
[43] The Apache Software Foundation. Apache Pulsar. https://pulsar.apache.org/. ([n. d.]). Last accessed: June 4, 2025.
[44] Apache Software Foundation. Apache Kafka. https://kafka.apache.org/. ([n. d.]). Last accessed: June 4, 2025.
[45] Alibaba Group. Apache RocketMQ. https://rocketmq.apache.org/. ([n. d.]). Last accessed: June 4, 2025.
[46] Eclipse. Eclipse Mosquitto. https://mosquitto.org/. ([n. d.]). Last accessed: June 4, 2025.
[47] iMatix. Zero MQ. https://zeromq.org/. ([n. d.]). Last accessed: June 4, 2025.
[48] Inc Onyara. Apache NiFi. https://nifi.apache.org/. ([n. d.]). Last accessed: June 4, 2025.
[49] ABLY REALTIME LTD. Ably Realtime. https://ably.com/. ([n. d.]). Last accessed: June 4, 2025.
[50] Apache Software Foundation. Apache SamZa. https://samza.apache.org/. ([n. d.]). Last accessed: June 4, 2025.
[51] VerneMQ. VerneMQ. https://vernemq.com/. ([n. d.]). Last accessed: June 4, 2025.
[52] Particular Software. NServiceBus. https://particular.net/nservicebus. ([n. d.]). Last accessed: June 4, 2025.
[53] Twitter. kestrel. https://github.com/twitter-archive/kestrel. ([n. d.]). Last accessed: June 4, 2025.
[54] Bitly. NSQ. https://nsq.io/. ([n. d.]). Last accessed: June 4, 2025.
[55] Synadia Communications. NATS. https://nats.io/. ([n. d.]). Last accessed: June 4, 2025.
[56] KubeMQ. KubeMQ. https://kubemq.io/. ([n. d.]). Last accessed: June 4, 2025.
[57] IBM. IBM MQ. https://www.ibm.com/docs/en/ibm-mq. ([n. d.]). Last accessed: June 4, 2025.
[58] Amazon Web Services (AWS). Amazon SQS. https://aws.amazon.com/sqs/. ([n. d.]). Last accessed: June 4, 2025.
[59] Microsoft. MSMQ. https://learn.microsoft.com/en-us/previous-versions/windows/desktop/msmq/. ([n. d.]). Last accessed: June 4, 2025.
[60] Oracle. Oracle Message Broker. https://docs.oracle.com/cd/E26576_01/doc.312/e24948.pdf. ([n. d.]). Last accessed: June 4, 2025.
[61] TIBCO. TIBCO Rendezvous. https://www.tibco.com/products/tibco-rendezvous. ([n. d.]). Last accessed: June 4, 2025.
[62] TIBCO. TIBCO Enterprise Message Service™. https://www.tibco.com/products/tibco-enterprise-message-service. ([n. d.]). Last accessed: June 4, 2025.
[63] MuleSoft. Anypoint MQ. https://docs.mulesoft.com/mq/. ([n. d.]). Last accessed: June 4, 2025.
[64] Microsoft. Azure Service Bus. https://learn.microsoft.com/en-us/azure/service-bus-messaging/. ([n. d.]). Last accessed: June 4, 2025.
[65] SAP AG. SAP NetWeaver Process Integration. https://help.sap.com/docs/. ([n. d.]). Last accessed: June 4, 2025.
[66] Craig Betts. Solace. https://solace.com/. ([n. d.]). Last accessed: June 4, 2025.
[67] Google. Google Cloud Pub/Sub. https://cloud.google.com/pubsub. ([n. d.]). Last accessed: June 4, 2025.
[68] Amazon Web Services (AWS). Amazon MQ. https://aws.amazon.com/amazon-mq/. ([n. d.]). Last accessed: June 4, 2025.
[69] Intel. Intel MPI Library. https://www.intel.com/content/www/us/en/developer/tools/oneapi/mpi-library.html. ([n. d.]). Last accessed: June 4, 2025.
[70] Amazon. Kinesis. https://aws.amazon.com/kinesis/. ([n. d.]). Last accessed: June 4, 2025.

[71] Microsoft. Azure Storage Queue. https://learn.microsoft.com/en-us/azure/storage/queues/. ([n. d.]). Last accessed: June 4, 2025.
[72] Iron.io. IronMQ. http://www.iron.io/mq. ([n. d.]). Last accessed: June 4, 2025.
[73] Red Hat. Fuse Message Broker. https://docs.huihoo.com/fuse/getting_started.pdf. ([n. d.]). Last accessed: June 4, 2025.
[74] Apache Software Foundation. Apache Qpid. https://people.apache.org/~jonathan/Programming-In-Apache-Qpid.html. ([n. d.]). Last accessed: June 4, 2025.
[75] Rabbit Technologies. RabbitMQ. https://blog.rabbitmq.com/posts/2020/07/disaster-recovery-and-high-availability-101/. ([n. d.]). Last accessed: June 4, 2025.
[76] Red Hat. HornetQ. https://hornetq.sourceforge.net/docs/hornetq-2.1.2.Final/user-manual/en/html_single/index.html. ([n. d.]). Last accessed: June 4, 2025.
[77] Red Hat. HornetQ. https://docs.jboss.org/hornetq/2.4.0.Final/docs/user-manual/html/interoperability.html. ([n. d.]). Last accessed: June 4, 2025.
[78] Inc Red Hat. Red Hat AMQ. https://access.redhat.com/documentation/. ([n. d.]). Last accessed: June 4, 2025.
[79] Red Hat. JBoss Messaging. https://docs.jboss.org/jbossmessaging/docs/usermanual-2.0.0.beta1/html. ([n. d.]). Last accessed: June 4, 2025.
[80] Inc Philotic. Beanstalk. https://raw.githubusercontent.com/beanstalkd/beanstalkd/master/doc/protocol.txt. ([n. d.]). Last accessed: June 4, 2025.
[81] Inc Philotic. Beanstalk. https://github.com/beanstalkd/beanstalkd/wiki/Client-Libraries. ([n. d.]). Last accessed: June 4, 2025.
[82] Mavimax. Enduro/X. https://github.com/endurox-dev/endurox/blob/master/doc/. ([n. d.]). Last accessed: June 4, 2025.
[83] Paul Fremantle. WSO2. https://ei.docs.wso2.com/en/latest/micro-integrator/setup/performance_tuning/tuning_jvm_performance/. ([n. d.]). Last accessed: June 4, 2025.
[84] Paul Fremantle. WSO2. https://apim.docs.wso2.com/en/4.1.0/install-and-setup/setup/mi-setup/transport_configurations/configuring-transports/. ([n. d.]). Last accessed: June 4, 2025.
[85] Paul Fremantle. WSO2. https://ei.docs.wso2.com/en/latest/micro-integrator/references/mediators/filter-Mediator/. ([n. d.]). Last accessed: June 4, 2025.
[86] HiveMQ. HiveMQ. https://docs.hivemq.com/hivemq/. ([n. d.]). Last accessed: June 4, 2025.
[87] Redis Labs. Redis. https://docs.redis.com/latest/rs/security/. ([n. d.]). Last accessed: June 4, 2025.
[88] EMQX Technologies. EMQX. https://www.emqx.com/en/blog/emqx-vs-mosquitto-2023-mqtt-broker-comparison. ([n. d.]). Last accessed: June 4, 2025.
[89] Eclipse. Eclipse Mosquitto. https://www.emqx.com/en/blog/mosquitto-mqtt-broker-pros-cons-tutorial-and-modern-alternatives. ([n. d.]). Last accessed: June 4, 2025.
[90] Eclipse. Eclipse Mosquitto. https://projects.eclipse.org/projects/iot.mosquitto. ([n. d.]). Last accessed: June 4, 2025.
[91] iMatix. Zero MQ. https://www.hivemq.com/article/mqtt-vs-zeromq-for-iot/. ([n. d.]). Last accessed: June 4, 2025.
[92] iMatix. Zero MQ. http://wiki.zeromq.org/area:faq. ([n. d.]). Last accessed: June 4, 2025.
[93] Apache Software Foundation. Apache SamZa. https://samza.incubator.apache.org/learn/documentation/0.7.0/comparisons/storm.html. ([n. d.]). Last accessed: June 4, 2025.
[94] Martin Kleppmann. 2019. Apache Samza. (2019).
[95] EMQX Technologies. EMQX. https://www.emqx.com/en/blog/emqx-vs-vernemq-2023-mqtt-broker-comparison. ([n. d.]). Last accessed: June 4, 2025.
[96] Particular Software. NServiceBus. https://docs.particular.net/nservicebus/. ([n. d.]). Last accessed: June 4, 2025.
[97] Particular Software. NServiceBus. https://docs.particular.net/tutorials/monitoring-setup/. ([n. d.]). Last accessed: June 4, 2025.
[98] Twitter. kestrel. https://github.com/memcached/memcached/blob/master/doc/protocol.txt. ([n. d.]). Last accessed: June 4, 2025.
[99] Synadia Communications. NATS. https://docs.nats.io/. ([n. d.]). Last accessed: June 4, 2025.
[100] Berk Ayaz, Nina Slamnik-Kriještorac, and Johann Marquez-Barja. 2022. Data Management Platform For Smart Orchestration of Decentralized and Heterogeneous Vehicular Edge Networks. In *Proceedings of the 2022 ACM Conference on Information Technology for Social Good*. 118–124.
[101] IBM. IBM MQ. https://cloud.ibm.com/catalog/services/mq. ([n. d.]). Last accessed: June 4, 2025.
[102] Iron.io. IronMQ. https://blog.iron.io/ibm-mq-vs-ironmq-pros-cons-and-choosing-an-mq/#4. ([n. d.]). Last accessed: June 4, 2025.
[103] IBM. IBM MQ. https://www.ibm.com/. ([n. d.]). Last accessed: June 4, 2025.
[104] Amazon Web Services (AWS). Amazon SQS. https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/quotas-messages.html. ([n. d.]). Last accessed: June 4, 2025.
[105] Microsoft. MSMQ. https://techcommunity.microsoft.com/t5/skype-for-business-blog/troubleshooting-microsoft-message-queuing-issues-on-microsoft/ba-p/619639. ([n. d.]). Last accessed: June 4, 2025.
[106] Oracle. Oracle Message Broker. https://docs.oracle.com/cd/. ([n. d.]). Last accessed: June 4, 2025.
[107] TIBCO. TIBCO Enterprise Message Service™. https://docs.tibco.com/pub/ems/. ([n. d.]). Last accessed: June 4, 2025.

[108] TIBCO. TIBCO Enterprise Message Service™. https://support.tibco.com/s/article/Tibco-KnowledgeArticle-Article-22588. ([n. d.]). Last accessed: June 4, 2025.

[109] TIBCO. TIBCO Rendezvous. https://www.tibco.com/products/tibco-cloud-events/pricing-plans. ([n. d.]). Last accessed: June 4, 2025.

[110] TIBCO. TIBCO Rendezvous. https://docs.tibco.com/pub/ems/8.6.0/doc/html/GUID-66774B42-2A5F-4221-864E-3331622E1091.html. ([n. d.]). Last accessed: June 4, 2025.

[111] TIBCO. TIBCO Rendezvous. https://docs.tibco.com/pub/rendezvous/. ([n. d.]). Last accessed: June 4, 2025.

[112] MuleSoft. Anypoint MQ. https://www.mulesoft.com/. ([n. d.]). Last accessed: June 4, 2025.

[113] Microsoft. Azure Storage Queue. https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-azure-and-service-bus-queues-compared-contrasted. ([n. d.]). Last accessed: June 4, 2025.

[114] Microsoft. Azure Service Bus. https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebussender.schedulemessageasync?view=azure-dotnet. ([n. d.]). Last accessed: June 4, 2025.

[115] Microsoft. Azure Service Bus. https://azure.microsoft.com/en-us/products/service-bus. ([n. d.]). Last accessed: June 4, 2025.

[116] SAP AG. SAP NetWeaver Process Integration. https://blogs.sap.com/2014/04/02/message-size-as-source-of-performance-bottleneck/. ([n. d.]). Last accessed: June 4, 2025.

[117] Craig Betts. Solace. https://docs.solace.com/. ([n. d.]). Last accessed: June 4, 2025.

[118] Craig Betts. Solace. https://www.solace.dev/. ([n. d.]). Last accessed: June 4, 2025.

[119] Google. Google Cloud Pub/Sub. https://cloudplatform.googleblog.com/2015/04/big-data-cloud-way.html. ([n. d.]). Last accessed: June 4, 2025.

[120] Amazon Web Services (AWS). Amazon MQ. https://docs.aws.amazon.com/amazon-mq/latest/. ([n. d.]). Last accessed: June 4, 2025.

[121] Intel. Intel MPI Library. https://texas.gs.shi.com/product/32703496/Intel-MPI-Library-for-Windows. ([n. d.]). Last accessed: June 4, 2025.

[122] Intel. Intel MPI Library. https://www.intel.com/content/www/us/en/developer/articles/technical/improve-performance-and-stability-with-intel-mpi-library-on-infiniband.html. ([n. d.]). Last accessed: June 4, 2025.

[123] Amazon. Kinesis. https://repost.aws/knowledge-center/troubleshoot-kinesis-agent-linux. ([n. d.]). Last accessed: June 4, 2025.

[124] Amazon. Kinesis. https://docs.aws.amazon.com/streams/latest/. ([n. d.]). Last accessed: June 4, 2025.

[125] Amazon. Kinesis. https://docs.aws.amazon.com/kinesisanalytics/latest/dev/error-handling.html. ([n. d.]). Last accessed: June 4, 2025.

[126] Iron.io. IronMQ. https://try.iron.io/pricing-worker-monthly/. ([n. d.]). Last accessed: June 4, 2025.

[127] Iron.io. IronMQ. https://blog.iron.io/apache-kafka-vs-ironmq-whats-best-for-your-business/. ([n. d.]). Last accessed: June 4, 2025.

[128] Oracle. Open Message Queue, Administration Guide, Release 5.0. https://javaee.github.io/glassfish/doc/4.0/mq-admin-guide.pdf. ([n. d.]). 2013.

[129] Oracle. Oracle Message Broker. https://docs.oracle.com/cd/E19316-01/820-6424/aerbz/index.html. ([n. d.]). Last accessed: June 4, 2025.

[130] Oracle. Oracle Message Broker. https://docs.oracle.com/cd/E19879-01/821-0028/aercs/index.html. ([n. d.]). Last accessed: June 4, 2025.

[131] Inc Red Hat. Red Hat AMQ. https://access.redhat.com/documentation/en-us/red_hat_amq/6.1/html/product_introduction/fmbscalable. ([n. d.]). Last accessed: June 4, 2025.

[132] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[133] Francisco García-Peñalvo and Andrea Vázquez-Ingelmo. 2023. What do we mean by GenAI? A systematic mapping of the evolution, trends, and techniques involved in Generative AI. (2023).

[134] Maanak Gupta, Charankumar Akiri, Kshitiz Aryal, Eli Parker, and Lopamudra Praharaj. 2023. From ChatGPT to ThreatGPT: Impact of Generative AI in Cybersecurity and Privacy. *IEEE Access* 11 (2023), 80218–80245. DOI: http://dx.doi.org/10.1109/ACCESS.2023.3300381

[135] Khen Bo Kan, Hyunsu Mun, Guohong Cao, and Youngseok Lee. 2024. Mobile-llama: Instruction fine-tuning open-source llm for network analysis in 5g networks. *IEEE Network* (2024).

[136] Amirhossein Ghaffari, Huong Nguyen, Alaa Saleh, Lauri Lovén, and Ekaterina Gilman. 2024. Traffic Accident Prediction and Warning System: Integration Use Case. In *Fourth Workshop on Knowledge-infused Learning (KIL 2024)*. OpenReview.

[137] Othmane Friha, Mohamed Amine Ferrag, Burak Kantarci, Burak Cakmak, Arda Ozgun, and Nassira Ghoualmi-Zine. 2024. Llm-based edge intelligence: A comprehensive survey on architectures, applications, security and trustworthiness. *IEEE Open Journal of the Communications Society* (2024).

[138] Yudong Huang, Hongyang Du, Xinyuan Zhang, Dusit Niyato, Jiawen Kang, Zehui Xiong, Shuo Wang, and Tao Huang. 2024. Large language models for networking: Applications, enabling techniques, and challenges. *IEEE Network* (2024).

[139] Sifan Long, Jingjing Tan, Bomin Mao, Fengxiao Tang, Yangfan Li, Ming Zhao, and Nei Kato. 2025. A Survey on Intelligent Network Operations and Performance Optimization Based on Large Language Models. *IEEE Communications Surveys & Tutorials* (2025).

[140] Shengzhe Xu, Christo Kurisummoottil Thomas, Omar Hashash, Nikhil Muralidhar, Walid Saad, and Naren Ramakrishnan. 2024. Large Multi-Modal Models (LMMs) as Universal Foundation Models for AI-Native Wireless Systems. *arXiv preprint arXiv:2402.01748* (2024).

[141] Tao Gong, Chengqi Lyu, Shilong Zhang, Yudong Wang, Miao Zheng, Qian Zhao, Kuikun Liu, Wenwei Zhang, Ping Luo, and Kai Chen. 2023. Multimodal-gpt: A vision and language model for dialogue with humans. *arXiv preprint arXiv:2305.04790* (2023).

[142] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. 2022. Flamingo: a visual language model for few-shot learning. *Advances in Neural Information Processing Systems* 35 (2022), 23716–23736.

[143] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface. *arXiv preprint arXiv:2303.17580* (2023).

[144] Rongjie Huang, Mingze Li, Dongchao Yang, Jiatong Shi, Xuankai Chang, Zhenhui Ye, Yuning Wu, Zhiqing Hong, Jiawei Huang, Jinglin Liu, et al. 2023. Audiogpt: Understanding and generating speech, music, sound, and talking head. *arXiv preprint arXiv:2304.12995* (2023).

[145] OpenAI. 2023. GPT-4 Technical Report. (2023). arXiv:cs.CL/2303.08774

[146] Chenfei Wu, Shengming Yin, Weizhen Qi, Xiaodong Wang, Zecheng Tang, and Nan Duan. 2023. Visual chatgpt: Talking, drawing and editing with visual foundation models. *arXiv preprint arXiv:2303.04671* (2023).

[147] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. 2023. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864* (2023).

[148] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2023. A survey on large language model based autonomous agents. *arXiv preprint arXiv:2308.11432* (2023).

[149] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems* 35 (2022), 24824–24837.

[150] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629* (2022).

[151] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*.

[152] openai. ChatGPT. https://chat.openai.com/. ([n. d.]). Last accessed: June 4, 2025.

[153] Dhruv Shah, Błażej Osiński, Sergey Levine, et al. 2023. Lm-nav: Robotic navigation with large pre-trained models of language, vision, and action. In *Conference on Robot Learning*. PMLR, 492–504.

[154] Yao Mu, Qinglong Zhang, Mengkang Hu, Wenhai Wang, Mingyu Ding, Jun Jin, Bin Wang, Jifeng Dai, Yu Qiao, and Ping Luo. 2023. Embodiedgpt: Vision-language pre-training via embodied chain of thought. *arXiv preprint arXiv:2305.15021* (2023).

[155] Cheng Qian, Chi Han, Yi R Fung, Yujia Qin, Zhiyuan Liu, and Heng Ji. 2023. CREATOR: Disentangling Abstract and Concrete Reasonings of Large Language Models through Tool Creation. *arXiv preprint arXiv:2305.14318* (2023).

[156] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2023. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128* (2023).

[157] Jihong Park, Sumudu Samarakoon, Mehdi Bennis, and Mérouane Debbah. 2019. Wireless network intelligence at the edge. *Proc. IEEE* 107, 11 (2019), 2204–2239.

[158] Lauri Lovén, Teemu Leppänen, Ella Peltonen, Juha Partala, Erkki Harjula, Pawani Porambage, Mika Ylianttila, and Jukka Riekki. 2019. EdgeAI: A vision for distributed, edge-native artificial intelligence in future 6G networks. *6G Wireless Summit, March 24-26, 2019 Levi, Finland* (2019).

[159] Shuiguang Deng, Hailiang Zhao, Weijia Fang, Jianwei Yin, Schahram Dustdar, and Albert Y Zomaya. 2020. Edge intelligence: The confluence of edge computing and artificial intelligence. *IEEE Internet of Things Journal* 7, 8 (2020), 7457–7469.

[160] Philipp Schoenegger, Peter S Park, Ezra Karger, Sean Trott, and Philip E Tetlock. 2024. Ai-augmented predictions: Llm assistants improve human forecasting accuracy. *ACM Transactions on Interactive Intelligent Systems* (2024).

[161] Laifa Tao, Haifei Liu, Guoao Ning, Wenyan Cao, Bohao Huang, and Chen Lu. 2025. LLM-based framework for bearing fault diagnosis. *Mechanical Systems and Signal Processing* 224 (2025), 112127.

[162] Adewumi Emmanuel Ojuolape and Shanfeng Hu. 2024. Explainable Fault Diagnosis of Control Systems Using Large Language Models. In *2024 IEEE Conference on Control Technology and Applications (CCTA)*. IEEE, 491–498.

[163] Thomas KF Chiu. 2023. The impact of Generative AI (GenAI) on practices, policies and research direction in education: A case of ChatGPT and Midjourney. *Interactive Learning Environments* (2023), 1–17.

[164] OpenAI. GPT-4 System Card. https://cdn.openai.com/papers/gpt-4-system-card.pdf. ([n. d.]). Last accessed: June 4, 2025.

[165] Abdulkadir Celik and Ahmed M Eltawil. 2024. At the Dawn of Generative AI Era: A Tutorial-cum-Survey On New Frontiers in 6G Wireless Intelligence. *IEEE Open Journal of the Communications Society* (2024).

[166] Aishwarya Vijayan. 2023. A prompt engineering approach for structured data extraction from unstructured text using conversational llms. In *Proceedings of the 2023 6th International Conference on Algorithms, Computing and Artificial Intelligence*. 183–189.

[167] Jeevana Priya Inala, Chenglong Wang, Steven Drucker, Gonzalo Ramos, Victor Dibia, Nathalie Riche, Dave Brown, Dan Marshall, and Jianfeng Gao. 2024. Data Analysis in the Era of Generative AI. *arXiv preprint arXiv:2409.18475* (2024).

[168] Ming Jin, Qingsong Wen, Yuxuan Liang, Chaoli Zhang, Siqiao Xue, Xue Wang, James Zhang, Yi Wang, Haifeng Chen, Xiaoli Li, et al. 2023. Large models for time series and spatio-temporal data: A survey and outlook. *arXiv preprint arXiv:2310.10196* (2023).

[169] Ching Chang, Wei-Yao Wang, Wen-Chih Peng, and Tien-Fu Chen. 2025. LLM4TS: Aligning Pre-Trained LLMs as Data-Efficient Time-Series Forecasters. *ACM Trans. Intell. Syst. Technol.* (Feb. 2025). DOI: http://dx.doi.org/10.1145/3719207 Just Accepted.
[170] Juan Morales-García, Antonio Llanes, Francisco Arcas-Túnez, and Fernando Terroso-Sáenz. 2024. Developing Time Series Forecasting Models with Generative Large Language Models. *ACM Trans. Intell. Syst. Technol.* (May 2024). DOI: http://dx.doi.org/10.1145/3663485 Just Accepted.
[171] Xiyuan Zhang, Ranak Roy Chowdhury, Rajesh K Gupta, and Jingbo Shang. 2024. Large Language Models for Time Series: A Survey. *arXiv preprint arXiv:2402.01801* (2024).
[172] Hao Xue and Flora D Salim. 2023. Promptcast: A new prompt-based learning paradigm for time series forecasting. *IEEE Transactions on Knowledge and Data Engineering* (2023).
[173] Junhao Zheng, Shengjie Qiu, Chengming Shi, and Qianli Ma. 2025. Towards Lifelong Learning of Large Language Models: A Survey. *ACM Comput. Surv.* 57, 8, Article 193 (March 2025), 35 pages. DOI: http://dx.doi.org/10.1145/3716629
[174] Uday Kamath, Kevin Keenan, Garrett Somers, and Sarah Sorenson. 2024. Multimodal LLMs. In *Large Language Models: A Deep Dive: Bridging Theory and Practice*. Springer, 375–421.
[175] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.
[176] LangChain. 2025. LangChain web site. (2025). https://www.langchain.com Accessed: 2025-01-03.
[177] Talha Zeeshan, Abhishek Kumar, Susanna Pirttikangas, and Sasu Tarkoma. 2025. Large Language Model Based Multi-Agent System Augmented Complex Event Processing Pipeline for Internet of Multimedia Things. *arXiv preprint arXiv:2501.00906* (2025).
[178] Vipula Rawte, Amit Sheth, and Amitava Das. 2023. A survey of hallucination in large foundation models. *arXiv preprint arXiv:2309.05922* (2023).
[179] Junseong Bang, Byung-Tak Lee, and Pangun Park. 2023. Examination of Ethical Principles for LLM-Based Recommendations in Conversational AI. In *2023 International Conference on Platform Technology and Service (PlatCon)*. IEEE, 109–113.
[180] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. 2023. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems* (2023).
[181] Zixuan Zhou, Xuefei Ning, Ke Hong, Tianyu Fu, Jiaming Xu, Shiyao Li, Yuming Lou, Luning Wang, Zhihang Yuan, Xiuhong Li, et al. 2024. A survey on efficient inference for large language models. *arXiv preprint arXiv:2404.14294* (2024).
[182] Mengwei Xu, Dongqi Cai, Wangsong Yin, Shangguang Wang, Xin Jin, and Xuanzhe Liu. 2025. Resource-efficient Algorithms and Systems of Foundation Models: A Survey. *ACM Comput. Surv.* 57, 5, Article 110 (Jan. 2025), 39 pages. DOI: http://dx.doi.org/10.1145/3706418
[183] Mohammad Rubyet Islam. 2024. *Generative AI, Cybersecurity, and Ethics*. John Wiley & Sons.
[184] Nitin Naik. 2017. Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. In *2017 IEEE international systems engineering symposium (ISSE)*. IEEE, 1–7.
[185] Bonan Min, Hayley Ross, Elior Sulem, Amir Pouran Ben Veyseh, Thien Huu Nguyen, Oscar Sainz, Eneko Agirre, Ilana Heintz, and Dan Roth. 2023. Recent advances in natural language processing via large pre-trained language models: A survey. *Comput. Surveys* 56, 2 (2023), 1–40.
[186] Le Xia, Yao Sun, Chengsi Liang, Lei Zhang, Muhammad Ali Imran, and Dusit Niyato. 2023. Generative AI for semantic communication: Architecture, challenges, and outlook. *arXiv preprint arXiv:2308.15483* (2023).
[187] Peiwen Jiang, Chao-Kai Wen, Xinping Yi, Xiao Li, Shi Jin, and Jun Zhang. 2023. Semantic Communications using Foundation Models: Design Approaches and Open Issues. *arXiv preprint arXiv:2309.13315* (2023).
[188] Praveen Kumar Donta, Boris Sedlak, Victor Casamayor Pujol, and Schahram Dustdar. 2023. Governance and sustainability of distributed continuum systems: a big data approach. *Journal of Big Data* 10, 1 (2023), 1–31.
[189] David Oniani, Jordan Hilsman, Yifan Peng, Ronald K Poropatich, COL Pamplin, LTC Legault, Yanshan Wang, et al. 2023. From Military to Healthcare: Adopting and Expanding Ethical Principles for Generative Artificial Intelligence. *arXiv preprint arXiv:2308.02448* (2023).
[190] Shorouk Alaa El Din Talha. 2020. A Semantic Based Annotation Technique for the Internet of Things. In *2020 the 3rd International Conference on Computing and Big Data*. 42–47.
[191] John F Sowa. 2000. Ontology, metadata, and semiotics. In *International conference on conceptual structures*. Springer, 55–81.
[192] Jiachen Chen, Haoyuan Xu, Yanyong Zhang, and Dipankar Raychaudhuri. 2017. Graph-pubsub: An efficient pub/sub architecture with graph-based information relationship. In *Proceedings of the Fifth ACM/IEEE Workshop on Hot Topics in Web Systems and Technologies*. 1–6.
[193] Wanting Yang, Zi Qin Liew, Wei Yang Bryan Lim, Zehui Xiong, Dusit Niyato, Xuefen Chi, Xianbin Cao, and Khaled B Letaief. 2022. Semantic communication meets edge intelligence. *IEEE Wireless Communications* 29, 5 (2022), 28–35.
[194] Lei Zhang, Yuge Zhang, Kan Ren, Dongsheng Li, and Yuqing Yang. 2023. MLCopilot: Unleashing the Power of Large Language Models in Solving Machine Learning Tasks. *arXiv preprint arXiv:2304.14979* (2023).

[195] Yifei Shen, Jiawei Shao, Xinjie Zhang, Zehong Lin, Hao Pan, Dongsheng Li, Jun Zhang, and Khaled B Letaief. 2023. Large language models empowered autonomous edge AI for connected intelligence. *arXiv preprint arXiv:2307.02779* (2023).

[196] Yagmur Yigit, William J Buchanan, Madjid G Tehrani, and Leandros Maglaras. 2024. Review of generative ai methods in cybersecurity. *arXiv preprint arXiv:2403.08701* (2024).

[197] Shenggui Li, Hongxin Liu, Zhengda Bian, Jiarui Fang, Haichen Huang, Yuliang Liu, Boxiang Wang, and Yang You. 2023. Colossal-ai: A unified deep learning system for large-scale parallel training. In *Proceedings of the 52nd International Conference on Parallel Processing*. 766–775.

[198] Zeyu Zhang and Haiying Shen. 2024. CSPS: A Communication-Efficient Sequence-Parallelism based Serving System for Transformer based Models with Long Prompts. *arXiv preprint arXiv:2409.15104* (2024).

[199] Jiarui Fang and Shangchun Zhao. 2024. A Unified Sequence Parallelism Approach for Long Context Generative AI. *arXiv preprint arXiv:2405.07719* (2024).

[200] Vijay Anand Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. 2023. Reducing activation recomputation in large transformer models. *Proceedings of Machine Learning and Systems* 5 (2023).

[201] Venkatesh Balavadhani Parthasarathy, Ahtsham Zafar, Aafaq Khan, and Arsalan Shahid. 2024. The ultimate guide to fine-tuning llms from basics to breakthroughs: An exhaustive review of technologies, research, best practices, applied research challenges and opportunities. *arXiv preprint arXiv:2408.13296* (2024).

[202] Kazuki Fujii, Kohei Watanabe, and Rio Yokota. 2024. Accelerating Large Language Model Training with 4D Parallelism and Memory Consumption Estimator. *arXiv preprint arXiv:2411.06465* (2024).

[203] Jinhao Li, Shiyao Li, Jiaming Xu, Shan Huang, Yaoxiu Lian, Jun Liu, Yu Wang, and Guohao Dai. 2023. Enabling Fast 2-bit LLM on GPUs: Memory Alignment, Sparse Outlier, and Asynchronous Dequantization. *arXiv preprint arXiv:2311.16442* (2023).

[204] Fei Yang, Shuang Peng, Ning Sun, Fangyu Wang, Yuanyuan Wang, Fu Wu, Jiezhong Qiu, and Aimin Pan. 2024. Holmes: Towards distributed training across clusters with heterogeneous nic environment. In *Proceedings of the 53rd International Conference on Parallel Processing*. 514–523.

[205] Sai Krishna Revanth Vuruma, Ashley Margetts, Jianhai Su, Faez Ahmed, and Biplav Srivastava. 2024. From Cloud to Edge: Rethinking Generative AI for Low-Resource Design Challenges. *arXiv preprint arXiv:2402.12702* (2024).

[206] Fali Wang, Zhiwei Zhang, Xianren Zhang, Zongyu Wu, Tzuhao Mo, Qiuhao Lu, Wanjing Wang, Rui Li, Junjie Xu, Xianfeng Tang, et al. 2024. A comprehensive survey of small language models in the era of large language models: Techniques, enhancements, applications, collaboration with llms, and trustworthiness. *arXiv preprint arXiv:2411.03350* (2024).

[207] Savitha Viswanadh Kandala, Pramuka Medaranga, and Ambuj Varshney. 2024. TinyLLM: A Framework for Training and Deploying Language Models at the Edge Computers. *arXiv preprint arXiv:2412.15304* (2024).

[208] Yuang Jiang, Shiqiang Wang, Victor Valls, Bong Jun Ko, Wei-Han Lee, Kin K Leung, and Leandros Tassiulas. 2022. Model pruning enables efficient federated learning on edge devices. *IEEE Transactions on Neural Networks and Learning Systems* (2022).

[209] Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. 2023. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695* (2023).

[210] Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems* 36 (2023), 21702–21720.

[211] Chuanpeng Yang, Yao Zhu, Wang Lu, Yidong Wang, Qian Chen, Chenlong Gao, Bingjie Yan, and Yiqiang Chen. 2024. Survey on Knowledge Distillation for Large Language Models: Methods, Evaluation, and Application. *ACM Trans. Intell. Syst. Technol.* (Oct. 2024). DOI: http://dx.doi.org/10.1145/3699518 Just Accepted.

[212] Xunyu Zhu, Jian Li, Yong Liu, Can Ma, and Weiping Wang. 2023. A survey on model compression for large language models. *arXiv preprint arXiv:2308.07633* (2023).

[213] Antonio Polino, Razvan Pascanu, and Dan Alistarh. 2018. Model compression via distillation and quantization. *arXiv preprint arXiv:1802.05668* (2018).

[214] Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. 2023. Knowledge distillation of large language models. *arXiv preprint arXiv:2306.08543* (2023).

[215] Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alexander Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. 2023. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. *arXiv preprint arXiv:2305.02301* (2023).

[216] Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. 2023. Llm-qat: Data-free quantization aware training for large language models. *arXiv preprint arXiv:2305.17888* (2023).

[217] Yen-Chang Hsu, Ting Hua, Sungen Chang, Qian Lou, Yilin Shen, and Hongxia Jin. 2022. Language model compression with weighted low-rank factorization. *arXiv preprint arXiv:2207.00112* (2022).

[218] Arnav Chavan, Raghav Magazine, Shubham Kushwaha, Mérouane Debbah, and Deepak Gupta. 2024. Faster and Lighter LLMs: A Survey on Current Challenges and Way Forward. *arXiv preprint arXiv:2402.01799* (2024).

[219] Dominik Kreuzberger, Niklas Kühl, and Sebastian Hirschl. 2023. Machine learning operations (mlops): Overview, definition, and architecture. *IEEE access* 11 (2023), 31866–31879.
[220] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).
[221] Alexandra Sasha Luccioni, Sylvain Viguier, and Anne-Laure Ligozat. 2023. Estimating the carbon footprint of bloom, a 176b parameter language model. *Journal of Machine Learning Research* 24, 253 (2023), 1–15.
[222] Xiaoxi Li, Jiajie Jin, Yujia Zhou, Yuyao Zhang, Peitian Zhang, Yutao Zhu, and Zhicheng Dou. 2024. From matching to generation: A survey on generative information retrieval. *ACM Transactions on Information Systems* (2024).
[223] Qiong Wu, Zhaoxi Ke, Yiyi Zhou, Xiaoshuai Sun, and Rongrong Ji. 2024. Routing experts: Learning to route dynamic experts in multi-modal large language models. *arXiv preprint arXiv:2407.14093* (2024).
[224] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. 2021. Carbon emissions and large neural network training. *arXiv preprint arXiv:2104.10350* (2021).
[225] Jesse Dodge, Taylor Prewitt, Remi Tachet des Combes, Erika Odmark, Roy Schwartz, Emma Strubell, Alexandra Sasha Luccioni, Noah A Smith, Nicole DeCario, and Will Buchanan. 2022. Measuring the carbon intensity of AI in cloud instances. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*. 1877–1894.
[226] Carole-Jean Wu, Ramya Raghavendra, Udit Gupta, Bilge Acun, Newsha Ardalani, Kiwan Maeng, Gloria Chang, Fiona Aga, Jinshi Huang, Charles Bai, et al. 2022. Sustainable ai: Environmental implications, challenges and opportunities. *Proceedings of Machine Learning and Systems* 4 (2022), 795–813.

Received 8 March 2024; revised 25 March 2025; accepted 31 May 2025