IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received March 9, 2022, accepted March 22, 2022, date of publication March 28, 2022, date of current version April 4, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3162863

Recent Advances in Data Engineering for Networking

ENGIN ZEYDAN, (Senior Member, IEEE), AND JOSEP MANGUES-BAFALLUY
Centre Tecnològic de Telecomunicacions de Catalunya, Barcelona, 08860 Castelldefels, Spain

Corresponding author: Engin Zeydan (engin.zeydan@cttc.cat)

This work was supported in part by the EU H2020 5GROWTH Project under Grant 856709, in part by the Generalitat de Catalunya under Grant 2017 SGR 1195, and in part by the National Program on Equipment and Scientific and Technical Infrastructure through the European Regional Development Fund (FEDER) under Grant EQC2018-005257-P.

ABSTRACT This tutorial paper examines recent advances in data engineering, focusing on aspects of network management and orchestration. We provide a comprehensive analysis of standardization efforts as well as platform development activities related to data engineering driven network design. We then focus on the integration aspects of the data engineering ecosystem and telecommunication networks. The results of our tutorial investigation show that despite various efforts towards standardization and network management and orchestration platforms, there is still a significant gap in applying recent developments in the evolving data engineering world to the telecommunication domain. New advanced functionalities in data engineering as well as clear separations between the building blocks of data engineering pipelines within the proposed standardized architectures have been overlooked or not explored in detail by the standardization or platform development bodies in the telecommunication domain. Therefore, at the end of the paper, we discuss these gaps and research challenges in the context of future development processes for data engineering-driven network design and applications of data engineering concepts in telecommunication networks. We also propose several recommendations for early adoption of these technologies and frameworks in telecommunication infrastructures and platforms.

INDEX TERMS Data engineering, network management, orchestration, tutorial.

I. INTRODUCTION

Over the past few decades, telecommunication operators and service providers have experienced exponential growth in connectivity. At the same time, there has been an increased demand for massive connectivity, huge amounts of data, and in some cases ultra-low latency communications. Because of this, network complexity has increased placing a tremendous burden on telecommunication providers to manage and orchestrate the network. To address the highly complex issues that such larger and highly integrated networks pose in the design, analysis, deployment, and management phases, recent advances in data science and engineering technologies in both academia and industry, have encouraged the adoption of various Artificial Intelligence (AI)/Machine Learning (ML) platforms and frameworks at different layers of the telecommunication network infrastructure. On the other hand, the advanced techniques used by large companies, such as Google, Facebook, NetFlix, Apple, and Amazon, to demonstrate how to leverage data, have led to major breakthroughs in the business landscape in terms of improving products, services, and customer experiences. As a result of the increasing computing power of computers and advances in AI/ML algorithms from IT and the cloud giants, new data processing capabilities are being introduced that have disrupted entire industries, including telecommunications. Therefore, improving network intelligence has been the focus of interest in the telecommunication world in recent years with many diverse and compelling use cases [1]–[7]. (Note that the detailed use cases and their classifications within each Standards Developing Organizations (SDOs) and alliance bodies are given in Section XI.)

Advanced algorithms (e.g., neural network-based Deep Learning (DL) algorithms) and computational patterns used in AI/ML platforms can help discover valuable information hidden in vast amounts of numerical data (images, videos, datasets, etc.). This will enable better decision making in the development of telecommunication infrastructure products,

The associate editor coordinating the review of this manuscript and approving it for publication was Derek Abbott.

services and applications, while opening up innovative business opportunities. The recently evolving AI/ML ecosystem, its open source community [8], partnerships between the public private industry and academia [9], and the results from the labs of large cloud and IT giants like Facebook AI Research (FAIR),[1] Google AI[2] and Microsoft AI[3] etc. will support and enhance data-driven intelligence by gaining real-time (from streamed data) and long-term (from stored data) insights to understand what is going on in their infrastructure and develop competitive advantages. This will help develop better and more personalized services for telecommunication providers.

However, a major challenge for all telecommunication providers in the world is to leverage recent advances and find current technologies to develop data science and engineering platforms. This is because of the various challenges in managing infrastructure and utilizing vast computing resources. Telecommunication companies serve millions of users who depend on their services for their daily needs. In order to maintain these critical services without interruption, telecommunication providers must be prepared to obtain the most relevant and accurate information so that they can make informed decisions and take the necessary actions. For this reason, the design, implementation and maintenance of systems that can process incoming telecommunication-related raw data sources and produce high-quality, reliable information to support data analytics and AI/ML systems is critical and falls within the scope of data engineering.

A. RELATED WORKS

There are numerous comprehensive review and position papers on the recent developments and deployment of AI/ML software, libraries and frameworks and their applications for data centers, network traffic, Software Defined Networking (SDN)/Network Function Virtualization (NFV)-enabled networks or Industry 4.0 [8], [10]–[25], [25]–[40].

In [10], the authors bridge the gap between DL and mobile and wireless networking research by presenting recent surveys and showing a typical pipeline of an application-level mobile data processing system. The paper in [11] provides an overview of DL for wireless communication networks. The authors in [12] explore the applications of DL algorithms in wireless networks for different network layers, including the physical layer, the data link layer and the routing layer. The paper in [13] gives an overview on unsupervised ML approaches that are applied in the networking domain. In [14], a framework for data-driven networks for proactive optimisation and related technologies in online data analytics for 5G systems are explored. The paper in [15] provides an overview of evolving ML algorithms applied to self-organizing cellular networks.

The authors in [16] focus on the possible solutions on how ML can help support the targeted 5G network requirements. The papers in [17], [18] provide an overview of ML algorithms and their applications in SDNs. The authors in [19] focus on the use of AI and ML for the design and operation of Beyond 5G networks. The paper in [20] gives an overview of ML in wireless communications and presents some unresolved issues. The paper in [21] gives an introduction to the use of data science and describes steps towards knowledge discovery in the context of wireless networks. In [22], various software architecture practices that exist in the context of ML-based software systems are described. The authors in [23] provide an overview of related research on AI-based green communication and how it can be used to accelerate applications in 6G. The position paper in [24] presents an agenda for addressing the challenges associated with analyzing network data through AI/ML so that it can be naturally adopted in the network domain. In [25]. a comprehensive overview of ML solutions for 5G cellular networks is given. However, the focus of these contributions is on data science aspects rather than on data engineering frameworks and building End-to-End (E2E) data engineering pipelines. There is also a lack of detailed analysis of the applications of these technologies in larger areas/domains of E2E network management and orchestration.

From a network management perspective, the paper in [26] addresses ML solutions that can be used as a tool for implementing network management, automation and self-organization from a 5G perspective. Self-healing solutions for emerging and future mobile networks are explored in [27]. Networking issues in Big Data are addressed in [28]. The authors in [29] discuss the role of AI techniques in the emerging concept of Zero-touch network and Service Management (ZSM). Although these papers focus on network and service management issues, the emphasis is on applying data science concepts to network management rather than exploring data engineering aspects. The survey paper in [30] brings together both Big Data Analytics (BDA) and Network Traffic Monitoring and Analysis (NTMA) research, and focuses specifically on approaches and technologies that can manage the big NTMA data. However, the focus of this research is only on aspects of network traffic monitoring and analysis, concentrates on a limited number of available Big Data tools, and lacks analysis of network management and orchestration aspects. The survey paper in [39] focuses on Data Center Networking and divides it into infrastructure and operations. However, the focus of this paper is on the operational and infrastructure aspects of data center networking rather than merging it with data engineering concepts and developments.

The authors in [31] conducted a comprehensive survey of wired, wireless, and hybrid data centers to assess whether they can meet the requirements of a real-time analytic Internet of Things (IoT) network. However, the analysis focuses only on real-time analytics and does not examine the network management and orchestration aspects. The paper in [32] provides an overview of the role of BDA in designing a

[1]Facebook AI Research, https://research.fb.com/publications/
[2]Google AI Research, https://research.google/pubs/
[3]Microsoft AI Research, https://www.microsoft.com/en-us/research/

variety of data communication networks. The authors in [33] aim to design an industrial data platform that provides higher real-time performance and compression ratio for industrial data acquisition and processing. However, these works lack details on building an E2E data engineering pipeline and do not unify these data engineering components with network management and orchestration. The authors in [34] provide an overview of Big Data tools, but the focus is on manufacturing applications rather than telecommunication networks. The review paper in [8] provides a comprehensive overview of the latest AI software developments with comparisons and trends in the development and usage process. In [35], a detailed survey of distributed learning frameworks such as federated learning, federated distillation, distributed inference, and multi-agent reinforcement learning over real-world wireless communication networks is presented along with the rationale for their deployment over wireless networks. However, these papers mainly focus on data science developments rather than data engineering aspects by presenting only the latest developments in DL, distributed learning or machine learning frameworks and libraries.

The paper in [36] provides an overview of popular Big Data frameworks for processing big data and compares them using batch and iterative workloads. However, the authors do not address how they relate to recent developments in network management and the orchestration aspects of telecommunications. In [37], the authors present an example BDA pipeline and architecture called LambdaTel for telecommunication enterprises. Similar to our six-stage categorization of the data engineering pipeline, the life cycle of BDA is divided into four sequential stages namely data acquisition, data preprocessing, data storage and data analytics in [38] for wireless networks. On the other hand, these pipelines do not focus on the application of these sequential phases to network management and orchestration problems in industry, academia, and standardization bodies. Moreover, they only cover a few components compared to the data engineering pipeline proposed here (e.g., details on the data management & orchestration component are missing).

In contrast to the survey and review papers, this paper provides a tutorial view of the recent date engineering solutions to drive state-of-the-art developments in telecommunications, including papers in this research area that fills in the gaps of the previous survey and tutorial papers on BDA and network management & orchestration. Despite the previous work on the potential applications of Big Data infrastructures in telecommunication networks, there is still a lack of a general E2E data engineering architecture for telecommunication and networking domains. In particular, it is unclear to practitioners, researchers and developers in this emerging field how to systematically collect, ingest, analyse, process, visualize, and manage telecommunication data for network management and orchestration. Therefore, in this tutorial article, we explain how telecommunication networks can be used for data-driven analysis and autonomous optimization in future telecommunication networks using data engineering. In this paper, we mainly focus on current data engineering concepts and technologies related to network management and orchestration, which are essential pillars for creating data-driven intelligent support for telecommunication operators. Finally, we point out gap analyses, major challenges, and research issues that need to be addressed in the future.

Data engineering concepts have emerged to develop scalable and resilient data warehouse systems or tools that can support complex analytics across AI/ML infrastructures, platforms, or products desired by data scientists, ML engineers, or product teams. It is designed as a superset of Business Intelligence (BI), data warehousing, DataOps, data management, data architecture, orchestration, and software engineering. Depending on complex business and technical requirements at large scale, data engineers are expected to continuously evolve data pipelines and processing models. An integrated environment for data acquisition, storage, analysis, monitoring, visualization, and a scalable computing environment in which data-driven applications and analytics tools can be deployed is of interest to the data engineering field. Data analytics enables organizations to gain key insights into the user experience using data pipelines. These pipelines are a key component to continuously leverage the evolving data-driven ecosystem in the ecosystem.

Historically, data engineering has its roots in the trending topic of Big Data, when the Hadoop project code was first released in 2006 [41]. Hadoop became popular by providing distributed Big Data storage and processing capabilities. The main systems consisted of four main modules: Hadoop core, Hadoop Distributed File System (HDFS). Map-Reduce [42], and Yet Another Resource Negotiator (YARN) [43]. After the introduction of Hadoop, many Apache projects emerged that built their core functionalities on top of Hadoop and from which the Hadoop ecosystem evolved. At the same time, Hadoop is still relevant today and continues to be used as the core foundation for many data engineering projects. In parallel with this movement, many organizations today rely on similar design principles and advanced algorithms on distributed systems to process data storage, messaging, management, and compute functions on multiple servers in parallel.

The movement and processing of data can be achieved by creating streaming pipelines or data pipelines. In data pipelining, multiple data processing modules are chained together and the output of each module is used as input to the next module. Data engineering toolboxes enable organizations to process huge amounts of data reliably and quickly while gaining access to better, cheaper, and more accessible data analytics software and services. On the other hand, each newly introduced technology or component within a data engineering pipeline brings its own configuration, protocols, metrics, and tools, adding complexity to the overall platform.

Today's challenges and requirements for a data engineering solution include processing millions of tasks per second, latency in the order of sub-milliseconds, stateful computation (via functions that store data across processing items or

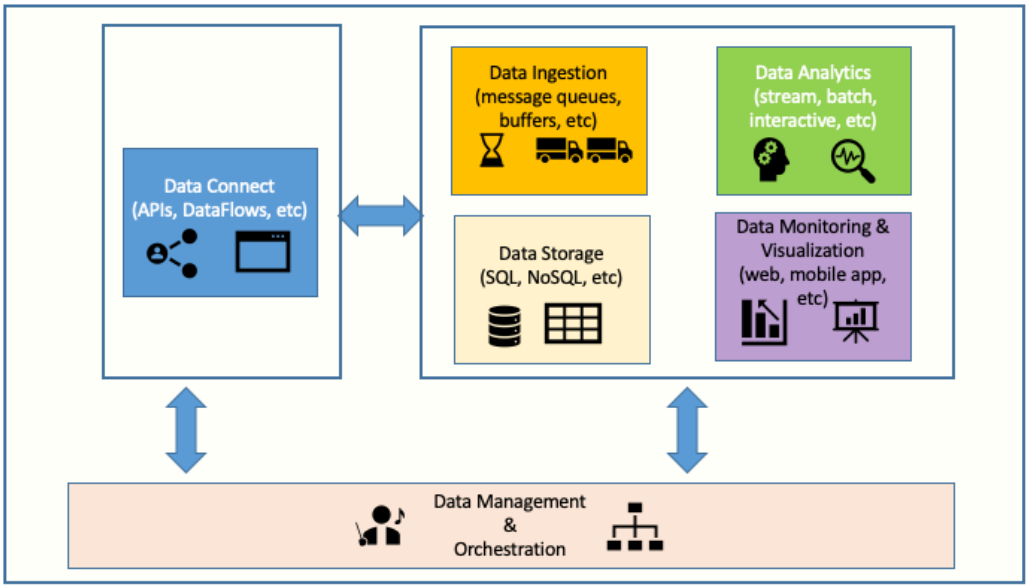

FIGURE 1. A high level illustration of the individual modules of a general data engineering platform to support the entire lifecycle of AI/ML.

events), and ensuring fault tolerance (e.g., by checkpointing state to recover state and positions in the stream). Most toolboxes are designed for scalability and are deployed in distributed environments where aspects of data distribution, replication, and coordination become important differentiators. At the same time, much of the software is becoming commoditized through open source software and packages, while the supply of data engineering solutions from cloud providers such as Microsoft, Amazon, and Google is increasing. As the data toolbox matures and the data engineering ecosystem blossoms, innovative solutions such as Online Analytical Processing (OLAP), scalable machine learning analytics will become more tangible to larger communities and enterprises.

B. OVERVIEW AND TUTORIAL OBJECTIVES

Fig. 1 shows an overview of the different modules of a general data engineering platform. It mainly consists of six modules: Data Connect, Data Ingest, Data Analysis, Data Storage, Data Monitoring & Visualization, and Data Management & Orchestration. Note that the connections between each module are only loosely represented and there may be multiple interfaces between these modules depending on the use case. Therefore, there may be multiple pipelines based on Service Level Agreements (SLAs) or non-functional requirements. One pipeline may be suitable for real-time notifications, while another may be more suitable for more relaxed requirements. Furthermore, in some scenarios, such as IoT networks, raw data (e.g., temperature, humidity) collected via the Data Connect module can be integrated with the Data Visualization component either via mobile applications or web user interfaces for direct visualization in a dashboard. In other scenarios, further data analysis and processing using recent advances in AI/ML algorithms may be required (e.g., in the case of high quality prediction, statistical analysis or root cause analysis [44]), which takes place between the Data Connection and Data Visualization modules.

In other scenarios that require high reliability of data (e.g., in Ultra Reliable Low Latency Communications (URLLC) services of 5G networks), the Data Ingestion and Data Analysis modules (for ultra-low latency real-time event processing) need to be embedded in the data engineering pipeline. More details on the modules and the corresponding distributed computing frameworks/landscapes available today to run each of these modules are provided later in the paper. In addition to the open source frameworks, we have also listed vendor-specific tools/frameworks for each module whose main advantage over in-house setup is the ease of set-up and maintenance support for the data engineering applications. On the other hand, cost can be considered as one of the main disadvantages when these applications are deployed on a large scale with vendor-specific tools.

The main contributions of this paper can be summarized as follows:

- Our goal is to provide a comprehensive and thorough overview of the recent advances and major technologies used in the context of data engineering.
- The paper categorizes the various modules of a general data engineering platform into Data Connect, Data Ingest, Data Analysis, Data Storage, Data Monitoring & Visualization and Data Management & Orchestration. This allows for easy understanding and comparison of studies within each area of the data engineering landscape.
- Our goal is to link the capabilities of the data engineering ecosystem with a possible link to future telecommunications systems. Unlike previous work on data

engineering, this paper also explores the necessary link that needs to be established between recent advances in data engineering and traditional telecommunication ecosystems in the context of network management and orchestration.
- The paper provides a comprehensive discussion of challenges, gaps and future directions in the convergence of data engineering and network management and orchestration, and also identifies research directions that arise.
- To drive research in data engineering solutions for future network management and orchestration solutions, we also discuss possible solution methods for each of the above challenges.

The remainder of the paper is arranged as follows. Section II introduces Data Connection frameworks and the possible data sources. Section III presents frameworks for Data Ingestion. Section IV discusses the latest frameworks for Data Processing and Analysis. Section V presents the latest frameworks for Data Storage. Section VI presents frameworks for Data Monitoring and Visualization. Section VII presents frameworks for Data Management and Orchestration. Section VIII discusses the relationship of data engineering projects with data science frameworks and AI/ML platforms used in the industry. Section IX gives an overview of network lifecycle management and orchestration. Section X provides an overview of standardization efforts in network management and orchestration and how they can be related to data engineering frameworks. Section XI gives an overview of data engineering use cases in telecommunication networks. Section XII provides the gap analysis, challenges and future directions. Finally, Section XIII presents the conclusions of the paper.

II. DATA CONNECTION FRAMEWORKS

The data connection serves as a trigger to connect to a data source, which can be either in a local file, on the web, on a mobile/IoT device, in a metastore table, or in a data store. Data sources can be in various forms, such as static data sources from files, databases (MySQL, MongoDB, etc.), network/server/storage or clickstream (web stream) logs, data taken from online resources, streaming on-demand data sources from third-party Application Programming Interfaces (APIs) or frameworks. This stage can be configured to transfer data into a Data Ingestion module. The connection frameworks are especially important for telecommunication service providers to enrich the already existing data with third party sources.

A. AVAILABLE TOOLS

Representational state transfer (REST)-APIs are mainly used to import data from many third-party APIs into the Big Data processing cluster. It benefits from the ideas of stateless servers and structured access to resources. An API gateway is a software component and acts an entry point into a system. It is responsible for allowing multiple APIs, backend systems or microservices to be accessed reliably and securely by end users. Kafka Connect, which is part of Apache Kafka [45], is used to enable data flows between Kafka and various types of systems such as message queues, Hadoop, Spark, Flink, TensorFlow, databases, object stores or flat files. It supports pluggable connectors (which are essentially jar files that can be downloaded from Confluent Hub for example). Apache Flume [46] is a service for efficiently collecting, aggregating and moving large amounts of data in a distributed and reliable manner. GraphQL[4] is a query language for APIs and designed as an alternative to REST-API that allows a variety of different frameworks to connect from the client side during client-server communication. The advantage of GraphQL is that it prevents over- and under-fetching of data compared to REST-APIs. Falcor[5] is another open source library used to retrieve data that may reside in a client's memory or over the network on the server. Apache NiFi[6] is mainly used to automate data movement between different systems. It provides a web-based user interface for creating, monitoring or transforming (e.g., converting Comma Separated Values (CSV) files into individual JavaScript Object Notation (JSON) records) data streams.

1) VENDOR SPECIFIC TOOLS

Amazon's API Gateway (for connecting to devices), Azure Event Hub and Data Gateway (a proxy that provides on-premise access to data) and Google Cloud Platform (GCP)'s Cloud Dataflow, other connector platforms such as FiveTrain,[7] Stitch,[8] Matillion[9] are some example tools and frameworks for data connectivity.

B. DATA SOURCES TO CONNECT

In telecommunication operators, there are various sources of information from which data can be retrieved for further analysis by data engineering frameworks. Generally data is available in three different systems: Information Technology (IT), network and application systems.

- **IT systems data** is collected by various IT systems such as Customer Relationship Management (CRM) systems, billing systems or customer care services. Some examples are basic customer, account, and user profile and characteristics information, billing data, business consumption data, social media data and marketing/sales department promotions/campaigns, etc.
- **Network systems data** (from both wired and wireless networks) includes network equipment data, Call Detail Records (CDRs), eXtended Data Records (XDRs), Machine-to-Machine (M2M) data, traffic data (both signalling and payload data), Operations Support Systems (OSS) data (network events (outages, alarms),

[4]http://spec.graphql.org/, accessed December-2021
[5]https://netflix.github.io/falcor/, accessed December-2021
[6]https://nifi.apache.org/, accessed December-2021
[7]https://fivetran.com/, accessed December-2021
[8]https://www.stitchdata.com/, accessed December-2021
[9]https://www.matillion.com/, accessed December-2021

network performance data, etc.), voice, SMS, network service data, user equipment (UE) mobility and location updates, quality-of-service (QoS) parameter data collected from access, transport and core networks (e.g. from Access and Mobility Management Function (AMF), NG-RAN, Authentication, Authorization and Accounting (AAA), Home Subscriber Station (HSS) servers, etc.). Some of the typical data sources in mobile networks are also described in [47].

- **Application/service data** is data from products and services (e.g., online mobile payments, online music and e-wallet applications or vehicle tracking, power grid information and health services, other value-added services, etc.) provided by telecommunication operators that contain user data (e.g., user access modes, addresses, timestamps, business preferences, consumption habits, customer care agent's data). Note that the underlying structures of this data are complex (either unstructured (text, images, videos), structured or semi-structured), so targeted data engineering pipelines for different data types are required depending on the use case.

III. DATA INGESTION FRAMEWORKS

Along with the advent of 5G, IoT and mobile sensor devices, powerful messaging platforms are needed that can ingest and cache all traffic for later processing. To reliably publish and subscribe to events, highly available, fault-tolerant ingest pipelines are required that can serve as the backbone of the streaming data infrastructure. To realize this, data ingest frameworks enable replication and partitioning of data across nodes in the cluster. The data ingestion module acts as an intermediary or multi-tenant data hub that connects incoming data from different data sources to diverse sinks to ensure that data is not lost during this movement. The source systems can be any vendor application, database or event. Data Ingestion is generally used to move data between external systems and Big Data clusters or data lake (e.g. based on Hadoop) for batch or stream processing (e.g., for filtering and mapping operations). A stream is an unbounded and continuously updated data set. In general, it consists of sequences of key-value pairs that are ordered, replayable, and fault-tolerant. Streaming data can be injected into clusters in real-time or near real-time. When loaded the data can be used for later processing (e.g., with Apache Spark) or storage (e.g., with HDFS).

When communicating during data ingestion, two types of message delivery patterns can occur [48]. One is **queuing** and the other is **streaming**.

A. QUEUING

In queuing, the order is not important because all events from the message queue are transferred from the device/user to a system. Some examples are payments and transaction processing, where they are mostly used for mission critical systems. Message queues provide asynchronous communication protocols where the sender and receiver do not need to interact with the message queues at the same time and are common features of data ingestion frameworks. Platforms and protocols such as RabbitMQ [49], JMS, AMQP and others enable asynchronous data integration between multiple systems by acting as a central hub. They are best suited for message queuing applications, such as real-time transaction services with zero tolerance to data loss. Therefore, consistency and durability are the most important features of these systems. On the other hand, these systems may suffer from scalability issues during peak loads (e.g., RabbitMQ is not designed as a distributed system). This is also true for web service/API based architectures due to their synchronous communication [50], [51].

Apache Kafka [45], on the other hand, is a popular and widely used framework for real-time processing of streaming data and is best suited for managing data pipelines to move large amounts of data between different systems. As a common event bus that decouples producers and consumers, it can handle hundreds of thousands of events per second, is scalable and widely used in the industry (used by many companies such as Twitter, Netflix, etc.). Therefore, integration with other technologies and frameworks in creating data pipelines has also become easy. For example, data sharing is possible with many interfaces including file-based systems, real-time messaging, REST web service, Structured Query Language (SQL) or Not Only SQL (NoSQL) databases, data lakes or data warehouses. In addition to simple data sharing, messaging and integration, it can also be used for data storage and processing. However, Kafka is not suitable for storing and processing large files such as images and videos as a whole. To ensure the reliability of streaming requests, Table 1 provides the three different consistency guarantees available for data ingestion or stream processing. Note that Message Queuing Telemetry Transport (MQTT), a lightweight publish-subscribe messaging transport protocol, provides a real-time and reliable messaging service and has also defined similar QoS levels. QoS level 0, level 1, and level 2 are at most once, at least once, and exactly once respectively. QoS level 0 acts as a best-delivery mechanism, QoS level 1 waits for the receiver's PUBACK packet and retransmits it if it is not received, and QoS level 2 sends a sequence of four messages to guarantee exactly once reception.

B. STREAMING

Stream processing engines/platforms (such as Spark Streaming, Apache Flink) enable strictly ordered and exclusive message passing while allowing computational logic to be applied to message streams [52]. In a streaming-based communication pattern, the ordering of events is important so that behaviour can be analyzed based on the sequence of ordered events. Streaming is most suitable for stateful applications and OLAP-oriented use cases (e.g. BI, dashboards, ML, etc.). Some examples are user behaviour analysis, anomaly detection and web traffic log analysis. In streaming, the loss of data can be tolerated in some parts as long as the correct

ordering of events is maintained. This is because stream processing systems provide fault tolerance and retries by rewinding the stream and replaying each event from the point of failure or occurrence of an error [53]. Apache Storm [54] is a distributed stream processing framework designed for both batch and distributed streaming data processing. Libraries of processing frameworks such as Spark Streaming (an extension of Spark's API [55] for stream processing), Apache Flink (using DataStream API for bounded (finite size)/unbounded (infinite in size) streams) also have data ingestion capabilities. Spark Streaming uses a micro-batch architecture for continuous data processing that can ingest data from Apache Flume, TCP websockets, or Kafka producers. Spark relies on exactly-once processing to ensure correctness.

Apache Pulsar[10] has recently emerged as a competing technology to Kafka. Pulsar has similar features to Kafka and acts as a distributed pub-sub messaging system with some differences in architecture, performance, and features. Pulsar can also integrate with full-fledge stream processing frameworks like Spark and Flink. On the other hand, Pulsar offers more flexibility as it uses a layered design compared to the monolithic design of Kafka. For example, Kafka uses Zookeeper (with plans to remove it in future releases with a new controller metadata quorum) and the Kafka broker itself (a two-tier architecture that tightly couples storage and serving), while Pulsar requires three distributed systems: Zookeeper, Apache Bookkeeper (three-tier architecture), and also RocksDB for certain storage tasks. The computations are performed on a broker in one tier and the stateful storage is managed in another tier (Bookkeeper). Therefore, Pulsar aims to separate serving and storage in different tiers and provide both queuing (messaging) and streaming capabilities in a single system. To achieve this, Pulsar offers four types of subscriptions, depending on the application's ordering and consumption scalability requirements:

1) **Exclusive subscription:** Only individual consumers may subscribe.
2) **Failover subscription:** Multiple consumers may subscribe to a single topic.
3) **Shared subscription:** Messages are delivered to multiple consumers in a round-robin fashion. This allows the number of consumers to scale beyond the number of partitions.
4) **Key_Shared subscription:** Multiple consumers can join the same subscription and message delivery will be shared among consumers that have the same key. It allows higher scaling as well as order guarantees at the key level.

For messaging/streaming applications, exclusive and failover subscription modes are most suitable for scenarios where partition level order guarantees are needed, while queuing applications can use shared and key shared subscriptions [56].

[10]https://pulsar.apache.org/, accessed December-2021

C. USE CASES AND REQUIREMENTS

During data ingestion, data enrichment, filtering, aggregation, transformation, etc. can also be performed for better use in sinks. Stream-based architectures have been shown to provide a better architectural foundation for many use cases in the industry including the use case for fraud detection [57]. In the telecommunications domain, the authors in [37] have provided several use cases for the application of streaming. Within the traditional telecommunication landscape, there are several components within the OSSs, Business Support Systems (BSSs), and OSS-BSS integration modules. The OSS/BSS landscape covers various functionalities across mediation, billing, CRM, e-business, data warehouse, service assurance, provisioning, etc. [58]. Therefore, the data ingestion frameworks described in this paper (such as Apache Kafka) can be used in various ways in these OSS, BSS and OSS-BSS integration development projects, e.g. in critical business applications such as payment or fraud detection applications.

Application requirements may vary depending on real-time (less than 10 ms), near-real-time (less than a few minutes), or high-throughput (processing data on the order of PB/day in a single cluster). At the same time, data ingestion can be performed with different components: Batch processing jobs managed by data orchestration engines can be used for complex processing and deep analytics. On the other hand, streaming jobs (e.g., Spark Streaming, which consumes data from Apache Kafka streams) can be used for fast feedback and anomaly detection, sync-async, publish-subscribe, change data capture, or REST services that expect data for the data cluster. Various changes can be made in the data ingestion frameworks to meet these different application requirements. For example, for applications that require low-latency, the *write-once-read-multiple times* (WORM) model can be used, where a dataset generated or copied from the source can be used multiple times later for different analyses [41].

When ingesting data streams, it should be possible to query the data as soon as it enters the system to enable immediate actions and insights. For this reason, various preprocessing analytics such as cleansing, profiling, aggregation or enrichment of the dataset can improve query response time. In the WORM model, querying can be faster if there is an additional cost in data entry, for example if each map-reduce job is entered before the analytic queries. Some of the fast query systems like Apache Druid [59], HiveQL [60] are based on the principle of additional cost incurred in data ingestion. For this reason, before executing queries, each data segment must be ingested using some map-reduce ingestion jobs. This is also related to the mutation rate requirements of data. For data coming from transaction applications, extensive writes must be performed via Online Transaction Processing (OLTP) (which incurs additional costs) and for data coming from OLAP systems, extensive reads can be performed but only small writes.

In the telecommunication industry, there are many use cases where open source or proprietary/vendor-specific frameworks for data ingestion are already in use, e.g. in monitoring, middleware, event hub platforms, and business applications (billing, supply chain management, BSS, B2B products, etc.). Communication of these various entities in a large enterprise via message brokers has certain advantages over REST-based API. In fact, there can be data ingestion issues when scaling applications over REST-APIs. As the application grows and more services are added, complex relationships between services arise, requiring API connections to be remapped. This also delays the workflow development and implementation process. Moreover, due to the synchronous communication structure of REST-based API's, when there is an influx of data, i.e., sudden burst requests, some of the services may operate slowly or even be unavailable, affecting the reliability of whole application [50]. For this reason, message queuing (especially for streaming applications) can be more robust compared to REST APIs and scale with the requirements of each application [51]. In the case of microservices based architecture, the ability to use pub/sub systems may be an appropriate way to inform microservices of the potential events (request) that may require a reply from the corresponding receiver (response).

D. OTHER AVAILABLE TOOLS

In addition to the Apache Kafka and Apache Pulsar streaming/messaging tools described above, there are other data ingestion frameworks that can work at scale [61]–[65]. Logstash and Beats are the core components of Elastic Stack that are used for ingesting data from any data source and transferring it to Elasticsearch [61]. Apache Heron is a real-time stream processing engine that has proven itself at Twitter for big data [62]. Apache Gobblin[11] is a distributed data integration framework that simplifies data integration for both streaming and batch data ecosystems. It is used to extract, transform, and load large amounts of data from various sources, such as databases, REST APIs, onto Hadoop, and also simplifies data ingestion, organizational replication, and lifecycle management. Apache RocketMQ[12] is a distributed messaging and streaming platform. Redis Pub-Sub [63] is an implementation of the messaging system provided by Redis. Apache Sqoop [64] is used to import/export data from/to MySQL databases to/from HDFS in specific file formats. Apache Camel [65] is an open source integration framework designed as message-oriented middleware that provides interfaces for the Enterprise Integration Patterns. A good comparison of some of the most recent message queuing systems (Kafka, RabbitMQ, RocketMQ, ActiveMQ, and Pulsar) can be found in [66]. A curating list of existing streaming frameworks and applications can also be found in [67].

[11]https://gobblin.apache.org/, accessed December-2021
[12]https://rocketmq.apache.org/, accessed December-2021

1) VENDOR SPECIFIC TOOLS

Amazon's Kinesis Streams,[13] (for buffering purposes and works similarly to Kafka), Facebook's Puma, Swift, and Stylus stream processing systems [48], Google's Cloud Pub/Sub[14] (data ingestion and messaging for event-driven systems as well as streaming analytics), and Azure's Event Hubs,[15] IoT Hub,[16] Stream Analytics[17] are the respective data ingestion services offered by the IT and cloud giants. Splunk[18] collects and analyzes machine-generated data on a large scale. As a messaging system, Oracle Enterprise Messaging Service and IBM Websphere MQ are other examples of event buses for processing asynchronous data streams.

IV. DATA ANALYSIS AND PROCESSING FRAMEWORKS

After data is ingested, it must be analyzed to gain insight so that resources, services, or assets can be managed more efficiently. The data processing and analysis phase ensures that this is achieved through various tasks such as data transformation, enrichment, filtering/deduplication, mapping, cleansing, updating state, joining, grouping, defining windows, aggregation, staging, integrity checking and combining streaming and batch data. Data analysis and processing frameworks generally fall between data ingestion and data storage frameworks. All data processing tools first consume data, then process it, and finally produce results. Batch processing involves processing large amounts of data, usually stored in data lakes/warehouse. In this case, the data does not change much or at all. Moreover, SLAs are loose in terms of processing time for a given dataset.

In batch processing, the results are obtained slowly but accurately. In stream processing on the other hand, the data is constantly changing (e.g., Twitter or Facebook feeds or weather sensor data in real-time) and is not stored in large data warehouses. At the same time, SLAs are more rigid and data processing should be done in real-time to serve other processes, i.e., no shifts in data processing are allowed as in batch processing. In streaming, the results can be fast depending on the requirements, but sometimes they need to be approximate because of the trade-off between low latency analysis and efficient use of computational resources over a distributed infrastructure. For this reason, the paradigm of approximate computing is used in the literature for streaming analytics, which allows trading output accuracy for computational power by analyzing only a subset of the input [68].

A. DATA ANALYTIC CATEGORIZATION

Data analysis can be categorized into four dimensions [71].

[13]https://aws.amazon.com/kinesis/, accessed December-2021
[14]https://cloud.google.com/pubsub/docs, accessed December-2021
[15]https://azure.microsoft.com/en-us/services/event-hubs/, accessed December-2021
[16]https://azure.microsoft.com/en-us/services/iot-hub/, accessed December-2021
[17]https://azure.microsoft.com/en-us/services/stream-analytics/#overview, accessed December-2021
[18]https://www.splunk.com/, accessed December-2021

TABLE 1. Comparisons of three different consistency guarantees based on reliability requirements of the streaming request.

Consistency Guarantees	Description	Example Applications
At Least Once	—The message is pulled once or multiple times and processed each time (likely duplicates are received). —During transmission, it is not possible for message to be dropped or lost, hence the receipt is guaranteed. —No data is missed. —Duplicate messages can be allowed and ignored after further processing considering the timestamps of the records. —Typical usage at scale	—Scaling service applications —Tracking a user's (or device) location via cell phone (or vehicular) records.
At Most Once	—The message is pulled once (no duplicates are allowed), hence the message may or may not be received. —Applicable for use cases where possible missing data is tolerable —Typical usage at scale	—Periodic sensor data (e.g. temperature, humidity) sent in high frequency intervals. —Scaling service applications
Exactly Once	—Message is pulled one or more times but processed once compared to at least once (no duplicates are allowed). —Like at least once, receipt is guaranteed, no missing data is allowed. Hence, there is only one complete successful process. —Especially useful for one-time processing scenarios such as where exactly once processing is crucial even when a broker or instance of function fails. —The complexity of the system increases with exactly once transmission mode of operation which introduces latency. —The benefits of exactly-once-processing should be weight against the drawbacks of additional latency cost. —Some data stream processing frameworks such as Flink Data Stream, Spark Streaming can enable exactly once processing at the expense of increased latency when there is deep pipelines and high volumes of input. —Distributed transactions at scale is difficult.	—Critical business infrastructure —OLTP applications (Transactional message streaming, billing systems, payment processing, etc.) —Mission-critical applications. —Other guaranteed lossless event processing applications (Image or video upload and processing in cloud).

- **Descriptive analytics** is used to understand what has happened in the past, including the recent past [72]. In this analytics, the data is examined statistically. Example: An alarm monitoring system has received an unusually high number of alarms at a telecommunication provider's network operation center. Using descriptive analytics can help understand what is going on in the infrastructure by using real data and corresponding statistics (alarm timestamp, location, severity, etc.).
- **Diagnostic analytics** is used to understand why something happened in the past [73]. It is a step further to descriptive analysis. For example, if many alarms have occurred in a telecommunication system, diagnostic analysis helps to understand the root cause of the alarms, such as power failure or connection error.
- **Predictive analytics** is used to predict what will happen or is most likely to happen in the future [74]. In a ML system, the model is trained and later used for inference in production systems. For example, an increase in alarms in network monitoring systems can be predicted to take preventive measures much earlier before an outage occurs, e.g., in transport links.
- **Prescriptive analytics** is used to recommend actions to influence outcomes [74]. It is another step further in predictive analytics. For example, if we know that alarms will increase due to a transport network failure, network operators can enable redundant transport links to route user traffic to a different routing path.

B. MODEL DEPLOYMENT

The frameworks developed in this module also complete the full cycle of the ML process through tasks such as feature extraction, distributed Graphics Processing Unit (GPU)-based model training, model deployment, model serving, and A/B testing. In practice, the models can be deployed in three different ways [75]:

1) **Offline deployment:** The model is deployed in an offline container and predictions are made ad hoc.
2) **Online deployment:** The model is deployed in an online model serving cluster where clients can send their prediction or batch requests over the network.
3) **Library deployment:** This case is similar to the online deployment in a serving container, except that the model is now embedded as a library and as another service.

Table 2 summarizes the various choices for ML deployment.

C. STREAM PROCESSING ENGINES

One of the earliest versions of data analysis and processing frameworks was based on the Hadoop's Map-Reduce paradigm. As the ecosystem continued to evolve, new and more advanced versions of Map-Reduce emerged based solely on batch processing, extending it to streaming applications as well. In streaming applications, data is continuously generated from multiple sources and frequently updated. Moreover, in most cases, the data sources send their information simultaneously. Therefore, it is important to analyze streaming data sources and derive, track, and interpret important streaming data quality metrics. In streaming applications, each data must be accurately processed to preserve its sequential order in time as well as its relationship with other data sources. A complex job like streaming with multiple input data streams and stages usually has many internal states

TABLE 2. ML deployment options.

ML Deployment Options		Platforms/ Frameworks
Deployment	**Platforms (Open Source)**	—MLflow, Kubeflow, —Seldon, Airflow, —Cortex*, Metaflow, —BentoML†
	Container Orchestration	—Kubernetes (AWS EKS§, GCP GKE) —Docker Swarm [69] —AWS (ECS, Fargate)
Data Versioning		—DVC, Wandb

ML Deployment Options		Platforms/ Frameworks
Customized Model Deployments	PyTorch Model	—TorchServe (any type of architecture) —Turbo-transformers‡ (transformer) —Convert to ONNX
	TensorFlow Model	—TensorFlow serving [70] —Convert to ONNX
	Any Model Deployment	—FastAPI¶ + Uvicorn ‖ —Gunicorn**
	ML to Deep Learning Conversion	—Hummingbird††
	MXNet Model	—Convert to ONNX
	ONNX Model	—ONNX Runtime (ORT) —Convert to PyTorch model —Convert to TensorFlow model

and performs more remote calls to other REST endpoints and databases, e.g., for joining operations.

Streaming can occur in various forms, e.g., socket-based streaming (connecting to a socket to get data), file-based streaming (connecting to a file stream to get data), or data ingestion tool-based streaming (e.g., connecting to Kafka or Kinesis to get queued messages reliably (i.e., with acknowledgements)). Note that analyzing and processing data in real-time data enables organizations to make decisions proactively, rather than reactively. Some of the applications of streaming data include scenarios such as sending real-time notifications to users via email or push notifications when events change in their user account, sending sensor data in vehicles, industrial equipment or machinery for predictive maintenance purposes, tracking of geolocation of user phones or vehicles for transportation and supply-chain purposes, monitoring patients' vital bodily functions for health purposes, or collecting streaming data and create personalized products and services for users of an online retail company. For these various reasons, streaming data analytics is becoming increasingly important for most industries.

Recently, many frameworks have also emerged that can are suitable for batch (offline data) and streaming (real-time events) analytics (Apache Spark [55], Apache Flink [76], Apache Beam,[27] etc.) while operating under a unified data processing layer to perform common computations and reduce data infrastructure complexity. These frameworks are known for their extremely scalable data processing capabilities that can analyze petabytes of data while extending to hundreds of instances. They can also provide different levels of abstraction. On the other hand, each of these frameworks has its own strengths and weaknesses.

Apache Spark [55] is a distributed computing platform that can run standard Extract-Transform-Load (ETL) processes on Big Data in batch and streaming mode using a set of APIs. It is capable of processing jobs 10 times faster than its Map-Reduce counterpart. The spark core API is able to load data from different platforms (e.g. Kafka, Flink, Kinesis, HDFS, Amazon S3, Azure Blob, etc.) and write it back to other platforms (e.g. Hadoop, Elasticsearch, Cassandra, etc.) after processing the data. The Spark ecosystem is also extensive and includes a number of libraries that can run on top of the Spark core and can easily implement and support various workloads and Spark programs such as BigDL [77] (which supports inference, transfer learning, and distributed training), TensorFlowOnSpark[28] (which supports inference, ML pipelines, and distributed training), Deeplearning4j[29] (which supports inference, transfer learning and distributed training), and DL Pipelines[30] (which supports large-scale inference/scoring, transfer learning, multimodal training, ML pipeline, and Spark SQL) for DL or Ray for Reinforcement Learning [78], etc. It can also work on training a model online using various techniques, including supervised, unsupervised learning (to ensure that the model is stable) and for online predictions or running model inference at run time. The core abstraction in Spark is essentially a dataframe or dataset (in the latest versions) that can be transformed to perform a number of operations. Spark Streaming can support flat files, TCP/IP data, Apache Kafka, Apache Flume [46], Amazon Kinesis or Twitter, Facebook, and other social media platforms as data sources.

Similar to Spark, Apache Flink [76] can be used for multiple operations and processes such as filtering, mapping, or running multiple stream joins on real-time or streaming data. Flink's core APIs like DataStream API supports both bounded and unbounded streams, while DataSet API supports bounded data sets for batch use cases. DataStream APIs

[27] https://beam.apache.org/, accessed December-2021

[28] https://github.com/yahoo/TensorFlowOnSpar, accessed December-2021k

[29] https://deeplearning4j.org/, accessed December-2021

[30] https://github.com/databricks/spark-deep-learning, accessed December-2021

TABLE 3. Various options to select different types of window operations depending on use case.

Windows	Types	Description
Time-driven Windows	Tumbling Windows	—There is no overlapping between the windows, e.g. observe strict T seconds intervals
	Sliding Windows	—Slide the window with the overlapping time intervals, e.g. monitor activities in the last T seconds
	Session Windows	—Based on the activity, e.g. during user session until pre-defined time-out period where user is inactive.
Data-drive Windows		—Windows every N elements (count window) and is only based on counts of events and is data independent.

allows for large and more sophisticated operations compared to DataSet APIs. Flink is designed to process streaming data quickly and accurately with its stateful stream processing capabilities.

D. WINDOWING OPERATION AND PARTITIONING

There are two main window types that can be applied to the data stream (see Table 3). For streaming applications, time spans are critical. The window concepts (tumbling, sliding and session window) that are mentioned above are closely related to the time characteristics, which are another important concept for data engineering pipeline and frameworks. In any data processing, the following key concepts of timing are important:

- **Event time:** represents the time at which an event is generated at the source of the system.
- **Ingestion time:** represents the time at which the event enters dataflow.
- **Processing time:** represents the time when an event was processed/observed in the system.

In an ideal scenario, the interval between event time and processing time should be zero (real-time applications). However, this may not be the case for various reasons (e.g. network congestion, software logic, etc.). Note that depending on the use case, time characteristics may be important (e.g. fraud detection, most billing applications).

Unlike Apache Spark Streaming which has time-based window criteria, Flink provides more powerful window semantics (both time- and data-driven window options) and allows to work with data-based or custom window criteria. Apache Samza [79], developed by Linkedin, is a stream processing framework for real-time analytics and is well integrated with Kafka. Apache Beam is used to build an execution pipeline that can implement both batch and stream processing under a unified programming model. For example, in the case of model serving, multiple runners such as Spark or Flink can be executed using Beam's distributed processing backends. So, in this respect, it is more of a software development kit than a stream processing engine. Frameworks such as Apache Livy[31] enable a REST service for executing Spark jobs via third-party applications.

In order to process large amounts of data, the data must first be partitioned so that it can be processed in parallel. For this reason, partitioning is a key concept. It can be done either in memory or on storage disk to improve performance and reduce cost. Most data processing frameworks perform partitioning in memory (e.g. Spark or Flink). There are several ways to manage a partition in Spark or Flink. Some of them are listed in the Table 4. Spark recommends 2-3 tasks per CPU in a given cluster to keep the level of parallelism high. For this reason, the recommended number of partitions in a given system is simply the number of CPUs $\times$[2 or 3].

E. QUERY-BASED ANALYSIS

Using SQL as a common interface for data has many advantages. Query-based engines allow to build applications that interact programmatically with metastore tables. Many different systems provide an SQL interface for streaming and batch data on different platforms. Originally developed approaches such as Apache Hive [60], Apache Pig [80], Trino[32] and Apache Impala [81] are used to query large distributed batch data with SQL-like syntax from Big Data storage systems such as HDFSs. On the other hand, newer data processing frameworks like Apache Spark SQL library (supports partial SQL functionalities), Apache Drill [82] (supports writing SQL queries similar to MySQL, Microsoft SQL Server, or Oracle), Apache Flink [76], Presto [83], Samza SQL [79], Apache Kylin [84], Apache Pinot [85] are some examples that are being used beyond these originally developed approaches to query data across many dimensions and metrics, various data sources including Hadoop clusters, NoSQL databases (Hbase, MongoDB, Cassandra, etc.), file systems/formats (CSV, Parquet, Avro, JSON, binary files, etc.), and cloud storage platforms. The graph query languages Cypher [86] and GQL[33] (which are more declarative and easy to read compared to SQL) are used for graph database queries. Kafka Streams or ksqlDB [45] is a Java-based library designed to

[31] https://livy.apache.org/, accessed December-2021
[32] https://trino.io/, accessed May-2021
[33] https://www.gqlstandards.org/, accessed December-2021

TABLE 4. Various ways to manage partitions in spark or flink.

Partition Management	Description
Repartitioning	Under this operation, the number of partitions can be increased or decreased. This is one of Spark feature (*transactions.repartition(n)* where n is desired the number of partitions) and is an expensive operation since it executes full shuffle.
Coalesce	This operation reduces the number of partitions and avoids a full shuffle. This is one of Spark feature (*transactions.coalesce(n)* where n is desired the number of partitions).
Custom partitioning	This operation uses a user-defined partitioner to select the target task for each element in case different performance is required from the in-built APIs. This is one of Flink's feature (e.g. *stream.partitionCustom(partitioner,0)*).
Random partitioning	Partitions elements randomly according to a uniform distribution. This is one of Flink's feature to avoid mulfunctioning of the system (*stream.shuffle()*).

Partition Management	Description
Rebalancing (Round-robin partitioning)	Partitions elements in a Round-robin manner to create equal load per partition. This is one of Flink's feature (*stream.rebalance()*).
Rescaling	Round-robin partitioning of elements, to a subset of downstream operations. This is one of Flink's feature (*stream.rescale()*)
Broadcasting	It broadcasts all elements into every partition. This is one of Flink's feature (*stream.broadcast()*).
—	—

provide streaming processing applications based on Kafka in a fault-tolerant and distributed manner. It can provide full-fledged stream processing capabilities that include consumption of Kafka topics, provision of state information, basic transformations, filtering, sliding windows, calls to ML jobs, exactly-once processing, etc. Similarly, ksqlDB allows processing of records/events using a SQL-like language. On the other hand, Kafka Streams is not well-suited for processing large amounts of data.

Apache Kudu [87] is designed for fast analysis of high speed data (that is rapidly changing data), and can perform queries over billions of rows and terabytes of data per second. In addition, there are several ways to access query-based Big Data frameworks, such as through a custom shell, a REST interface, a web interface, or through transport protocols such as Java Database Connectivity (JDBC)/Open Database Connectivity (ODBC) drivers, database protocols (MySQL, Postgres, Hive, etc.), and Remote Procedure Call (RPC) with Thrift, Protobuf, JSON, XML and so on. Spark's Thrift server, for example, enables this functionality by turning SQL queries into Spark jobs. Apache Arrow[34] is a language independent big data layout, enables in-memory analytics for fast processing and movement of data. It essentially enables sharing and serialization of high performance data and serves as a communication interface. Apache Arrow provides bindings between many components, e.g. reading a file in a given format (e.g., in Parquet data format) and converting it to another format (e.g., Spark dataframe/dataset) for further processing without conversion issues. Some other frameworks, such as the Ray architecture [78], are based on Apache Arrow.

[34]https://arrow.apache.org/, accessed December-2021

1) VENDOR SPECIFIC TOOLS

Google provides Big Data tools like DataFlow,[35] DataProc, BigQuery (query engine for static datasets), Cloud AutoML, etc. as a service to its customers. Amazon offers SageMaker for ML, AWS Kinesis[36] for real-time streaming, Amazon Athena (based on Presto), and AWS Redshift Spectrum for interactive query services and EMR. Microsoft offers Azure Event Hubs.[37] Databricks offers Databricks Unified Analytics Platform as a managed service. Dremio offers a cloud data lake engine for Big Data queries.

V. DATA STORAGE FRAMEWORKS

Data storage is part of the data pipeline. Storage systems can be divided into three different systems:

A. RELATIONAL DATABASES

Relational databases consist of constructs (e.g., tables and rows) and constraints (e.g., primary keys and referential integrity constraints) and provide both OLTP and OLAP. For structured data, traditional relational/transactional databases/systems such as MySQL, PostgreSQL are used for persistence performance. In these systems, data related to users (e.g. credentials data from frontend), production and business systems can be stored. In these systems reading data in the database is cheaper than writing data to the database. On the other hand, (horizontal) scaling of these datasets is a big problem, where with increasing table size or many concurrent queries, important operators like grouping or joining become slower (with bad time complexity).

[35]https://cloud.google.com/dataflow/, accessed December-2021
[36]https://aws.amazon.com/kinesis/, accessed December-2021
[37]https://azure.microsoft.com/en-us/services/event-hubs/, accessed December-2021

B. DATA LAKES

Data Lakes have essentially evolved to complement data warehouses, which in the 2010s were unable to support semi-structured and unstructured data. Data lakes can store more types and amounts of raw data than relational databases, which have several scaling issues. They are based on unlimited, cheap storage. Unstructured data such as text, documents, images, videos, etc., semi-structured data such as web server logs, streaming data from IoT, etc., and data with high variety, volume, and velocity are stored in Data Lakes (e.g. in distributed storage systems (Hadoop clusters HDFS, Ceph [88]), NoSQL databases (such as MongoDB, Apache Cassandra (inspired by Amazon DynamoDB [89]), CouchDB [90], Hbase, CosmosDB), NewSQL databases (such as MariaDB, MemSQL, VoltDB, InfluxDB, NuoDB)). Depending on the data models, NoSQL databases can also be divided into the following categories:

- **Graph databases** store data as nodes and the connections between the data are called links or relationships. Graph data is used to create whole connections of data that is representations of data elements and their mutual relationships. In networks, for example, network services can be represented as graphs to better focus on the connections between network components. Graphs are versatile and can model some systems better than representations in tabular format. Graph databases (e.g., Neo4j [91], Apache Giraph[38]) are used by analytics engines to derive more insights, values and patterns from networked behaviour. They are particularly useful for mesh connections in a data lake. Apache Spark's API GraphX is also used for graphs and graph-parallel computations. (Some use cases are social networks, knowledge graphs, etc.),
- **Key-value stores** store data as a key-value pair containing an attribute name (or ''key'') and a value (e.g., for user profiling use cases). Some examples are Memcached as an in-memory key-value store used for read performance [92] and Redis server[39] is an in-memory key-value store that can work as a cache, message broker, or database,
- **Column-oriented databases** store data as a set of columns. (for analytical use cases). Some examples are Cassandra [93] or Hbase [94],
- **Document-oriented databases** store data in formats such as JSON or XML documents. Some examples are MongoDB,[40] Elasticsearch [61], Apache CouchDB.

Data Lakes are designed to be much cheaper, easy to write, and store large amounts of data. However, it is difficult to access/read data from Data Lakes because the data may not be stored according to the analysis requirements and lacks consistency/isolation features. They also have complex system architectures due to multiple storage systems with different semantics. On the other hand, recent advances in ML libraries and data science ecosystem projects (e.g., TensorFlow) are starting to be integrated into data lakes after data preparation.

C. DATA WAREHOUSES

Data Warehouses have been widely used since the 1980s to prepare data for business analysis and decision making. They are specifically designed for SQL analysis and BI that require a well-defined schema, indexes, etc., for storage with strong management features (e.g., Atomicity, Consistency, Isolation, Durability (ACID) transaction support). Data Warehouses are basically Massively Parallel Processing (MPP) databases that can handle large amounts of data (usually structured). In a ETL workflow, data is taken from an operational data system or source such as a data lake, transformed, and placed into a data warehouse so that a materialized view of the data can be created for reports and BI. For analytics purposes, data warehouses can be used to run queries (usually written in SQL) over repositories of current and historical data to gain insights. Data warehouses are useful for long-used data that does not change or as repositories for more refined forms of data (enriched, aggregated, etc.). Some examples of data warehouse tools are Presto and Apache Hive, as well as Google BigQuery, Amazon Redshift and Snowflake for cloud native data warehouses and IBM, Oracle, Teradata for on-premise data warehouses.

D. AVAILABLE TOOLS

The following are some examples of open source databases. Apache Pinot [85] can be used as a real-time distributed OLAP datastore and also provides fast OLAP queries on large datasets with low latency. Apache Hudi[41] is a storage abstraction framework that helps enterprises build and manage petabyte-scale data lakes. Hudi enables features such as upserts and incremental pulls to absorb data changes and apply them at scale to Hudi Data Lakes. ClickHouse[42] is an open source OLAP column-oriented database, similar to Druid and Pinot, designed to aggregate as much information (on the order of several petabytes) as quickly as possible. TimescaleDB[43] is an open-source relational database for time series data, intended for query-oriented workloads and based on PostgreSQL.

Delta Lake[44] (from Databricks) is an open-source storage layer. It mainly provides ACID transactions for Data Lakes, Apache Spark and Big Data workloads/engines for interactive, batch, and streaming queries. Databricks has recently developed a new Lakehouse database paradigm that combines the benefits of Data Warehouses and Data Lakes into a single technology [95]. Iceberg[45] (from Netflix), recently released by Netflix as open source, is a new table format for storing

[38] https://giraph.apache.org/, accessed December-2021
[39] https://redis.io/, accessed December-2021
[40] https://www.mongodb.com/
[41] https://hudi.apache.org/, accessed December-2021
[42] https://github.com/ClickHouse/ClickHouse, accessed December-2021
[43] https://www.timescale.com/, accessed December-2021
[44] https://delta.io/, accessed December-2021
[45] https://iceberg.apache.org/, accessed December-2021

large, slow-moving tabular and analytical datasets. Alluxio[46] provides a distributed storage system and distributed data orchestration across hybrid clouds. For object metadata management, Hive metastore takes care of mapping from SQL tables to files and directories in the storage component. The Hive metastore service (a binary API based on the Thrift protocol) is used to update metadata stored in Relational Database Management System (RDMS) such as MySQL, MariaDB or PostgreSQL.

1) VENDOR SPECIFIC TOOLS

Amazon offers S3 for low-cost storage in Data Lakes, AWS Redshift for data warehousing and DynamoDB for database purposes, Amazon Neptune (as a graph database), AWS Redshift and Amazon Aurora (as a relational database), and AWS Glue for metadata management (similar to the Hive metastore service). GCP also offers several storage options, which are as follows: Cloud Storage (a service for storing objects, i.e., immutable data), Cloud Spanner (NewSQL database with unlimited scale, strong consistency and up to 99.999% availability), BigTable (a scalable NoSQL key-value database service for large analytic and operational workloads), BigQuery (serverless multi-cloud data warehouse), Cloud SQL (fully managed database service and can manage relational databases such as MySQL, PostgreSQL), and Cloud Datastore (NoSQL document database). Microsoft offers Azure Blob Storage, Data Lake and Cosmos DB. Other proprietary database solutions include MPP databases (Teradata, Vertica), SAP HANA, InfiniteGraph or Tableau. For metastores and their management, platforms such as DataHub, Alation and Collibra can be used. There are also two categories of solutions for Hadoop:

(i) Hadoop on-premise solutions are Cloudera Manager, Mapr, IBM InfoSphere and Hortonworks. On the other hand, due to the challenges of managing Hadoop systems on-premises, most organizations have also invested in Data Lakes in the cloud.

(ii) Hadoop in-cloud solutions. There are several reasons for organizations to move to the cloud, including flexibility, efficiency (performance, durability and cost), security, consistency, easy access to AI platforms, reduced organizational and engineering overhead, etc. Some examples of Hadoop in-cloud solutions are Microsoft Azure HDInsight, Amazon EMR, Google Dataproc, SAP Cloud Platform, Qubole.

E. PRACTICAL ASPECTS

Depending on the technical requirements, there are different ways to select databases. In the enterprise database layer, there can be a combination of NoSQL, in-memory, or relational databases may be present to take advantage of each strength depending on the use case. For example, if the data is unstructured, Data Lakes may be chosen; if the data is structured and the workload is transactional, SQL databases (in the case of single-node systems), newSQL databases (in the case of horizontal scalability), or NoSQL databases can be selected. On the other hand, if the data is structured but the workload is data analysis solutions such as MongoDB, i.e. databases offering NoSQL services (in the case latency requirements in milliseconds) or one of the data warehouse solutions described above (in the case of latency requirements in seconds) can be chosen, depending on the latency requirements.

There are also various data source formats used commonly by some Big Data processing systems and platforms to store data or exchange data. These include columnar data formats such as Parquet and ORC as well as other various data formats such as CSV, TXT, JSON, JDBS, Avro, binary files, etc. Apache Parquet is an open file format for columnar data that can be used in HDFS to store data along with its schema information and enable various I/O optimizations (e.g., compression), fast columnar analysis and aggregation. It is the default data source for many Big Data processing frameworks and platforms, including Apache Spark for analytics workloads. The open file format of Avro data storage is also used in Apache Spark and Apache Kafka when (de)serializing messages. The Avro file format is row-oriented (as opposed to Parquet). It is a framework for serializing data and can provide direct mapping to JSON, which improves the speed and efficiency of data processing.

Data applications often access and use hot/warm storage databases (e.g., Redis) for real-time access, while cold event data is stored in lower-cost storage devices (e.g., cloud storage, HDFS). Depending on the storage characteristics, there are also different types of durability. **In the persistent type**, the data is not lost even if the cluster fails completely. **In replicated type**, the data is not lost even if a limited node in the cluster fails and **in transient type**, the data is lost in case of failures. In most data processing frameworks, a sharded and persistent database is required that can provide a control store database responsible for storing metadata such as task specifications, task dependencies, or critical system information to recover from failures (i.e., for fault tolerance purposes). When a node in the cluster fails and critical information is in danger of being lost, this datastore is used to regenerate the data by re-executing the tasks required for lineage-based fault tolerance. These specifications are typically stored in a control store database (e.g., ZooKeeper in the Hadoop ecosystem [96] or the global control store in Ray [78], a cluster of Redis databases).

Finally, note that data storage can also be costly as data accumulated over years. Therefore, it is desirable to store more processed smart data than more large raw data, unless required by regulation.

VI. DATA MONITORING AND VISUALIZATION FRAMEWORKS

Monitoring and visualization of data plays an important role in the world of telecommunications. With proper monitoring and visualization, it is easy to uncover insights and patterns, understand relationships between observations, or describe

[46] www.alluxio.io, accessed December-2021

trends or seasonality in telecommunication data. Data visualization is traditionally used for regular performance reporting to fully explain and present the results of data analysis or the data itself. It can also serve as an interface for users to run or compile analytics on data processing and analysis frameworks (e.g., queries against loaded data) and visualize the results. Many business decisions are made on dashboard-based pipelines through daily monitoring and interpretation of the data itself. Notification services and alerts, monitoring dashboards using tools like Grafana[47] or Kibana [61], business intelligence dashboards (drill downs, top K results), ad hoc query clients/notebooks like Jupyter, heatmaps and user feedback can all be done during the data presentation or visualization stage.

Some of the open source tools and frameworks available for data visualization are as follows. For exploring, visualizing and discovering data through dashboard visualization, Kibana from ELK stack [61] provides a free and open user interface. Grafana is used as an analytics and monitoring tool that allows to monitor infrastructure, applications, and metrics by connecting to databases. Kiali provides [97] dashboards, service mesh observability. Gephi[48] provides an open graph visualization platform. Dash[49] is a production Python framework for building web applications and data visualization apps using Flask, Plotly.js, and React.js. Apache Superset[50] is an open-source visualization tool developed by Airbnb that is used to visualize analytics results (with chart types such as word counts, heat maps, boxplots, etc.). Apache Superset can also be used for queries with Apache Druid. Apache Druid [59] serves as an analytics database, but can also be used as a dashboard for quick results on complicated analytics tasks. As a data application framework, Streamlit[51] can be used to easily create ML and data science web applications. Metabase[52] is another open source tool for business intelligence purposes.

To monitor data pipelines, entire infrastructure, or cloud native applications, Prometheus [98], Datadog,[53] Sentry[54] provide monitoring services for servers, databases, tools, and cloud applications. One of the most popular cloud native monitoring tools is Prometheus. Prometheus can identify the applications to be monitored (either 3rd party or on-premise) through its service discovery feature, which can be scrapped through Exporters. The scraped data can be stored in local storage to be queried with PromQL or visualized with Grafana. Prometheus also allows sending alerts to various notification systems such as email, chat systems, etc. based on the configured alert rules.

[47]https://grafana.com/, accessed December-2021
[48]https://gephi.org/, accessed December-2021
[49]https://dash.plotly.com/, accessed December-2021
[50]https://superset.incubator.apache.org/, accessed December-2021
[51]https://www.streamlit.io/, accessed December-2021
[52]https://github.com/metabase/metabase, accessed December-2021
[53]https://www.datadoghq.com/, accessed December-2021
[54]https://sentry.io/welcome/, accessed December-2021

1) VENDOR SPECIFIC TOOLS

For reports and dashboard outputs, Tableau, Looker, and Mode; for embedded analytics outputs, Sisense; for advanced analytics outputs Thoughtspot, Outlier Analytics, Anodot, Sisu; for building ML and data science web application framework, Plotly Dash; for data visualization, Google's Cloud Datalab,[55] TIBCO Spotfire, MicroStrategy, Zepl, SAP's Lumira, Microsoft's Power BI; for running high-performance queries on petabytes of structured data to create powerful reports and dashboards Amazon Redshift and Vertica; and for monitoring applications and infrastructure in the cloud, Amazon's CloudWatch, Google's StackDriver, and Microsoft's Azure Monitor can be used.

VII. DATA ORCHESTRATION AND MANAGEMENT FRAMEWORKS

The use and popularity of microservices and cloud/container orchestration frameworks is increasing due to various benefits such as service discovery or easy horizontal scaling. There is no sign that the ecosystem of data engineering and infrastructure is coalescing into a unified, manageable form. Over time, new distributed databases, frameworks, platforms, and libraries will be introduced into the ecosystem. Because of this, data management frameworks can bring everything together with suitable APIs as these systems evolve into more complex structures day by day. For example, stream processing jobs are more akin to microservices and thus require support for managing services and applications including cluster management, debugging and continuous monitoring. Note that approaches such as centralized management approaches (e.g., scheduling) can create a bottleneck in the system that can only provide a finite amount of throughput (i.e., in terms of processing capabilities of tasks per second). Therefore, the importance of automated and distributed resource management, scheduling, and orchestration is steadily increasing in the modern data ecosystem. The main goal is to start or manage computational resources, services or containers with a single or a set of API calls to reduce operational costs and complexity.

In data applications, all computations consume and produce data which must be orchestrated in a data-aware manner. Several open-source distributed frameworks are available for distributed orchestration, seamless distributed execution, workflow definition and execution, resource management, or resource-aware scheduling of clusters, data centers, or cloud environments. These frameworks are mainly used with applications (e.g., Kafka, Hadoop, Elasticsearch, Spark, etc.) and provide APIs for resource management, orchestration, elasticity, high availability (or zero downtime), fault tolerance, and scheduling. Traditional CRON jobs and other similar solutions such as configuration files are loosely coupled and do not allow users to formally define dependencies. Recently developed workflow engines, on the other hand, allow dependency graphs, proper execution order,

[55]https://cloud.google.com/datalab, accessed December-2021

consolidated logs, expression of tasks using operators, and use data pipelines/workflows as a code paradigm.

A. SCALING

The number of data sources that an organization can ingest and process is growing much faster than the number of resources for data engineering. In addition, the variance of data traffic in production systems can be unexpectedly high at certain times. For these reasons, the system should be scalable without compromising throughput and latency. Scalability is the ability of the system to provide a moderate performance as the load increases (e.g., high volume, high throughput or high velocity data). Scalability can refer to different dimensions: Processing, Serving or Storage. A scalable architecture means that the system should continue to function smoothly even if 1 user or 1 million users access the application. For systems, there are two ways to scale a database:

1) **Vertical scaling (or Scaling up):** is done by improving the Central Processing Unit (CPU), Random Access Memory (RAM), and storage capacity of the existing machine in hardware or by optimizing algorithms and application code in software. This eventually reaches its limits when the amount of data on a single machine grows.
2) **Horizontal scaling (or Scaling out):** is done by adding additional machines to the database cluster. Note that requests in this case not only consume CPU, but also require network resources. In this case, it is important to shard the data so that a single query can be processed on a single machine. Due to difficulties in horizontal scaling with traditional SQL databases, NoSQL and NewSQL databases with horizontal scaling options have emerged. However, they lack strong consistency guarantees and relational models (see Section V for details).

The complexity of the scaling process depends on the type of service that infrastructure provides. There are two types of services:

- **In stateless services**, scaling is usually done based on user traffic and on-demand, e.g., web server applications. Deploying and scaling stateless services is relatively straightforward.
- **In stateful services**, consistency of user data across data centers is critical for scaling, e.g., for scaling databases. Deploying and scaling stateful services is more difficult and complex because copies of the database need to be managed in different data centers. A possible solution to reduce complexity would be to partition the database and enable different replication factors in different locations so that the overall replication factor can be reduced. This would also help reduce traffic between data centers.

B. PARALLELISM

To increase processing capacity, most data engineering pipeline frameworks exploit the potential of parallelism. Parallelism can be either over data or task selected according to the application. In data parallelism, input data is partitioned to scale processing (e.g., in low-cost HDFS). In task parallelism, multiple tasks (CPUs) are executed over the same data. In event processing, there are various forms of shuffling and data exchange capabilities within parallel computing frameworks. These mainly fall into the following categories:

- **Forward or one-to-one:** In this case, the data is processed with the same node and there is no data exchange between the nodes of the cluster.
- **Broadcast:** In this mode, data is broadcast to all nodes because it must be used by the tasks running on each node.
- **HashKey:** In this case, the data is collected/grouped by key to elegantly distribute the data among the work tasks of the nodes in a cluster.
- **Rebalance or random:** In this case, random repartitioning is used when not much is known about the data. The goal is to balance the distributions between the nodes to increase the data processing capabilities of the nodes.

C. MICROSERVICES AND DEPLOYMENT

Microservices and microservices-based architecture are some of the recent trends that enable flexibility and scalability. Data orchestration and management teams are adapting them to run and manage distributed application components. Service mesh architecture enables microservices to communicate with each other. Typical types of communication between microservices can be divided into three dimensions:

1) **Event-driven** based on platforms such as the message/event bus,
2) Orchestration based on **REST callouts** using some frameworks such as Apache Camel,
3) Orchestration based on some **workflow engines** like Apache AirFlow.[56]

In microservices architecture, different deployment patterns are possible [99]:

1) **All-at-once deployment:** Making changes on top of existing configurations, terminating the old version and releasing the new version. So, the deployment is immediate. This pattern is better suited for workloads with low concurrency.
2) **Blue/Green deployment:** Allows traffic to be moved to a new live environment (green) while the old production environment (blue) is still kept warm. Later, the transition to the new version can take place. This pattern is better suited for workloads with medium concurrency.
3) **Canary/Linear Deployment:** Deploying a small number of requests for a new software version to analyze the impact on a small number or subset of users. Then, full rollout can be done. This pattern is better suited for workloads with high concurrency.

[56]https://airflow.apache.org/, accessed December-2021

Kubernetes also has similar rollout deployment strategies such as Recreate, Ramped, Blue/Green, Canary, A/B testing and Shadow.[57]

D. CI/CD AND IaC

Continuous Integration (CI)/Continuous Development (CD) are terms that have been common in the industry for a decade. Platform engineers and DevOps teams are continuously developing infrastructure tools to improve engineer productivity. At the same time, data applications should also be flexible enough to be deployed by CI/CD platforms for testing and development purposes. For data engineering applications, integration can be done in three modes:

1) **Data integration:** This is related to data movement patterns. It can take place either at the node level or at the cloud level, and ETL is an example of this. Data integration is challenging as it has to deal with heterogeneous data sources with different sampling rates and data generation models.
2) **Application integration:** uses APIs such as REST, SOAP, etc. between applications and their internal subscriptions. In application integration, the amount of data exchange is not as large as data integration.
3) **Event integration:** combines the benefits of application and data integration, where a significant amount of data is also exchanged when events are triggered between applications.

Infrastructure as a code (IaC) enables scalable infrastructure deployment via standardized interfaces such as YAML or JSON files. IaC can be implemented via containers (e.g. via Docker, LXC [100]), container orchestration (e.g. via Kubernetes [101], Docker Swarm [69], Apache Mesos [102]) and infrastructure provisioning (e.g. via Terraform). Two different models for IaC or automation are:

1) **Declarative model** requires the user to define a desired state to be provisioned by Kubernetes. The relationship between the infrastructure and the application is declarative. Applications make declarative requests to the infrastructure (e.g., via YAML, a data serialization language and easily understood by humans and machines) with implementation details abstracted by the underlying framework (e.g., the Kubernetes cluster). One of the main advantages of this model is that once the plan is created, it is the responsibility of the framework to develop and execute the work plan in a way that is optimized for the complexity of the infrastructure. This ensures the transition from the *infrastructure as code* paradigm to the *infrastructure as data* paradigm. Any new application can be easily enriched using the declarative tag of the declarative YAML management file.
2) **Imperative model** requires users to specify commands or plans in a specific order to achieve and maintain a desired state, such as the service provided by Apache AirFlow. Although this model provides flexibility, imperative models have scaling issues as the number of components increases and complexity grows exponentially.

E. AVAILABLE TOOLS

For highly reliable distributed coordination of multiple machines in a cluster, distributed key-value stores such as Apache Zookeeper [96], etcd,[58] Consul[59] provide reliable data stores for distributed system access. They can perform functions such as leader elections, fault tolerance for machine failures, network configuration automation, and service discovery. To ensure strong consistency and replication, consensus algorithms and protocols such as Raft consensus algorithms and Paxos protocols are used to determine the order in which data is stored and when it becomes visible to users. Data management tools such as Deequ[60] are particularly useful for dealing with corrupt or bad data in large datasets. MLflow [103], is an open source ML lifecycle platform, is used to manage the E2E ML lifecycle so that model-based experiments and quality metrics can be managed, tracked, and reproduced. Kubeflow [104] provides a data automation framework for Kubernetes clusters and enables connectivity with a variety of databases and services. This enables a highly modular design. Weights & Biases[61] is another related developer tool for ML. These ML lifecycle management frameworks act as CI/CD tools in the ML domain.

Some of the most popular systems for defining workflows and programmatically scheduling jobs/tasks are Apache Mesos [102] and Apache YARN [43] (for running the cluster and monitoring executed jobs, e.g. Cloudera/Hortonworks run Spark jobs using YARN), Apache AirFlow (provides workflow-level abstraction for building data pipelines using Directed Acyclic Graphs (DAGs)), Dagster[62] (for data orchestration), Prefect[63] for automating data flows and creating, executing and monitoring millions of data workflows and pipelines, Spotify's Luigi,[64] LinkedIn's open source scheduler Azkaban,[65] and Apache Oozie [105] (for distributed coordination and scheduling of workflows in Hadoop clusters), Apache TEZ [106] (for building complex directed acyclic graphs of tasks to process data), Apache Ambari[66] (provides a management interface), Kubernetes [101] (provides high flexibility as the dominant container orchestration framework), OpenShift[67] (provides a web console to run

[57] https://github.com/ContainerSolutions/k8s-deployment-strategies, accessed December-2021

[58] https://etcd.io/, accessed December-2021
[59] https://www.consul.io/, accessed December-2021
[60] https://github.com/awslabs/deequ, accessed December-2021
[61] https://www.wandb.com/, accessed December-2021
[62] https://dagster.io/, accessed December-2021
[63] https://www.prefect.io/, accessed December-2021
[64] https://github.com/spotify/luigi, accessed December-2021
[65] https://azkaban.github.io/, accessed December-2021
[66] https://ambari.apache.org/, accessed December-2021
[67] https://www.openshift.com/, accessed December-2021

tasks directly), Yunikorn[68] (resource scheduler for containerized systems), and Docker Swarm. Apache Calcite [107] is a dynamic data management framework and is used to as an intermediary between applications and data stores and data processing engines. Amundsen[69] (from Lyft), Metacat[70] (from Netflix), DataHub[71] (from Linkedin) provide both metadata management and data discovery options to improve the productivity of data professionals interacting with data. Some of these tools (such as Kubernetes) are also important tools for managing and deploying containers (e.g., container lifecycle and resources, or facilitating application development through container orchestration) in the context of network operations and management in 5G networks.

Some common IaC and automation tools that can help eliminate the risk of human error, increase the speed of code development speed (by reliably building, testing, and deploying software), and reduce costs are Ansible,[72] Chef[73] and Puppet[74] for configuration management, Terraform[75] (infrastructure deployment orchestration tool) and Jenkins,[76] GitHub Actions[77] as part of the CI/CD chain. These tools have significantly improved the way applications and workflows are deployed and managed within the infrastructure through configuration (installing packages, configuring servers, and deploying applications to the infrastructure) and orchestration of the infrastructure. Great expectations provides data quality, documentation, profiling and testing.[78] Finally, the literature also provides an excellent curated list of data pipeline frameworks, libraries[79] and workflow engines.[80]

1) VENDOR-SPECIFIC TOOLS

Amazon offers AWS Elastic Beanstalk, Amazon ECS (as a container orchestration framework), and AWS CloudFormation for infrastructure provisioning and IaC tool, CodeDeploy for deployment, CodePipeline for unit and integration testing, CI/CD pipeline, Microsoft provides Azure Resource Manager (ARM) and Pipelines for the deployment and management service and CI/CD Pipeline over Azure, TestPlans for unit and integration testing, Google provides Google Composer as a managed Apache Airflow service on GCP and Google Kubernetes Engine (GKE),[81] Deployment Manager for infrastructure automation, Cloud Build for deployment, unit and integration testing, CI/CD pipeline, Puppet Enterprise also offers Puppet as configuration management, HashiCorp offers Terraform for deploying and managing any cloud, infrastructure or service. Pulumi provides a modern Infrastructure as Code for building, deploying, and managing infrastructures in any cloud. For visibility and observability of data pipelines, Unravel, Fiddler and Acceldata can be used. For data modeling and analytical engineering workflow, dbt (Data Build Tool) and LookML can also be used to transform data into data warehouses more efficiently.

Considering all the above descriptions, starting from data connection to data orchestration, Table 5 shows the summary of characteristics of open source frameworks in the data engineering pipeline and the corresponding related works. In addition, some of the most important tools and their connection to the components of the data engineering pipeline framework are described in Fig. 2.

VIII. RELATION WITH DATA SCIENCE FRAMEWORKS

With the advent of DL and new computational workloads, scaling and distributed computing are becoming increasingly important in AI/ML. Data Engineering also helps data science developers build distributed computing frameworks so that they can focus on developing their own AI/ML algorithms instead of dealing with the intricacies of distributed computing. On the other hand, AI/ML based platforms build on Data Science libraries play an important role. For example, in a processing pipeline, data scientists work on defining and preparing the model, and data engineers implement the aspects that serve the model. For this reason, the ability to interact with AI/ML models (export, import, etc.) from data science tools is important. In the ecosystem, there are several open source platforms for AI/ML. Model training and online prediction, inference layers can be done through these special projects developed for a number of different important workloads.

A. DATA SCIENCE FRAMEWORKS

Some of these related Data Science projects and tools can be summarized as follows: For building **Machine Learning** applications, Spark MLlib (Apache Spark's scalable machine learning library) [110], statistical modeling libraries (scikit-learn (machine learning in Python) [111] and statsmodel[84]), Apache MADlib (Big Data Machine Learning in SQL) [112], Aerosolve[85] (a machine learning package designed for humans), Mahout[86] for developing scalable, performant ML applications and Knime[87] as an open-source platform for data analytics.

[68] https://github.com/apache/incubator-yunikorn-core, accessed December-2021
[69] https://www.amundsen.io/, accessed December-2021
[70] https://github.com/Netflix/metacat, accessed December-2021
[71] https://github.com/linkedin/datahub, accessed December-2021
[72] https://www.ansible.com/, accessed December-2021
[73] https://www.chef.io/, accessed December-2021
[74] https://puppet.com/, accessed December-2021
[75] https://www.terraform.io/, accessed December-2021
[76] https://www.jenkins.io/, accessed December-2021
[77] https://docs.github.com/en/actions, accessed December-2021
[78] https://github.com/great-expectations/great_expectations, accessed December-2021
[79] https://github.com/pditommaso/awesome-pipeline, accessed December-2021
[80] https://github.com/meirwah/awesome-workflow-engines, accessed December-2021
[81] https://cloud.google.com/kubernetes-engine, accessed December-2021
[84] https://www.statsmodels.org/stable/index.html, accessed December-2021
[85] https://github.com/airbnb/aerosolve, accessed December-2021
[86] https://mahout.apache.org/, accessed December-2021
[87] https://www.knime.com/, accessed December-2021

TABLE 5. Characteristics of open-source frameworks in data engineering for networking and corresponding related works.

Frameworks		Characteristics	Related Works
Data Connection		— API service to receive data from its sources. — Used to put data into temporary storage places (e.g. queues, databases)	Kafka Connect, Flume NiFi, Falcor REST APIs, GraphQL
Data Ingestion		— Act as distributed pub-sub messaging system — Enables ingestion of data into cluster — Manage data (transformation and enrichment) — Messaging system (asynchronous and in real-time) — Avoids overwhelming the data pipeline	Spark Streaming, Gobblin RabbitMQ, Heron Kafka, Pulsar Storm, RocketMQ Flink DataStream
Data Processing & Analytics	**Processing**	— Analyze data in batch, interactive, streaming or real time — Model/Inference/Prediction Serving — Distributed data structures (RDDs, DataStreams and DataSets)	Map-Reduce, Tez Spark, Flink Samza, Beam
	Log Analytics	— Continuous log stream processing (parsing, joining, augmenting) — Real-time application performance monitoring — Generate new derived data as the results of queries from logs, metrics, web applications, data stores	Logstash, Splunk Timber, Prometheus Sentry, Fluentd* Filebeat, Heka, Fluent Bit
	Query	— Enables query capabilities over big data (e.g. Hadoop cluster), NoSQL databases and cloud storage — Provide OLAP capability to big data — Supports SQL (or a subset) functionality — Streaming, batch and hybrid modes	Hive, Pig Spark SQL [55], Presto [83] Flink SQL [76], Drill [82] Kylin [84], Arrow Pinot [85], Samza SQL
	Search	— Execute search engine over big data	Elasticsearch, Solr [108]
	Analysis	— Classification, Clustering, Regression, — Deep Learning, Distributed Model Training. — Distributed Reinforcement learning, Hyperparameter search	TensorFlow, Deeplearning4j Spark MLlib [55], PyTorch [109] BigDl, RLlib
Data Storage		— Graph databases — In-memory and analytics databases — Distributed storage systems	Neo4j, Giraph Redis, ClickHouse HDFS [41], Ceph [88], Hbase [94]
Data Visualization and Monitoring		— 1D, 2D and 3D visualization — Temporal and spatial analysis — Monitoring pipelines, dashboards, alerts and notifications	Kibana, Grafana Superset, Druid Graphite†, Dash
Management, Orchestration and Scheduling		— Resource management, defining and scheduling workflows/tasks and services across cluster, data center and cloud —Container orchestration, auto-scaling	YARN, MESOS, AirFlow Kubernetes, Docker Swarm YuniKorn, LXC

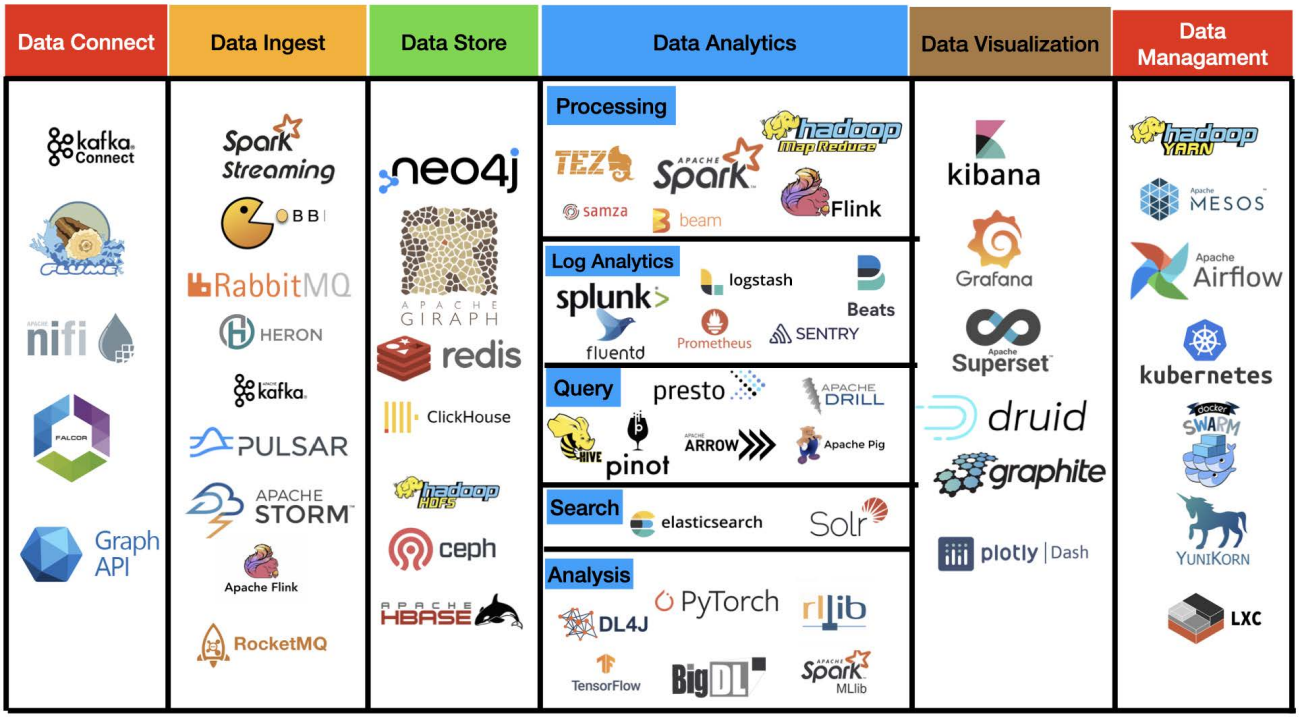

FIGURE 2. Some key data engineering tools and their connection to the components of the data engineering framework.

To build and train models with **DL**, libraries and frameworks such as TensorFlow (an E2E open source machine learning platform) [113], Deeplearning4j (DL for Java), Torch (an open-source machine learning library based on C) [114], PyTorch (an open-source machine learning framework that accelerates the path from research

prototyping to production deployment) [109], Chainer [115], a Python-based DL framework that aims for flexibility and intuitiveness over neural networks, Sonnet[88] a library built on TensorFlow 2.0 to provide simple, composable abstractions for ML research, BigDL [77], a distributed DL framework for Apache Spark, Apache Singa [116], which focuses on distributed training of DL and machine learning models, Apache MXNet [117], an open-source DL framework suitable for flexible prototyping in research and production, Deeplearning.scala,[89] for building complex neural networks, Sparkflow,[90] an implementation of TensorFlow on Spark, Theano [118], Caffe [119], Keras [120], PaddlePaddle,[91] ONNX[92] to support DL model creation and deployment, Microsoft's Cognitive Toolkit (CNTK)[93] for distributed DL and more recently Ludwig,[94] JAX[95] and Trax[96] are used.

For developing **Reinforcement Learning** applications, Ray provides (a fast and simple framework) RLlib for building and running distributed, parallel, scalable reinforcement learning-based applications [121], Stable Baselines[97] to produce a set of improved implementations of reinforcement learning algorithms based on OpenAI Baselines, Garage[98] to develop and evaluate reinforcement learning algorithms, Coach,[99] a Python-based reinforcement learning framework that includes implementations of many state-of-the-art algorithms, Tensorforce,[100] a TensorFlow library for applied reinforcement learning, ChainerRL, a deep reinforcement learning library that includes several state-of-the-art deep reinforcement algorithms in Python with Chainer [115] (a flexible DL framework), OpenAI [122]'s Gym, Retro and Neural MMO frameworks (environments for training agents with reinforcement learning, e.g. DotA as an application), Unity's ML agents, Microsoft's Project Malmo and DeepMind's Lab and Control Suit (e.g., 3D learning environment, RLax library,[101] AlphaGo as an application) are also developing their own new distributed reinforcement learning algorithms to be implemented or building their own infrastructure/tools for their applications to achieve the required flexibility and performance.

For **Distributed Training** frameworks such as Horovod [123] along with TensorFlow, Keras, PyTorch, and Apache MXNet, Distributed TensorFlow, for **Model Serving** (takes the trained model and sends predictions or recommendations to specific applications) tools/frameworks such as Clipper [124], which is used for low-latency prediction serving systems for ML when integrated with client systems, TensorFlow Serving, TorchServe,[102] Ray Serve, or Seldon[103] can be used depending on use case that requires low-latency model deployment for large-scale prediction services. For **Hyperparameter Search** (either via manual search, grid search, Bayesian optimization, evolutionary optimization or random search), Advisor[104] (which is an open source implementation of Google's Vizier) is used for the hyperparameter tuning system for black-box optimization, Hyperopt [125] and Tune[105] are used for distributed hyperparameter optimization and scalable hyperparameter tuning, respectively. For **NLP**, tools such as spaCy [126], Hugging Face [127], AllenNLP [128] and more recently GPT-3 [129] can be used.

1) VENDOR-SPECIFIC TOOLS

Proprietary software such as SAS, Datatron, ModelOp, etc. can also be integrated with various products and services for ML and model serving purposes. Some companies like Cloudera, Databricks, Dataiku, Domino Data Lab, etc. have also been offered data science workbenches/platforms as ML service.

B. MACHINE LEARNING PLATFORMS IN INDUSTRIAL ENVIRONMENTS

Several startups and cloud companies offer E2E ML tools and platforms, including IT and cloud giants Google, Amazon and Microsoft. The following is an overview of ML platforms used in industrial environments:

- Uber uses Michelangelo [75] as its internal ML-as-a-service platform. It consists of open-source components such as HDFS, Kafka, Spark, Samza, Cassandra with libraries such as MLLib, XGBoost and Tensorflow.
- Airbnb uses Bighead, a combination of Zipline data management tool, the containerized Jupyter notebook service Redspot, and the Bighead library for data pipeline abstractions, transformations, and data track lineage.
- Netflix uses Metaflow,[106] a Python-based framework built on AWS, to handle model training and data management by running DAGs on an AWS Serverless Orchestration platform.
- Lyft has open-sourced its cloud native platform called Flyte,[107] that can invoke machine learning and operations together, termed as ML operations (MLOps).
- Amazon offers SageMaker as a complete solution for ML with the latest libraries such as TensorFlow, Keras,

[88] https://github.com/deepmind/sonnet, accessed December-2021
[89] https://github.com/ThoughtWorksInc/DeepLearning.scala, accessed December-2021
[90] https://github.com/lifeomic/sparkflow, accessed December-2021
[91] https://github.com/PaddlePaddle/Paddle, accessed December-2021
[92] https://onnx.ai/, accessed December-2021
[93] https://docs.microsoft.com/en-us/cognitive-toolkit/, accessed December-2021
[94] https://github.com/uber/ludwig, accessed December-2021
[95] https://jax.readthedocs.io/en/latest/notebooks/quickstart.html, accessed December-2021
[96] https://github.com/google/trax, accessed December-2021
[97] https://stable-baselines.readthedocs.io/en/master/index.html, accessed December-2021
[98] https://github.com/rlworkgroup/garage, accessed December-2021
[99] https://github.com/NervanaSystems/coach, accessed December-2021
[100] https://tensorforce.readthedocs.io/en/latest/, accessed December-2021
[101] https://github.com/deepmind/rlax, accessed December-2021
[102] https://github.com/pytorch/serve, accessed December-2021
[103] https://www.seldon.io/, accessed December-2021
[104] https://github.com/tobegit3hub/advisor, accessed December-2021
[105] https://docs.ray.io/en/latest/tune.html, accessed December-2021
[106] https://metaflow.org/, accessed December-2021
[107] https://lyft.github.io/flyte/, accessed December-2021

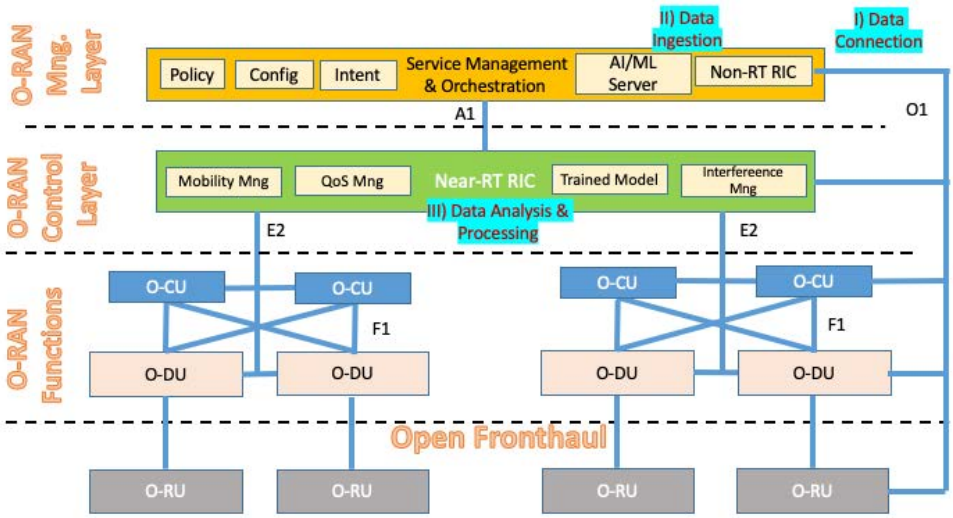

FIGURE 3. A high-level illustration of the O-RAN architecture with RAN intelligent controllers (near real-time and non real-time), control and distributed units, and mapping with data engineering pipeline components.

PyTorch, and MXNet and model deployment options in the cloud or at the Edge.

- Microsoft uses Azure Machine Learning Studio which supports a full range of frameworks including TensorFlow, Keras, PyTorch, MXNet scikit-learn, and XGBoost.
- IBM uses Watson ML to support various frameworks on both CPUs and GPUs in collaboration with its own products.
- Databricks uses MLflow [103], an open source platform for managing the lifecycle of ML which includes four components: MLflow Tracking (for recording and querying experiments), Projects (for packaging code and running it on any platform), Models (for deploying ML models across environments) and Model Registry (for discovering, storing, annotating, and managing models through a central repository).
- Intel Analytics Zoo,[108] which provides an E2E analytics and AI platform (high-level pipeline APIs, integrated DL models, etc.), is available as open source.
- Apple provides Overton [130] to build, monitor and improve ML systems in production environments.
- Facebook uses FBLearner [131] as a ML platform to automate tasks such as training on clusters and developing custom ML code.
- Google offers the Cloud AI Platform[109] to develop AI applications and run them both on GCP and on-premises.

IX. NETWORK SERVICE MANAGEMENT AND ORCHESTRATION OVERVIEW

Thanks to advances in network services, new applications such as the tactile internet, holographic-type communications and teledriving are expected to emerge in the next decade. Many of these E2E services also require high levels of precision which has significant implications for the management of these networks and services. Managing and maintaining distributed computing functions in a network environment with elevated levels of service requirements requires hundreds of operations at any given time. This makes the human-centric and standard network service management and operation solutions already used inadequate and ineffective.

Therefore, automation of network management and service deployment is critical and there is a need for unified network Lifecycle Management (LCM) and orchestration across multiple administrative and technological domains. Network service orchestration enables network operators to connect and configure systems and multiple network elements through a optimized workflow. This enables the delivery of optimal services to users and contributes to automation by coordinating interactions and service flows across multi-domain, multi-layer, multi-vendor networks. Approaches that rely on intelligent automation of network operations (e.g., via the emerging field of Zero Touch network and service management (ZSM) [132]) can make a big difference when multiple network functions need to be owned, maintained, and operated at scale throughout the network.

A. DIFFERENT ASPECTS OF NETWORK SERVICE LCM

Network service LCM addresses the necessary operations to create, deliver, manage and orchestrate network services to meet the diverse needs of end users and enterprises over network infrastructures. The goal is to guarantee autonomic network service assurance and dynamic service delivery. Some of these operations include network functions deployment, provisioning, onboarding, updating on the fly, storing, ensuring zero downtime, demand-based scaling in an intelligent

[108] https://analytics-zoo.github.io/0.2.0/, accessed December-2021
[109] https://cloud.google.com/ai-platform, accessed December-2021

and cost-efficient manner, supplying suitable infrastructure resource orchestration, anomaly detection at run time. These operations are achieved via software-based/virtualized network functions or cloud native microservices deployed across fog/edge/cloud infrastructure.

The ETSI-defined cross-domain E2E network service LCM is divided into three main processes (see [133] and references therein):

- *Service on-boarding* procedure is used to add new service model to the E2E service management catalogue. Some examples of service management include E2E service orchestration (to control the service model and maintain the service catalogue), domain orchestration (to send alerts when changes are made to the catalogue and to request missing entries in the catalogue), ZSM integration fabric (to manage subscriptions, data generation and consumption) and ZSM data services (to store data and provide data persistence).
- *Service fulfilment* procedure is used to manage E2E service instances from instantiation to termination. The following processes are provided for the provisioning of E2E service instances. (i) Service instantiation (creates an E2E service instance), (ii) Service activation (activates an E2E service instance), (iii) Service configuration (modifies the configuration of an E2E service instance), (iv) Service deactivation (deactivates an E2E service instance), (v) Service decommissioning (removes an E2E service instance and releases all its resources), (vi) Update E2E inventory (keeps up-to-date information about resources and domain service instances).
- *Service assurance* procedure is used to ensure that E2E service level requirements are met. The following processes are provided to deliver E2E service assurance. (i) Service assurance set-up (assures an E2E service), (ii) Service quality management (manages service quality), (iii) Service problem management (investigates cross domain service problems), (iv) Service assurance tear-down (defines procedures to tear down the collection of information).

B. MANAGEMENT PLATFORMS

In this section, we describe two of the most popular management and orchestration platforms.

1) ONAP

Open Network Automation Platform (ONAP) project leverages SDN and NFV technologies to improve network service deployments and provisioning.[110] It is designed to provide a unified framework for monitoring solutions to observe and verify E2E SLAs and Key Performance Indicators (KPIs). ONAP provides a scalable and distributed approach to managing multi-site and multi-Virtualized Infrastructure Manager (VIM) resources. This is achieved through the components of the so-called Data Collection, Analytics and Events (DCAE) module, which can be geographically distributed across multiples sites and hierarchically interconnected. Data monitoring at different levels is usually done with open-source software such as Prometheus, which can distribute its functionalities across geographically separated sites.

Synchronization messages and local processing results are exchanged between different sites using a submodule called Data Movement as a Platform (DMaaP). This submodule supports both file-based and message-based data exchange via the publish & subscribe paradigm.

2) OPEN SOURCE MANO

Open Source MANO (OSM) is a collaborative open source project to develop an NFV Management and Orchestration (MANO) stack that is conforms to the European Telecommunications Standards Institute (ETSI) NFV Information Models and APIs.[111] The focus is on the Network Service Orchestrator (NSO) part of ETSI MANO NFV Orchestrator (NFVO). Network slice support, LCM of Network Slice Instances (NSIs), monitoring capabilities including Virtual Network Function (VNF) metrics collection are some features of recent OSM releases. For deployment and management, OSM can operate in two modes, namely Full E2E Management (Integrated Modelling) and Standalone Management(Vanilla NFV/3GPP) [134].

X. NETWORK MANAGEMENT AND ORCHESTRATION IN STANDARDIZATION

In recent years, several standards organizations, independent alliances, and forums have been involved in developing standards for developing platforms that work with AI and ML. In addition, zero-touch network management and orchestration frameworks, which are based on significantly simplifying the tasks performed by human to manage and orchestrate network slices, are currently being extensively researched by standardisation bodies. Standards organizations such as Open Radio Access Network (O-RAN), ETSI, The 3rd Generation Partnership Project (3GPP) are working on embedding intelligence into emerging next generation architectures to efficiently meet the diverse needs of communication network users. In this subsection, we provide an overview of some of the work that has been done in these SDOs to build a AI/ML platform. A good overview of existing standardization efforts related to AI for 5G systems as well as some of the identified gaps in standardization, are also summarized in [135].

A. O-RAN'S AI/ML ARCHITECTURE

O-RAN alliance aims to define a next-generation radio access network (RAN) infrastructure based on software-defined technology and general-purpose hardware, driven by both intelligence and openness at every layer of the RAN architecture [136]. O-RAN currently offers an attractive solution for

[110] https://www.onap.org/, accessed February-2022

[111] https://osm.etsi.org/, accessed February-2022

creating next-generation multivendor networks that embrace the concepts of programmable, open, collaborative, and intelligent communications. Therefore, AI and ML are the main pillars for the realization of O-RAN.

O-RAN high-level architecture can be divided into two layers, namely Service, Management and Orchestration (SMO) and radio access site as shown in Fig. 3 [137]. In the radio access entities, there are RAN intelligent controllers (RICs) (near real-time (near-RT), non real-time (non-RT)), (the vertically divided control (CP) and user (UP) planes of the central units (CU)) as well as open interfaces that interconnect the O-RAN nodes. Both near-RT and non-RT controllers are introduced to extend the existing network functions with more embedded intelligence within the O-RAN architecture. Near-RT RIC is interfaced with a centralized unit control plane (CU-CP) for transmission of signals and configuration messages and centralized unit user plane (CU-UP) for data transmission and can be used for control loops on the order of ms time scale. Non-RT RIC is interfaced with near-RT RIC via interface A1 (for policy management and coordination) and to the CU-CP, the Distributed Unit (DU) and the Radio Unit (RU) via interface O1 and can be used for control loops on the time-scale in the order of greater than 500 ms [138]. In addition, applications (xApps/rApps) are introduced to be hosted either on the near-RT RIC or on the non-RT RIC depending on how sensitive the applications are to control processes.

In the architecture, different interfaces (O1, A1 and E2) are used depending on the results of the AI/ML algorithms, the actions and the actors. For example, the O1 interface is used to configure Control and Data Units and near-RT RIC for fault and performance management. A1 interface enables non-RT RIC to provide RAN optimization functionalities to near-RT RIC functions. These functionalities include policy management, ML model management, or data enrichment. The E2 interface is used for communication between near-RT RIC and centralized/distributed units of RAN in the O-RAN architecture. AI/ML algorithms can run on top of near-RT RIC (e.g., xApps for energy, resource or beamformer optimization, traffic steering) or the non-RT RICs (e.g., rApps for RAN automation applications such as network deployment, optimization, frequency band selector). These algorithms can also be reconfigured based on data availability, control timescales, network load, and overall mobile operator requirements.

Data pipeline generation via O-RAN architecture: To enable automated and intelligent network functions, O-RAN architecture has been standardized to include three types of control loops (categorized by the time sensitivity of the required decision-making process) and AI/ML dedicated nodes. The first control loop is responsible for scheduling at the Transmission Time Interval (TTI) level and operates on a time scale of TTI ms or above. The second control loop operates in the near-RT TIC and operates within the range between 10-500 ms and above. The third control loop operates in the non-RT RIC and makes decisions at a time greater than 500 ms (e.g., for policy and orchestration purposes). These control loops can also operate in parallel. Offline/online training and inference can be supported via O-RAN components such as SMO, non-RT and near-RT RICs.

In line with recent developments in data engineering, a mapping can be made between the components of O-RAN and the existing data engineering ecosystem. The O1 interface is used for data collectors and preprocessing entities within the service and management orchestrator (e.g., within the Open Network Automation Platform (ONAP) [139]). The O1 interface can be used to transmit RAN metrics associated with the performance of the RAN nodes to the SMO. In addition, non-RT RIC placed in the SMO can enable RAN optimizations using the collected RAN metrics and contextual (external) data. The SMO can later control the RAN and apply configuration changes. Fine-grained data collection is performed via the E2 interface, to enable near-RT control and optimization of RAN elements.

Data collectors and preprocessing entities may use the previously defined data connection frameworks described in Section II. The Virtual Event Streaming (VES) collector in O-RAN[112] is used as a telemetry collection interface and supports data gathering from O-RAN. Subsequently, the collected data can be shared with non-RT RIC using the data ingestion frameworks described in Section III.

AI/ML models trained in AI/ML platforms can later be queried by non-RT RIC via batch processing or Big Data query engines. Regular AI/ML workflow including model training, inference and updates, batch data processing, big data query generation processes and policy-based guidance of applications/features can be performed using the data analysis and processing frameworks defined in Section IV. Note that in the O-RAN architecture AI/ML model training can be hosted in the non-RT RIC, while ML model inference can be located either in the non-RT RIC or in the near-RT RIC when supervised/unsupervised ML/DL approaches are used. If reinforcement algorithms are used, ML training and inference host can be colocated either in the non-RT RIC or in the near-RT RIC. The subjects of the action are near-RT RIC, CU, DU and RU units. The goal of this analysis can be predictions, alarms or suggested actions in the event of unknown network conditions (e.g., SLA violation prediction, predictive maintenance of a RAN instance, energy optimization for coverage maximization.)

When near-RT is subject to an action, the ML inference results or policies/intents can be transferred via the A1 interface to near-RT RIC, where near-RT RIC can apply streaming, real-time, or interactive analytics to the dataset. Appropriate configurations can then be applied to control or data units via the E2 interface which establishes communication between the lower RAN modules (RU, CU and DU) and the near-RT RIC and is used for time-sensitive control of the RAN components.

[112]https://github.com/o-ran-sc/smo-ves, accessed March 2022.

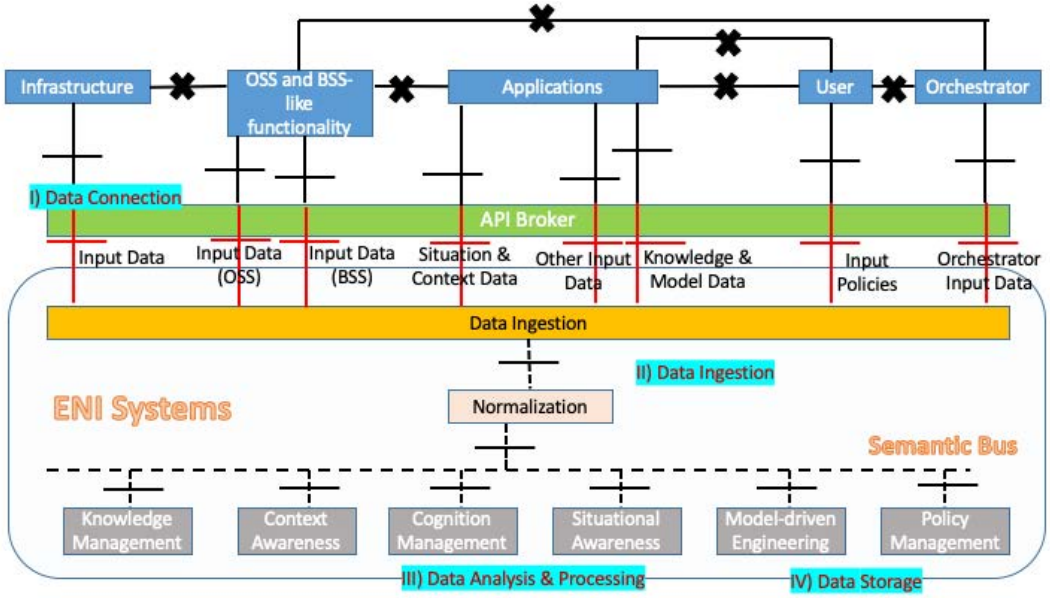

FIGURE 4. Overview of the ETSI ENI reference points and the architecture and mapping to the components of the data engineering pipeline.

For performance monitoring, performance data from the ML models deployed either in near-RT or non-RR RICs can be linked to the data visualization and monitoring frameworks. By monitoring these relevant performance metrics, decisions can be made such as whether or not a model retraining is required. These update decisions can be triggered by either a rule-based policy (e.g., threshold-based) or by using trend analysis approaches.

B. ETSI'S AI/ML ARCHITECTURE

ETSI has several specification groups working on embedding intelligence into network services and management infrastructures. ETSI's Industry Specification Group (ISG) ZSM aims to develop a new horizontal and vertical E2E architectural framework designed for closed-loop 100% automation and optimized for data-driven AI/ML algorithms [132]. An extensive list of requirements for zero-touch management that contains more than 170 topics on autonomic management is already defined by the ETSI ZSM group. [140]. In ETSI's ISG NFV, the AI/ML platform will eventually be considered as part of the MANO stack. The ETSI Technical Committee (TC) Core Network and Interoperability Testing (INT) is investigating the Generic Autonomic Network Architecture (GANA) for the purposes of autonomic networking [141]. Similarly, the ETSI Experiential Networked Intelligence (ENI) group is working on the application of AI in telecommunication networks to help operators manage infrastructure and provide more resilient services offered to end users and has also recent published standards [142]. Experiential learning is learning through experience. The ETSI ENI architecture uses AI techniques and policies driven by context awareness and metadata so that the services provided can be adapted to environmental conditions, user needs and business goals. The main goal is to develop a control loop based on the ''observe-orient-decide-act'' model.

Fig. 4 shows an overview of the ENI architecture and reference points and their corresponding mappings with the presented data engineering components. API broker in the middle is an optional functional block and acts as a gateway (i.e. translator) and maps the data connection framework of the data engineering pipeline. Within the ENI system, there are several function blocks that mainly represent the management and application components that are connected to the semantic bus. These functional blocks can be implemented as part of the data analysis and processing and data storage framework of the data engineering pipeline. For example, situation awareness blocks are used to detect events and behaviors in the ENI system and its environment. In the case of high traffic and large amounts of information, the corresponding streaming application used in the data analysis and processing framework is an important differentiating factor. The policy management functional block allows users to create and edit policies so that consistent and scalable decisions can be made about the system behaviour. The model-driven engineering block uses a set of domain models that abstract all concepts related to managing objects in the ENI system. Therefore, both the policy management functional block and the model-driven engineering block can be mapped to the data storage framework in the data engineering pipeline. A comprehensive overview of ETSI ENI on using AI techniques for network management and orchestration can also be found in the references [143], [144].

C. ITU MACHINE LEARNING PIPELINE

International Telecommunication Union (ITU)'s FG-ML5G group is working on ML pipeline [145]. Fig. 5 shows an

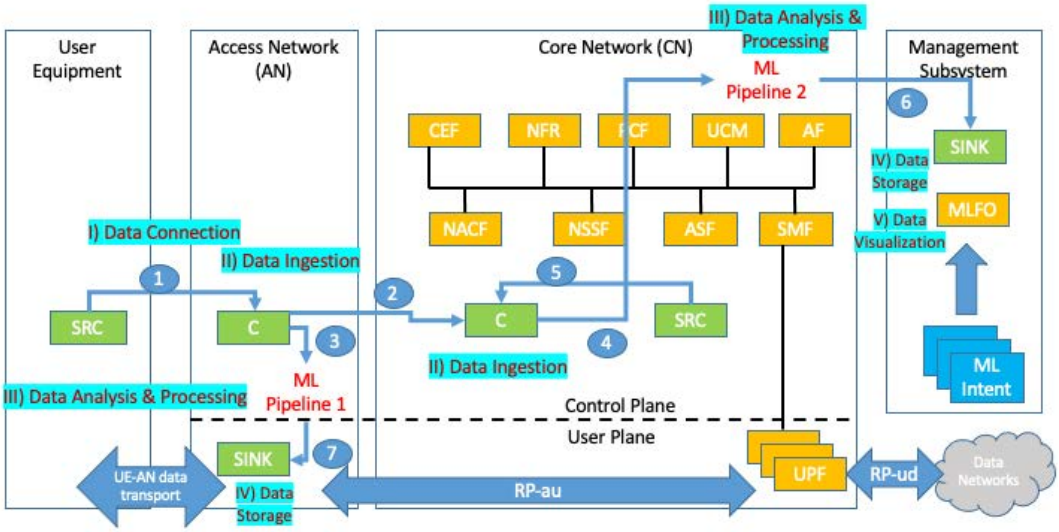

FIGURE 5. High-level architecture in an IMT-2020 network and mapping to data engineering pipeline components.

example realization of the high level architecture in an IMT-2020 network and the corresponding mapping. In this pipeline of Fig. 5, there are several nodes for creating ML pipelines:

- source (represented by SRC) is a node that generates data to be used as input to the ML function.
- collector (represented by C) is a node responsible for collecting data from SRC.
- pre-processor (represented by PP) is a node used for pre-processing the data
- model (represented by M) is a ML model used for prediction (note that training of the model is performed in a sandbox (not shown in Fig. 5)).
- policy (represented by P) represents the control mechanism for improving the operation
- distributor (represented by D) is responsible for distributing the ML results to the corresponding sinks
- sink (represented by C) is the target node of the ML output where the action is performed (for inference purposes).

Note that in Fig. 5, some subsets of nodes (e.g., PP, M, P, D) are inside the ML pipeline and are not shown. In Fig. 5, latency-sensitive applications use ''ML pipeline 1'', while latency-tolerant applications use ''ML pipeline 2''. Inputs from UE are processed by the ML pipeline represented by arrows 1-> 2-> 4-> ''ML pipeline 2'', to make predictions for the Core Network (CN) (e.g., MPP-based ML applications). In the ML pipeline arrows represented by 5-> 4-> ''ML pipeline 2''-> 6, the inputs of CN (as well as some combinations of UE inputs) are combined to make some predictions in the CN so that these actions can be performed by management functions (e.g., actions such as Self Organizing Network (SON)-level decisions at CN or closed-loop decisions about resource allocations in the network). In the ML pipeline arrows represented by 1-> 3-> ''ML pipeline 1''-> 7, the inputs of UE are used for latency sensitive applications in the access network, where model hosting and serving are also performed. When mapped to the corresponding data engineering pipeline frameworks, the connection of SRC to collector (represented by C) is via the data connection module, where collector can be selected from data ingestion framework and ML pipeline 1 & 2 falls into the category of data analysis and processing frameworks.

D. 3GPP NETWORK DATA ANALYTICS FUNCTION

The architecture framework for 5G management and orchestration is specified in 3GPP. TS 29.520 in R-16 is the standardization effort of 3GPP for 5G network automation using ML and data analytics. Within the latest approved 5G specification in 3GPP (Release 15), 3GPP identifies two main building blocks responsible for data analytics [146]:

1) **NWDAF (Network Data Analytics Function)** collects data from core network functions and provides network data analytics services to other Network Functions (NFs) of the 5G Core which are subscribed as NWDAF consumers [147]. The NWDAF offers two services (called Nnwdaf services). The first is called *Nnwdaf _EventsSubscription* service, which allows NF service consumers such as PCF, OAM to subscribe or unsubscribe to various analytics events provided by the NWDAF. The second is called *Nnwdaf _AnalyticsInfo* service, which is used by NF consumers to request and receive specific analytics from the NWDAF. Hence, through a service-oriented interface N_{nwdaf}, other NFs can access analytics information. In 3GPP Rel. 15, Policy Control Function (PCF) and Network Slice Selection Function (NSSF) are envisioned as possible consumers of network data analysis. For instance, PCF

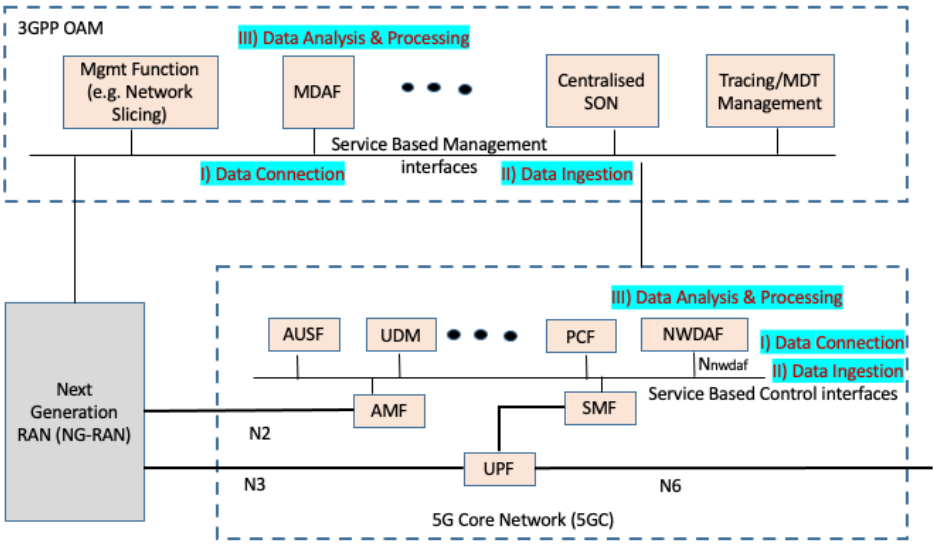

FIGURE 6. Functional framework of the 3GPP 5G system to support management and network data analytics services and mapping to the components of the data engineering pipeline.

may use this data to adjust QoS parameters, or NSSF can use the slice-level load data for slice selection. Some relevant use cases defined in 3GPP TR 23.791 are related to service experience prediction, load analysis, UE behaviour and pattern prediction, etc.

2) **MDAF (Management Data Analytics Function)** is responsible for providing management data analytics services. The analytics results generated with the Operation, Administration and Management (OAM) data can be used by other management functions such as C-SON (Centralized SON) to recommend appropriate actions to network operators.

Finally, Fig. 6 shows the functional framework for the management and network data analytics services in 3GPP 5G systems and the corresponding mapping to the defined components of the data engineering pipeline. Data connection and data ingestion modules are located within Service Based Management and Control Interfaces. Note that both the NWDAF and the MDAF provide data analytics. Data analysis and processing frameworks can be deployed in NWDAF and MDAF, while data ingestion frameworks are deployed as part of the service-based management and control interfaces. Using data visualization and monitoring tools (e.g., Grafana, Kibana), graphical dashboards can be used to provide charts and notifications on the current operational status of each monitored source as well as analytical results.

E. OTHER ACTIVITIES AND SUMMARY

There are also other industry alliances in GSMA,[113] BDVA,[114] and TM-Forum[115] that are also working on specifications of AI in larger domains including telecommunications and their corresponding gap analysis.

In summary, recent advances in the standardization bodies have brought their own ideas and proposals for shaping the possible integration options of the AI/ML platform with the network infrastructure. On the other hand, recent technological advances in data engineering are progressing rapidly and the novelties and new functionalities of each framework in data engineering have not yet been sufficiently explored in the telecommunications standardization bodies. Similarly, a clear separation of data engineering pipelines within the proposed architectures has been neglected.

XI. DATA ENGINEERING USE CASES IN NETWORK MANAGEMENT AND ORCHESTRATION

There are several use cases in the telecommunications industry where the data engineering frameworks described above can be applied. Some of the relevant use cases are also discussed within the standardization bodies as well as the alliance organization in ETSI [1], 3GPP [2], ITU [3], GSM Association (GSMA) [4]. Some descriptions of them are as follows.

- ETSI ENI document [1] has classified use cases in four different dimensions: (i) infrastructure management (energy optimization using AI, handling planned peak events), (ii) network operations (intelligent fronthaul management and orchestration, radio coverage and capacity optimization), (iii) service orchestration and management (closed-loop (autonomic) fault-management, autonomic performance management, context-aware service experience operation, intelligent network slicing management) and (iv) assurance (network fault identification and prevention, assurance of service requirements)

[113] https://www.gsma.com/artificialintelligence/applied-ai-forum/, accessed December-2021
[114] https://www.bdva.eu/sites/default/files/AI-Position-Statement-BDVA-Final-12112018.pdf, accessed December-2021
[115] https://www.tmforum.org/ai-data-analytics/, accessed December-2021

- ITU document [3] has compiled more than 30 use-cases and their requirements. The use cases are divided into five categories: (i) Network slice,service, (ii) User plane, (iii) Applications, (iv) Signaling management, (v) Security. The requirements are divided into three categories: (i) Data collection, (ii) Data storage and processing, (iii) ML applications.
- GSMA report [4] has detailed typical seven different use cases for intelligent autonomous networks in China: (i) AI for network planning and deployment, (ii) AI for network maintenance and monitoring, (iii) AI for network optimization and configuration, (iv) AI for service quality measurement and improvement, (v) AI for network energy saving and efficiency improvement, (vi) AI for network security protection, (vii) AI for operational services.

Data engineering solutions can help provide closed-loop automation, self-organizing, self-healing, self-decision making and self-optimizing network solutions for a variety of problems in network management and orchestration, network planning and design, network construction, network optimization, and network operations. The scope of data engineering solutions can be diverse in wireless, fixed networks, core networks and data centers. For example, considering that about 2000 parameters need to be optimized in 5G networks [148], network automation using the recent advances in data engineering and data science becomes crucial factor to bypass the human-based optimization process. In [11], [149], several novel use cases for (wireless) network design using DL and AI capabilities are presented. Below are some examples where data engineering frameworks can enable or influence their functionalities:

- **Using data connection frameworks**, providing API gateways for network providers,
- **Using data ingestion frameworks**, real-time monitoring applications for hardware (routers, switches, other network devices), software and security (threat discovery and mitigation, DDoS, etc.), data distribution (multimedia distribution (IPTV, content delivery service, etc.), text messaging service, chatbots (for quick access to inquiries and information (e.g., known faults, etc.))), OSS/BSS-related functionalities (providing real-time information (inventory/assets) to supply retail stores as part of supply chain management, billing services, network fault ticket management, network alarm management),
- **Using data processing and analytics frameworks**, leveraging AI/ML algorithms for CDR processing for churn analysis, real-time alarm correlation to identify and predict network faults, root cause analysis for network faults, preventive maintenance, anomaly detection, Customer 360 (tracking and analyzing user behaviour/needs and their interactions with channels), inventory/asset tracking as part of B2B products, recommender systems, network analytics, contact tracing, fraud detection, traffic flow prediction, intelligent network and service slices management and configuration, intelligent initial access and handover at RAN, context-aware service experience optimization, intelligent carrier management SD-WAN, SLA path adaptation for network delays, intelligent software rollouts, energy optimizations with AI, policy-driven IP-managed networks, intelligent fronthaul management and orchestration, service requirement assurance, federated learning for privacy awareness, transmission optimization, opportunistic data transmission in vehicular networks, predictive power management, automated scaling of VNFs, automated deployment of network service slices, automated site design based on coverage and capacity.
- **Using data monitoring and visualization frameworks,** visualization of transport network equipment to enable data-driven infrastructure decisions, visualization of service mesh topology to monitor traffic flow and metrics display.

Finally, note that each of the frameworks or their interconnected versions in the Table 5 can be deployed at different levels of the network depending on the requirements of use cases. These levels can be divided into three levels:

- **In node level operation,** data is collected, processed or analyzed at individual nodes (e.g., at UEs or device level) and no network connection is established. This is useful for data security and privacy, reducing latency and complexity (since the data is processed at the device level). However, since the processing is done at the node level, performance limitations in the analysis results are expected.
- **In network level operation,** data is collected, processed and analyzed within a single domain of the network (e.g., at RAN or in the core network). This increases data diversity because the data catalogue contains data from multiple nodes in that domain. (e.g., from AMF, User Plane Function (UPF), etc. in the 5G core domain) leading to better performance optimizations. On the other hand, data security/privacy and delays are some of the drawbacks of network level processing.
- On the other hand **in global level operation,** data collection, processing and analysis are done with complete knowledge of the data sources in the network in different domains (e.g., E2E network service management). One main benefit of this approach is high performance. On the other hand, there are some issues related to global data integration and deployment cost in this way of operation.

XII. GAP ANALYSIS, CHALLENGES AND FUTURE DIRECTIONS

Most telecommunication companies today require more comprehensive solutions that address both the complexity of their infrastructure and the intense needs of their users. Since data engineering technologies are younger compared to traditional telecommunication services and products, a few telecommunication infrastructure providers are aware of the capabilities

of data engineering solutions. However, there is a growing number of use cases and a growing community and interest in early and rapid adoption of data engineering tools and technologies in the telecommunications world. Some early case studies have highlighted some of the existing gaps and challenges in the adoption of Big Data analytics in the telecommunication industry [150]–[155]. As telecommunication providers try maintain their status quo, they are at risk of being left behind with their products and services in a rapidly changing ecosystem. In this section, we explore the potential gaps, challenges, and future directions in the adoption of data engineering approaches in the telecommunication industry.

A. GAP ANALYSIS

Our survey results show that there are several gaps between developments in the world of data engineering and telecommunications. These can be summarized as follows:

(i) Data engineering framework deployment risks: It is critical for telecommunication infrastructure and service providers to understand the advantages and disadvantages of the technologies currently in use and the emerging cutting-edge technologies in the data engineering world. There are stringent requirements for operational efficiency, availability, reliability, robustness, and stability of telecommunication networks when systems are deployed in production environments. For this reason, the risks associated with the potential deployment of these emerging technologies within a mature and traditional telecommunications infrastructure must be fully assessed. For example, in the case of deploying ML models, the inherent randomness of ML systems may make it difficult to achieve reproducible results or workflows across different experiments [156], which may affect the reliability of the overall deployment process of the data engineering pipeline.

There are several industry players such as Nokia, HPE, Juniper, etc. that already offer network solution products (mostly in the wireless area) that leverage BDA as listed in Table 3 of [32]. However, most of the newer products are not very mature and not fully tested in production environments. Therefore, most of the new AI/ML-based technologies, e.g., advances in DL based neural network architectures, have not yet been tested in real-world applications for telecommunication applications. (This is due to their low Technology Readiness Levelss (TRLs)), which makes it difficult to assess their potential adoption. One way to monitor failures or potentially problematic scenarios in data engineering/science projects and their use in production is to monitor their adoption in other industries so that they can be intelligently adopted in the telecommunication domain. For example, some of the challenges described in [156] in developing pipelines for data management, model learning, verification and deployment in various domains such as computer vision, human-in-the-loop neuropathology, etc. may also be useful for telecommunication operators. TRL assessments of some of the most important and representative AI technologies, as listed in [157], can help telecommunication operators better understand the state of the art of a particular technology when planning to adopt it in telecommunication infrastructure. The integration and scaling issues raised in [158] when deploying AI/ML in a cross-organizational context (e.g., between a hospital and a service provider) may be useful for telecommunication operators when integrating AI/ML platforms into their vertical industries [159], [160].

For this reason, it will be valuable for telecommunication providers to have first-hand knowledge of the shortcomings of these platforms, recognize the weaknesses of these systems, adopt the good parts of the most needed frameworks, and learn from past experiences. This will allow them to adapt to the changing landscape of data technology with less operational and technical complexity.

(ii) Operational costs: Introducing new features and capabilities related to emerging use cases within the telecommunication infrastructure is attractive but at the same time can be costly due to operational expenditures. For example, IT and cloud giant Google has shown that deploying ML-enabled systems real world incurs huge ongoing maintenance costs due to a variety of tasks such as configurations, data collection, feature extraction, data verification, analytics tools, machine resource management, process management tools, serving infrastructure and monitoring in addition to creation of ML code [161]. Moreover, operations units and network engineers working on the day-to-day operation of telecommunication networks should have additional skills such as data modeling, software engineering and system design, ML libraries, etc. On the other hand, data engineers, data scientists, and ML engineers working in the telecommunication world should acquire domain expertise in the functioning of the legacy systems so that accurate modeling of these systems using the data landscape is possible.

(iii) Support for services: Traditional telecommunication infrastructures and vendor-based solutions are mature technologies with advanced enterprise support in case of service outages or activation of new features. On the other hand, many enterprises also rely on ''microservices'' because they are highly flexible and can be easily developed to meet dynamic business needs. Microservices based design requires constant communication between the various components of these services to keep them in sync with each other.

On the other hand, the data engineering ecosystem is still young and rapidly evolving. Hence, some open source technologies developed within the data engineering ecosystem may have a larger community and advanced ecosystem compared to vendor-based data analytics solutions. For example, Big Data technology providers such as Cloudera/Hortonworks or MapR offer support services and subscription service models for their open source toolboxes. However, deploying these open source data engineering technologies as a telecommunication service may require extensive support from internal teams such as OSS or CRM departments of telecommunication providers. The authors in [162] have shown that providing reliable data science

services (e.g., when simply combining open data from different open APIs) can pose several software engineering challenges to enterprises.

(iv) Performance guarantees: Mobile and fixed networks have different SLA guarantees, as mobile networks consist of RAN, transport and core networks. For example, RAN is prone to interference and complex propagation environments that can lead to unpredictable outcomes. Therefore, in different network scenarios and conditions, algorithms within different layers in the protocol state (e.g., in the RAN domain, L3 algorithms (load balancing, mobility and session management, etc.) and L1/L2 algorithms (power control, link adaptation, scheduling, etc.)) aim to improve telecommunication-specific KPIs collected from various data sources (flows, logs, streams, databases, etc.) and bring network conditions to a steady state.

At the same time, in the world of data engineering, other KPIs such as scalability, latency (which measures how close the system is to delivering streaming and messaging in real time, for example *P95 latency of less than 5 ms* means that 95% of all data processing requests should complete in less than 5 ms), input rate (how much data flows from a system like Kafka or Pulsar in one second), processing rate (which indicates how fast data analysis can be performed), etc. For example, if the input rate is greater than the processing rate, the system lags behind, so scaling within the cluster must be done to handle the greater data load. For this reason, data engineering and telecommunication-specific KPIs are interrelated. However, there is currently no standard way to combine network-specific KPIs with data engineering KPIs. The specification of the common KPIs and the resulting necessary SLAs remains an open research area, depending on the different data-related use cases in such an integrated production system.

(v) Customized functionalities: Some of the custom demands from the telecommunication world would be difficult to implement in the data engineering ecosystem, as the convergence of these requirements may be different. Therefore, some use case may require more effort and investment. Some examples that require complex functionalities in telecommunication world are response time less than 1 ms and reliability of above 99.99999% in connected autonomous solutions (e.g., drone delivery systems, drone swarms, etc.), guaranteed microsecond delay jitter in industrial automation and robotics solutions [163], [164], data rates up to 100 Gbps for highly mobile hotspots [165].

(vi) E2E ML lifecycle management: Due to exponential use of AI/ML technologies in software and hardware systems, the development and deployment of ML systems currently tends to be rushed, isolated from real-world environments, and without the context of larger systems or broader products into which they are to be integrated for deployment [156]. For this reason, current ML project lifecycle processes and guidelines do not follow clearly defined processes and testing standards that facilitate the development of high quality and reliable results. This is also true for development in the telecommunications sector, which relies on AI/ML technologies.

In a typical telecommunications system, typical concerns such as data, experimentation or model management, deployment, reproducibility, and testing & monitoring should be considered depending on the ML platform. In a ML project lifecycle management as described in [166], all business requirements and goals of the project must be defined first before the project starts. After the business requirements are co-decided and the project objectives are defined, data collection and preparation phase follows. Then come the feature engineering and model training stages (in the case of DL, these are grouped under the term model training) and model evaluation are performed during the AI/ML training process. After the best model is selected, the model deployment, model serving, model monitoring, and model serving stages need to be executed sequentially. Depending on which execution stage, different feedback must also be provided to the previous stages. For example, in training phase, model evaluation stage provides feedback to the model training, data collection & processing, and even to goal definition stages. Similarly, model maintenance can provide feedback to the model training and data collection & processing stages. A lean Machine Learning Technology Readiness Levels (MLTRL) framework for developing and deploying robust, reliable, and responsible ML systems, as proposed in [156], can also be used in a telecommunications engineering project.

(vii) Data collection and preprocessing: The data collection/gathering must be continuous to keep resulting ML model up-to-date and compatible with practical infrastructure and systems [167]. Most ML models must function in dynamic data environments in production. For this reason, ''concept drifts'' (i.e. the degradation of model performance due to less similar data in production on which the model was trained) are likely and can affect the accuracy and reliability of the model over time. Therefore, building robust ML models in a changing mobile environment is different and requires active (continuous) learning. For this reason, continuous training has been proposed as part of the MLOps practice to re-train the production model [167], [168] frequently as new data becomes available or model performance degrades.

Uniform and homogeneous data collection from all components of the network (NFV, IoT, 5G, etc.) and the data discovery process remain a significant gap between practice and ongoing efforts in both standardization and framework development. Not all vendor-provided functionality can be standards-compliant or has clear interfaces for acquiring data for analytics services. At the same time, obtaining real data can be time consuming and complex, especially when it comes to obtaining the accurate data set from a variety of sources (data streams, logs, databases, etc.) and extracting useful information from it. Data may be dirty, not easily accessible (e.g., proprietary) or not available at certain times within production systems. In addition, the frequency of

sampling the system and temporally stationary conditions of distribution over the sampled data are also critical to data collection and must be investigated depending on the application.

In parallel with data collection, data stored in Data Lakes needs to be cleaned and categorized as it may be incomplete, not correctly normalized or labelled, and noisy. These data should also be prepared for further analysis in the data engineering pipeline applications (e.g., in data processing and analysis frameworks for AI/ML algorithms). Some of the preprocessing activities are handling missing data (imputation of missing values for numeric and categorical data), data scaling, e.g. min-max scaler, standard scaler, max abs scaler, robust scaler, power transformer, quantile transformer, normalizer, etc.,[116] outlier data, transforming data types, dimensionality reduction, identifying numerical and categorical features, encoding categorical features, feature engineering/selection, sampling tasks, to name a few are other missing dimensions in the current data engineering architectures for network management and orchestration. These aspects are critical as missing values or incorrectly populated datasets can lead to inconsistent analysis results.

As a general rule, less than 1% missing data is trivial to handle, 1–5% missing data can be manageable, 5–15% missing data requires sophisticated methods to handle, and more than 15% can seriously affect any kind of interpretation [169]. Some of the most popular solutions are either removing the data (which leads to information loss and biased assessments) or using some advanced imputation techniques and maximum likelihood methods [170]. Multiple imputation using chained equations (MICE) [171] and factor analysis of mixed data (FAMD) [172] are some of the commonly used imputation techniques for hybrid missing data sets (e.g., in both categorical and continuous data). Recent developments and solutions using DL such as DataWig (which trains a neural network based classifier to predict the missing values) can also be used for scenarios with many missing observations [173]. As the authors note in [174], while there are many approaches that deal with missing values, they are mostly designed for matrices only. However, in many real-world applications, the data is not only available in numeric format, but may also be in textual form or as an image. Another main problem with the above traditional approaches is that rare values that are common in heavy tailed real-world datasets cannot be accurately identified with the trained models [175].

In particular, mobile data collected from network devices is often subject to redundancy, loss, mislabeling or class imbalance, and thus needs to be preprocessed before it can be used directly for training. Compared to traditional ML, DL methods that process missing values in batch mode, telecommunication systems are more dynamic and more missing data is received in less time per second due to the nature of networks and wireless communication infrastructure. Therefore, special care must be taken when processing dynamic and fast telecommunication traffic.

(viii) Model training with real data: The goal of model training is to optimize and rapidly converge model parameters to optimal and consistent values given a set of training data. However, developing a model to meet specific product requirements may require combinatorial search for parameters, variables, etc., which can become difficult as the model becomes more complex. Training models for production systems can be costly due to a lack of either data or properly labeled data (especially for supervised trained models that require data augmentation, e.g., labeling large amounts of data, experts experience). In such scenarios, simulated data can be used to train the models, but the gap between practice and theory, i.e. the lack of adequate knowledge transfer from simulation to real conditions, can be a major problem.

Training requires observing a wide range of scenarios and in the case of network management and operations, generating different network configurations and scenarios that can potentially disrupt network operations for training purposes. For example, in reinforcement learning applications, the agent must interact with the environment as it tries different actions and receives feedback to improve the outcome based on its actions. During this process, the agent makes several mistakes while learning which requires a large number of steps to converge to an optimal or near-optimal solution. For this reason, the training of the agent is not performed in the real infrastructure, since errors can have serious consequences for the networks (failures, false alarms, downtime, etc.). Instead, a simulator can be built that mimics the real network environment, and the agent is trained offline in this simulator environment. However, the simulator must meet high fidelity requirements because the real network may be different from the network used for training. Moreover, the agent cannot operate appropriately if the discrepancy between the real and simulated environments is large [176].

In the case of ever-changing mobile network environments, models of ML should have the ability to learn continuously (active learning discussed in Section XI. B bullet no. (xviii)) or perform transfer learning [177]. Transfer learning can accelerate the training process when the conditions in the mobile network change significantly. Basically, transfer learning aims to reduce the amount of training data required to learn a task, by reusing the feature extraction layers learned on other datasets [177]. In other words, it enables the rapid transfer of knowledge from pre-trained models to other types of datasets. Transfer learning can be used to improve the performance of models that learn with limited data. For example, if software viruses are spreading rapidly in the network, the anomaly detection model or antivirus software detection model built into the network equipment should be able to respond to these attacks in a timely manner with the limited information available.

In transfer learning, a model learnt in a particular environment (e.g. cellular Base Stations (BSs) operating at low-bands, $<$ 1 GHz) can be transferred and adapted to

[116] https://scikit-learn.org/stable/modules/classes.html#module-sklearn.preprocessing, accessed: July 2021

another network node operating in a different environment (e.g. cellular BSs operating at mid-bands (1-6 GHz) or high-bands (> 20 GHz)). This is technically referred to as frequency-based transfer learning in [178]. When the system dynamics in the environment change, e.g., due to a device malfunction, a different, previously unknown terrain, different frequencies, etc., the result is a completely different data set than the one previously used for training. Therefore, the previously trained model must be trained again in this new scenario/at this new frequency, which is inefficient because all these datasets must be acquired again.

Transfer learning approaches applied in wireless communication aim to tune the existing models with a small amount of data in the changed environment [179]. For example, frequency-based transfer learning transfers the models trained on different frequencies to the target frequency, while scene-based transfer learning transfers the models trained on different scenes to target scenes that use the same frequency [178]. The authors in [178] also showed that both frequency-based and scene-based transfer learning models can predict path loss with small errors by using limited data of the new environment and learning the regularities between path loss and scenario information in detail.

(ix) Model deployments in real-world environments: Comprehensive monitoring of system behavior and taking automatic actions (without direct human intervention) are crucial for higher system reliability in the long run. On the other hand, deploying models in production is still not an easy task. There are pre-deployment, deployment, and non-technical challenges during the deployment and during the operation of ML models in practice [180]. According to the authors in [181], the main challenges are mainly related to model integration (operational support, code and model reuse, software engineering anti-patterns and mixed team dynamics), model monitoring (feedback loops, outlier detection, custom design tools) and model updating (concept drift and continuous delivery). Note that in a production environment dozens or hundreds of models may be running simultaneously. Therefore, the developed ML models should be monitored, alerted and automatically recovered in case of failures to achieve a certain service level goal.

Depending on the area where improvements are needed in terms of intelligence, use case requirements and static/dynamic characteristics of the environment, the update frequency of the ML model may also vary. For example, choosing an optimal threshold in auto-encoder-based neural networks (e.g., in anomaly detection applications) is important to achieve a good trade-off in certain metrics such as precision and recall [182]. Thus, if a new training/validation dataset is created frequently, the optimal threshold and the corresponding model must also be updated frequently to cope with changes in the state of outside world. In the case of dynamic network environments (e.g., RAN algorithms using L1 to L3 transmission parameters, modulation and coding schemes, resource allocations, etc.), the model and corresponding hyperparameters should also be updated frequently (fast time scales on the order of seconds/milliseconds), as the ML models can degrade or exhibit biases and user behaviour may change over time [183]. In network optimization, the model update frequency can be on the order of hours/days/weekly (e.g. hyper-parameters for SONs algorithms). However, for network design, this update frequency can be on the order of weeks/months (e.g., when deploying new cell in a given geographic region) (see Figure 1 of [183] for more information on the main areas of performance improvement). For this reason, depending on the scenario considered, an appropriate feedback loop for the deployment of the model is also required to achieve good and timely results.

As frequency of ML model usage in a service provider increases, many models need to be supported either sequentially or concurrently by a model server. Several deployment options are available, such as *A/B testing* (one set of data is used by one model and the remaining is used by another model) [184], *ensembling* (combine multiple models to get a stronger model) [185] or *cascading* [186] (make predictions based on a model (e.g., detect anomalies within the infrastructure or find the root case). Depending on the requirements and the complexity of the deployment pattern, different options for the selection of algorithms, architectures, tools, etc. need to be defined. For example, unsupervised learning algorithms may offer lower latency and cost savings at the expense of performance degradation in data analysis and processing compared to supervised learning algorithms. As an alternative solution, the developed ML models for production systems can also be embedded in the operating system kernel and provided as a system service.

(x) New architecture for event driven applications: Traditional telecommunication systems and their legacy applications are based on an application architecture that are using APIs [187]. A common gap in the traditional telecommunication network architectures is that they were not originally designed for event-driven applications, although recent efforts on Service Based Architecture (SBA) in the 5G core network have shown some tendencies towards their deployment [188]. On the other hand, data-driven architectures for telecommunication systems are based on using huge amounts of data and transferring them to a AI/ML platform for large scale analytics [14]. However, this may disregard some of the already existing application capabilities, such as enterprise integration capability or agility. For this reason, a balance is needed between a data-driven architecture (which specializes in transferring large amounts of data between applications) and an application architecture (which ensure that the functionality of one application is executed in response to a request from another application).

The general approach in the industry is to move to stateful, event-driven and event-time-aware processing using the concepts of events, streams, producers, and consumers. Many industries have already started to move from a monolithic architecture to a microservice architecture for scalability and maintainability reasons [189]. Event-Driven Architecture

(EDA) has already proven itself in the cloud and IT communities and will become the software architecture paradigm of choice in the telecommunication domain in the coming years. Together with the introduction of the concept of SBA in the 5G core [190], it is expected that it will soon be used in the architecture of mobile networks.

An event is a change of state or an update in the system. EDA uses a sequence of events to trigger and control communication between microservices (i.e., decoupled services). It is particularly suitable for applications based on microservices interconnected by fast asynchronous events [191]. In an event-driven system, there are collections of independent services between which there is no direct coupling. The data schema is the only dependency between them. This increases the resilience (since failures in a service do not escalate) and extensibility (easy addition of new independent services to the existing systems, e.g., notification service) of the systems. Therefore, an EDA can successfully provide streaming, Pub/Sub, and Push patterns, while web services with REST/HTTP, API gateways, cronjobs, RabbitMQ, Kafka or data at rest with a Data Lake cannot. As a result, enterprises are starting to adapt to EDAs or event sourcing.

There are several ways in which events captured in real-time can be useful for data analysis (e.g., by correlating events with other introduced features, recent incidents, etc.). For example, in most streaming frameworks (e.g., Spark Streaming), window operations are performed as each message is received by the streaming processing framework (i.e., in *processing time*). However, the exact way to customize window operations is to support more advanced event time windows and perform computations based on *event time*, i.e., when the event was created [53]. Another good example of the application of EDA in telecommunication systems would be Network Management System (NMS), where critical events can be quickly responded in order to mitigate the problems in the network. The general workflow associated with EDA for this example could be as follows: (i) NMS detects an anomaly and publishes an AnomalyDetected event (ii) The Root Cause Service subscribes to the event, processes it and computes the location and root cause of the anomaly (iii) The Root Cause Service then publishes the RootCause event (iv) The Region Support Service subscribes to this event and sends a notification to personnel in that region explaining the root cause of the problem.

(xi) Ecosystem integration: Telecommunications network technologies are becoming more complex with each passing decade than previous generations. In 5G networks, for example, URLLC, Enhanced Mobile Broadband (eMBB) and massive Machine Type Communications (mMTC) type communications require specialized technologies (e.g, Massive MIMO [192], coordination algorithms (Carrier Aggregation (CA), Coordinated Multi-Point (CoMP) transmission/reception [193], Single Frequency Networks [194], Multi-Connectivity), new spectrum (high frequency bands (>20 Ghz) such as from millimeter and terahertz (THz) wavebands to visible light), Device-to-Device [195], dynamic network slicing [196], network virtualization [197], Edge Computing [198], integrated satellite-terrestrial communications [199], [200], intent-based networking [201], etc.) that need to be embedded in telecommunication networks. As services become more complex (e.g., with dozens of microservices interacting with each other), management and orchestration operations also become more complex and costly.

To tackle this complexity in the management and control plane of the telecommunications infrastructure, there are several automation tools that can manage the network service management lifecycle (e.g., Open Source MANO (OSM),[117] Open Network Automation Platform (ONAP),[118] Cloudify,[119] etc.). However, they also require complex MANO procedures. For this reason, specialized skills (e.g., network virtualization, cloud services, etc.) are required to fully exploit their application potential and seize the opportunity to develop new and innovative value-added services. At the same time, these new technologies also bring their own specific challenges and obstacles when it comes to integration with data engineering frameworks. Therefore, the tools and libraries selected from the data engineering ecosystem should be well integrated with the broader telecommunication infrastructure systems based on the use cases and requirements. The support of the data engineering ecosystem or community for high quality tools and adoption of the latest technologies into the telecommunication infrastructure are also crucial in this process.

(xii) Licensing: In parallel with ecosystem integration, the licensing gaps for hybrid deployment types need to be further explored. Many of the open source tools for data engineering are licensed under Apache 2.0, which does not imply vendor-specific licensing. On the other hand, legacy telecommunication infrastructures are based on various vendor-specific equipment. Avoiding vendor dependency helps enterprises to develop their own customized services and explore new opportunities and business goals. The gap between the interplay of open source and vendor-locked systems deployments is an ongoing issue and needs to be further explored.

(xiii) Synchronization aspects: In a traditional telecommunication system, one task may orchestrate multiple calls to internal or external services. Telecommunications systems require strict synchronization between multiple components of the data engineering platform and the telecommunications infrastructure. If synchronous orchestration of services fails, the entire service flow fails. Some of these synchronization requirements also arise during data collection, model and hyperparameter updates when multiple actions need to be performed by the AI/ML platform between the interconnected network domains (e.g., joint actions performed on both the core network and the transport networks). For example, after the data connection and ingestion phases are completed, the

[117] https://osm.etsi.org/, accessed: December-2021
[118] https://www.onap.org/, accessed: December-2021
[119] https://cloudify.co/, accessed: December-2021

extracted and transformed data must be continuously synchronized with the original data sources. However, since data sources can be heterogeneous and change dynamically over time, a data source may be out of sync and out of date at the time of integration. This can lead to discrepancies in data schema and definition and cause problems in synchronizing these heterogeneous data sources. The development of such solutions for telecommunication networks in the field of data engineering is still an open research area.

(xiv) Lack of rigorous methodology in networking: Throughout its evolution, networking has evolved both scientifically and through trial and error and configuration based deployments in real systems. As a result, there are complex interaction patterns among the components of telecommunication networks. For example, in most cases, the network is designed to be distributed and each node (router, switch, gateway, etc.) overlooks only a portion of its environment. This also makes it difficult to apply conventional approaches/algorithms from computer vision or Natural Language Processing (NLP) (which also use standardized datasets such as the MNIST (Modified National Institute of Standards and Technology) database for handwritten digits or the ImageNet database, etc.) for direct comparisons of learning or inference algorithms to the complex networking systems. Therefore, a more rigorous and scientific approach is required when designing AI/ML systems in the area of complex and large-scale telecommunication systems.

(xv) Hybrid approach to data operations: Note that all of the above analytics frameworks including data collection, data analysis, data monitoring, data visualization, etc. can be performed either at the device, network or global level as described in Section XI. However, depending on the use case, a hybrid approach may also be required, comprising a flexible and distributed analytics architecture where some necessary data processing is performed at the device level and/or some partial processing is performed at network the level.

Distributing some of the functionalities of these frameworks across these levels can help improve network performance by reducing bandwidth overhead or network latency (which can be helpful for real-time applications). In [202], the authors have shown the benefits of such a hybrid approach to reduce the cost of data communication while ensuring that the accuracy of decision making for IoT networks does not significantly decrease. A flexible placement strategy of different data analytics modules that can be dynamically selected, combined or switched to achieve the best I/O performance is also explored in [203]. The benefits of data orchestration of use case-based analytics for 5G scenarios are proposed in [204].

In the case of such a hybrid approach, a different data analysis setup can be created for different use cases (e.g., mMTC, eMBB or URLLC in 5G network slices). In a network slicing setup where data flow over the industry outside the industrial site (e.g., a factory) is not desired, the edge computing paradigm can be enabled. In this edge computing setup, for example, AI-based image processing for quality inspections can be performed at the edge instead of in cloud servers to reduce traffic and eliminate critical I/O performance bottlenecks. On the other hand, other network slices can continue to run their analytics modules on central servers. In the IoT data processing scenario, computationally intensive data training and inference generation can be performed at a global level (e.g., in the cloud). At the same time, data generated locally at the device level (e.g., from sensors) can be transformed and aggregated locally to save transmission energy and increase data protection while maintaining global accuracy as much as possible.

Another example of functionality distribution is via federated learning. In federated learning, with limited interaction between nodes in the network, local construction of AI/ML models can be instantiated within a single component/node of the network. At a later stage, these small models at the individual distributed nodes can be sent back to a network level coordinator to build a global model and view of the network domain. Finally, the global models can be sent back to the local devices/nodes to improve performance [205].

(xvi) Practical aspects versus system complexity: Another problem that is usually overlooked in the design of data systems is the increase of system complexity in practical systems, e.g. algorithms that are data hungry (increased amount of data) and require high-computational (excessive use of CPU, RAM or storage capacity in servers). Indeed, deep neural network architectures require complex structures and in many cases provide powerful results (e.g., high classification accuracy) that represent a trade-off between accuracy and computational cost (e.g., computational cost for inference, time for hyperparameter optimization). However, despite their high model performance metrics demonstrated in particular in the fields of NLP and computer vision, they also require a significant amount of computational resources and power (e.g. Convolutional Neural Networks (CNN) which rely on operators such as convolution, rectified linear unit (ReLU), pooling and classification) and larger systems such as multicore CPUs and GPUs for fast and accurate performance computation [206].

For this reason, in some use cases, the deployment of deep neural networks, especially on embedded and mobile devices (e.g., training a complex image classification model using local data on resource-constrained (in terms of energy and capacity) mobile devices) may be either expensive or not possible. Therefore, very deep neural networks may not be suitable for these scenarios, as they would compromise some performance metrics (e.g., accuracy). Instead, lightweight architectures that are less suitable for complex tasks should be chosen. This trade-off should be considered especially in resource-constrained smart environments [207]. As a solution, some advanced techniques and toolboxes can be used to deploy these complex DL models in mobile network applications (e.g., while compensating for small performance degradations) [10]. On the other hand, in some cases, e.g., when exploring tabular datasets, tree ensemble algorithms such as XGBoost can outperform deep neural network models

in terms of accuracy, inference efficiency, and optimization time, as shown in [208], which also needs to be considered before increasing model complexity.

At the same time, note that adding new and more sophisticated data components can also slow down the entire process of data engineering pipeline in practical systems. In addition, new systems or components may poorly represent uncertainty, and may lack transparency and trust. Therefore, it is important to weigh the technical pros and cons of the benefits of purely research-based solutions when designing the entire data engineering pipeline in practical real-world systems.

(xvii) Cloud vs. on-premise infrastructure: When designing a data engineering pipeline, the different deployment options (e.g., cloud (public), on-premise (private), or hybrid cloud) and the corresponding trade-offs should be thoroughly analyzed. First of all, there are several advantages to using cloud services. For example, cloud services offer high availability, easy scalability, resilience, cost reductions, and easy accessibility when a product reaches a higher level. On the other hand, building an on-premise infrastructure can ensure that privacy, security and regulatory compliance for mission-critical services.

From a cost perspective, iteratively processing data in the data engineering pipeline (e.g., ML-based data analytics and processing frameworks) and running applications 24/7 in the cloud can be expensive compared to on-premise solutions. For this reason, in some scenarios, enterprises may be interested in taking advantage of both private and public clouds. Hybrid options can leverage the different features and characteristics of multiple platforms as well as traditional on-premise resources. For example, if the data load in one of the frameworks in the data engineering pipeline explodes, additional public resources in the cloud can be helpful until the data load levels drop back below a certain threshold. Hybrid options can also be beneficial for high availability and disaster recovery scenarios [209]. Day-to-day production systems can be maintained on-promise while a backup or recovery environment can be moved to the cloud to provide agility in a disaster recovery scenario.

xviii) Computing resources for training in wireless networks: Wireless networks also have their own challenges, such as uncertainties in the environment (e.g., dynamic channel, security, congestion, interference, connectivity, network expansion, etc.), limited resources (e.g., transmit power, spectrum) or hardware constraints (e.g., computational power) that make training models difficult [35]. Mobile data is dynamic, distributed over a large geographic area, exhibits changing patterns over time, and has inherent characteristics associated with human mobility, location topology, local culture (e.g. events, festivals), etc. For example, the spatio-temporal behaviour of residents may differ significantly depending on the time of day or week [210]. Some of the devices (e.g., mobile devices) also have limited hardware capacities and cannot train complex ML/DL models with large datasets.

In complex and large architectures and environments such as 5G, powerful hardware and software are required to support both training and inference (as data volume and quality become increasingly important) if intelligence is to be built on top of the network infrastructure, as described in survey paper [10] and the articles referenced therein. Therefore, computational and time resources for training processes need to be considered when learning with large datasets especially in wireless applications where patterns change over time [210], [211]. When model training is performed with large distributed datasets on central servers, additional communication and storage costs are incurred and the solution does not scale. An elegant solution is to perform model execution on distributed nodes while ensuring good performance on local data and reducing the load on central servers (e.g., federated learning on wireless networks [212]). To stabilize the training process and accelerate convergence, the optimization process can also be updated as conditions change [213].

Finally Table 6 provides a summary of the gap analysis described above.

B. CHALLENGES

In order to reap the benefits of integrating data engineering ecosystem solutions at different layers of the telecommunication network infrastructure for both telecommunication providers and users, there are also some challenges that need to be overcome. Some of the challenges in putting together a data pipeline architecture are related to the following issues:

(i) Inter-working between different programming languages, tools, computation runtimes: Developers and data engineers use a variety of tools and programming languages (Python, Java, Scala, R, Julia, SAS, etc.). Using multiple languages often increases the cost of effective testing and leads to difficulties in transferring responsibility to others. For example, some message queuing systems such as RabbitMQ [49] or Kafka [45] can be implemented in Java, some other data modules such as Apache Spark are written and work best in Scala programming language (there is also support for Java and Python), most of ML algorithms are better supported by the Python programming language and libraries, and user web applications can be written in the C# programming language.

For this reason, the field of data tools and systems is inherently heterogeneous, diverse and fragmented, as multiple workflows are involved in the process of creating data engineering pipelines. Therefore, supporting multiple languages and decoupling the components of the data engineering pipeline can be critical to reducing the overall complexity of the system and accommodating heterogeneity. Furthermore, it is desirable that entire layouts of existing frameworks in a data engineering pipeline are language-independent and provide software abstractions. In summary, integration with other systems is an ever-growing area and requires overarching tools when data connections between different frameworks are required.

(ii) Choosing the right toolset: Big Data can be categorized under ''7 Vs'': *volume, velocity, variety, variability,*

TABLE 6. Summary of the gap analysis.

Gaps (I)	Description (I)
(i) Data engineering framework deployment risks	Understand the advantages and disadvantages of the technologies currently in use and the emerging cutting-edge technologies in the data engineering world.
(ii) Operational costs	Introducing new features and capabilities related to emerging use cases within the telecommunication infrastructure is attractive but at the same time can be costly due to operational expenditures
(iii) Support for services	Deploying open source data engineering technologies as a telecommunication service may require extensive support from internal teams such as OSS or CRM departments of telecommunication providers
(iv) Performance guarantees	There is currently no standard way to combine network -specific KPIs with data engineering KPIs
(v) Customized functionalities	Some of the custom demands from the telecommunication world would be difficult to implement in the data engineering ecosystem, as the convergence of these requirements may be different

Gaps (II)	Description (II)
(vi) End-to-end ML lifecycle management	Current ML project lifecycle processes and guidelines do not follow clearly defined processes and testing standards that facilitate the development of high quality and reliable results.
(vii) Data collection and preprocessing	Collecting data and building robust ML models in a changing mobile environment is different and requires active (continuous) learning. Special care must be taken when processing dynamic and fast telecom traffic
(viii) Model training with real data	Developing and training a model to meet specific product requirements may become difficult and costly as the model becomes more complex and due to lack of either data or properly labeled data
(ix) Model deployments in real-world environments	Depending on the requirements and the complexity of the ML deployment pattern, different options for the selection of algorithms, architectures, tools, etc. need to be defined
(x) New architecture for event driven applications	A balance between a data-driven architecture (that transfers large amounts of data between applications) and an application architecture (that ensures the functionality of one application is executed in response to a request from another application)

Gaps (III)	Description (III)
(xi) Ecosystem integration	The tools and libraries selected from the data engineering ecosystem should be well integrated with the broader telecommunication infrastructure based on the use cases and requirements.
(xii) Licensing	The gap between the interplay of open source and vendor-locked systems deployments is an ongoing issue and needs to be further explored
(xiii) Synchronization aspects	The development of synchronization solutions for telecommunication networks in the field of data engineering is still an open research area
(xiv) Lack of rigorous methodology in networking	A more rigorous and scientific approach is required when designing AI/ML systems in the area of complex and large-scale telecommunication system

Gaps (IV)	Description (IV)
(xv) Hybrid approach to data operations	Depending on the use case, a hybrid approach is required, comprising a flexible and distributed analytics architecture where some necessary data processing is performed at the device level and/or some partial processing is performed at network the level.
(xvi) Practical aspects versus system complexity	Consideration of a trade-off between accuracy and computational cost (e.g., computational cost for inference, time for hyperparameter optimization) when deploying data engineering solutions for telecom specific use cases.
(xvii) Cloud vs. on-premise infrastructure	The different deployment options (e.g., cloud (public), on-premise (private), or hybrid cloud) and the corresponding trade-offs should be thoroughly analyzed when deploying data engineering pipelines.
(xviii) Computing resources for training in wireless network	Computational and time resources for training processes need to be considered when learning with large datasets especially in wireless applications where patterns change over time

veracity, visualization and *value* [214]. So, depending on rhe use case and the different industry requirements, either one or more of these Vs may be important. For URLLC applications (e.g., telemedicine and autonomous driving) velocity and veracity, for eMBB applications (e.g., remote metering), volume, or for mMTC applications, veracity of

data may be important parameters to optimize when selecting the appropriate data engineering tools from a variety of design solutions. At the same time, to manage the complex workflows and the needs of different stakeholders demanding various network services, a comprehensive list of tools, platforms and frameworks should be used based on the different characteristics of the data sources and the requirements of the data processing.

For example, in data ingestion and transformation, Apache Storm [54] can be used for high volume real-time data, Apache Nifi can be used for medium volume real-time data, and Sqoop can be used for batch data with low latency requirements. In addition, extensive comparisons of some of the latest message queueing systems (e.g., Kafka, RabbitMQ, RocketMQ, ActiveMQ, and Pulsar) have shown that Kafka can be used for higher throughput, RabbitMQ is more suitable for lower latency, while RocketMQ can provide both low latency and high quality of service for applications and services [66]. Some tools, such as Apache Druid, only allow querying a single data set, so joining with multiple other data sources is not possible. Since in such scenarios it is not an optimal combine all data sources into a single data source, e.g., due to the nature of the different services producing data, other custom tools such as Presto can be used for these purposes.

As another example, when developing a streaming application, there is an inherent trade-off between data quality and data speed. To provide a fault-tolerant and scalable system with an *exactly-once-guarantees*, various platforms such as Spark's Structured Streaming and Delta Lake can be used. For out-of-order data processing, Flink's data stream processing is an ideal candidate. Some OLAP solutions designed for Big Data such as ClickHouse itself, are only designed for fast queries over large data set and do not support real-time record-by-record ingestion. Only after integration with a streaming platform such as Kafka is real-time data streaming possible, allowing ClickHouse to act as a message consumer. Therefore, depending on the required reliability of the request (either streaming or batch) and possible trade-offs in performance, data engineers need to choose different tools.

Given all these different options, it can be difficult to find a suitable set of tools for building a data pipeline. The choice depends on numerous factors, such as the analysis results of the pros and cons of the tools or the understanding of their suitability for the use cases under consideration. Ideally, the selected tools should not be tied to a specific vendor, should be supported by a large community, should have clear documentation, should be easy to integrate with the rest of the platform, and should be independent of various software, including cloud services and third-party vendors.

(iii) Support for containerization: There is a growing need for support for containerization to build flexible, service-oriented and cloud-native applications [215]. The general trend is to build services using infrastructures such as Kubernetes clusters (to enable production-grade container orchestration). A production system would run multiple machines, each with hundreds of containers that can be restarted, rescheduled or terminated at any time. As an example of a scale-out architecture, one container-based microservice can be exposed with REST-APIs over Hypertext Transfer Protocol (HTTP), another container can be accessed using Protobuf and gRPC, or another with real-time streaming requirements can expose its microservice via websocket APIs. Therefore, using frameworks such as Kubernetes to deploy containers/microservices provides flexibility in deployment, ease of automation, movement, and scaling.

On the other hand, although modern open source projects such as Pulsar, Spark or Flink provide native support for Kubernetes, there are still many components in the Hadoop ecosystem that have not moved away from YARN or do not provide standard support (e.g. Kafka). For example, automatically resizing jobs in a container (scaling up/down, scaling out/in) for stream processing jobs depending on lags or other performance parameters is also currently a challenging problem.

(iv) Lack of a unified framework for data processing and analysis: In a general data engineering pipeline, online and offline data processing are handled in separate pipelines, each using different computing engines such as Kafka, Spark Streaming, Flink, Hive, Map-Reduce, etc. However, this can add maintenance overhead for enterprise development teams. Inside telecommunication operator, there are a variety of data analytics nodes and tools deployed in various sub-units to perform customer experience management, service quality of service management, revenue assurance, or user/marketing analytics. On the other hand, it is a difficult task to integrate all these separate analytics nodes with traditional systems (e.g., with data visualization/notification applications for reporting, with network management and orchestration tools for service automation) in a single framework.

In data engineering, some frameworks such as Spark or Flink can provide a single, unified data engineering pipeline solution for both online real-time and offline data. However, to generalize application development, some other computational patterns such as distributed training, model serving, streaming, distributed data processing, distributed reinforcement learning, etc. need to be implemented as libraries in addition to these frameworks. Although there are frameworks that provide a unique set of abstractions and a unified APIs for both batch and stream processing jobs, consolidating an advanced data engineering pipeline cannot be achieved with a single unified framework to perform general distributed computation, online multi-stream processing, window operations, stateful analysis or DL simultaneously.

For example, Kafka's data ingestion benefits may outperform Spark Streaming data ingestion framework, while additional data processing such as multi-stream joins or generating additional features for online and offline data can be more effortlessly performed only with Spark and not with Kafka. Similarly, adding support for some libraries (e.g., a current DL framework) may be excluded from the mainstream development process due to lack of resources,

suitable use cases or interest in the community (e.g., because industry needs are not yet mature enough) and because it is very time-consuming to append a new framework to the overall AI/ML stack.

(v) Use of multiple AI/ML frameworks: In many organizations, it is common to use multiple systems and frameworks for different workloads. For example, in data storage, a data lake, many data warehouses, custom specialty databases for graphs, streaming, time-series databases, etc. are common practice. At the same time, some of the emerging areas like DL are advancing very quickly and depending on the task at hand, different DL frameworks can be more effective. While it is easy to experiment with a new framework, it is very expensive to add production support for each new DL library. In cross-domain applications where many of these different computational frameworks or patterns need to be combined, serious challenges can arise. For example, in cases where reinforcement learning or some online learning applications require processing data streams, training and deploying models which may exceed the limits of specific purpose integrated systems. This, of course, increases complexity.

In practice, one way to overcome this problem is to find a way to connect the different frameworks together to create applications that are independent of any particular framework. Another way is to build a new system from scratch that can supports the functionalities of these frameworks with simple APIs for new algorithms, creating general purpose systems. However, these two ways have their own pros and cons. For example, when merging different systems, it is not efficient to move data between frameworks, which can lead to additional overhead and inflexibility (e.g., inferences drawn by the system cannot be updated frequently due to model update difficulties). In addition, the learning curve of all these different frameworks can be steep.

On the other hand, designing and developing a new system from scratch and moving to a new general-purpose system can require a great deal of engineering effort for new application development processes. Despite these challenges, many organizations of today are moving to develop their own internal data platforms consisting of a variety of open-source tools and frameworks, rather than relying on closed proprietary systems.

(vi) Data security and privacy: Legal compliance, encryption, key management and data governance & integrity are the main pillars of data security and privacy. If not managed properly, a large data set distributed across an enterprise can cause major headaches for data owners in terms of security, authentication, authorization and information integrity. With strict regulatory requirements (e.g., the GDPR (General Data Protection Regulation) in Europe) preventing data from being moved to the cloud, many organizations are looking for and investing in tools that can allow only authorized individuals to manage sensitive data on-premises. At the same time, data sources cannot always be trusted, which can lead to gaps in the system. For this reason, ensuring data security is also crucial in model training and validation. The accuracy and integrity of the data set must be ensured by avoiding data collection from faulty or compromised network nodes/users to protect against unfavorable data sets.

All data breaches must be detected as soon as possible. For this reason, data stream processing is ideal for developing security applications that allow to respond immediately. For example, in a typical enterprise, bots, scraper detection, or access monitoring are importance requirements that can be met with available stream processing platforms that provide state management and checkpointing capabilities. Real-time data should also comply with privacy regulations, similar to data stored in data warehouses, data lakes, or traditional data stores. Some companies such as Confluent are already offering new connectors, such as the Privitar Kafka connector which improves the value of streaming data assets without compromising user privacy.[120] Data stream processors such as Splunk DSP can also mask sensitive data.

(vii) Event streaming support: Traditional event streaming in OSS uses various protocols such as Simple Network Management Protocol (SNMP) for routers and service gateways, gRPC and protobuf (Google binary protocol buffer) for telemetry, or syslog events for soft switches for monitoring purposes. However, these various vendor-specific protocols also bring some challenges. Some of them are: Complexity in real-time analysis, multiple data semantics & naming across different device types and data sources. In BSS systems, there are also various challenges related to different systems for broadband, mobile and fixed services, technology stacks (fiber, copper, 4G/5G, etc.), and other Value Added Services (VASs). Several BSS system components need to be extended to include services such as recommendations, augmented reality, payment integration, etc. and integration with legacy middleware components of CRM systems (ETL, Enterprise Service Bus (ESB)) also needs to be done.

Together with data ingestion frameworks, these disparate data streams can be normalized to a common schema facilitating real-time analysis and display of the global network infrastructure. Moreover, data ingestion frameworks can be used to achieve asynchronous communication between components and interoperability between different service providers of OSS and BSS systems. On the other hand, streaming support for model serving purposes is an important feature when selecting data processing frameworks. Although most model serving applications are based on REST, it is not desirable to use REST inside streaming applications and make many calls outside the execution environment. For this reason, new libraries (such as Flink Tensorflow [216]) are gradually emerging that can support streaming model serving.

(viii) Architecture decisions: Various data architecture decisions for streaming and batch processing involve tradeoffs, and organizations are willing to choose the one that offers more flexible scaling, lower operational overhead with

[120]https://www.confluent.io/hub/privitar/privitar-kafka-connector, accessed December-2021

high availability and reliable performance. Some important considerations when choosing a data architecture are scalability, operability (which is more difficult with stream processing jobs due to potential lags), bridging both offline and online scenarios (especially useful for applications using active learning) and ease of data access and movement capabilities (due to the inherent semantics differences in the data). Different data architectures are available depending on the use case and SLAs. Table 7 summarizes the descriptions and drawbacks of these different available architectures.

(ix) Operational complexity: Selecting data engineering systems that can work in a single system reduces operational complexity. However, there will not be a single platform, system or compute runtime that can handle all the entire underlying heterogeneous data engineering infrastructure. This is because there are different data types (big data or small data, graph or log data, etc.) and access patterns (streaming or parallel) in the data landscape or ecosystem. Also, the introduction of new technologies requires new, well-trained people who can handle the sheer growth of the technology stack.

However, managing and deploying data tools is getting easier by the day. The latest data engineering tools greatly abstract and simplify workflows, allowing data engineers to focus on selecting the simplest and most cost-effective solutions that deliver the greatest value to the business. As a result, these incremental developments in the data tooling landscape are expected to significantly reduce the operational complexity of deploying future data architectures.

(x) Batch computing challenges: Most frameworks that rely on batch data ETL using SQL or SQL-like functionality can be difficult to integrate when complex logic is required compared to simple low-level programming. Batch computing queries become problematic when resources are limited because aggregations are not additive when new elements are added to the computed results. As a result, batch computing can also be inflexible and difficult to manage, which can lead to errors.

When using third party solutions to improve efficiency, utilization, and performance, some dependencies and versions (e.g., jobs that depend on Spark or Hive versions) can be difficult to integrate due to heterogeneity in workloads (e.g., analytic, transactional), infrastructure (e.g., cloud, on-premises), deployments (e.g., Kubernetes, bare-metal nodes, custom Platform as a Service (PaaS)) and data pipeline environment, increasing operational overhead. To overcome these challenges, frameworks such as Kubernetes with its containerized approach can be used.

(xi) System stability: System stability depends on both data consistency and the longevity of the systems used. If the same data engineering pipeline cluster is used by multiple use cases, the entire cluster may become unstable during sudden traffic spikes. For example, a throughput-intensive application may impact or slow down the data availability of another application or service if the pipelines are not adequately planned.

Solving data consistency problems caused by multiple systems is a difficult engineering challenge. Data inconsistencies can lead to data loss or duplication of data. Bringing these events to a consistent state requires additional effort from the data and operations engineering teams. As a platform for data orchestration and management, the multi-tenancy support in Kubernetes aims to enable such isolation and fair resource sharing between multiple use cases so that their workloads can be reliably shared in a single cluster. However, this approach should also be extended to the E2E components of the entire data engineering pipeline.

(xii) Data sharing: Many telecommunication service providers struggle to understand and identify what data should be shared, while ensuring regulatory compliance and defining/understanding the standardized interfaces for data sharing. Implementing a secure and distributed approach to data sharing between different network nodes is critical. In future networks, for example, many telecommunication service providers will have to share part of their infrastructure with each other. This will also force them to share critical infrastructure-related information for better network management and orchestration [218].

Traditional APIs used for data sharing can perform poorly (a slow-working API can be a bottleneck for the service) or converting legacy services to an API-based service can be costly. Monolithic databases where every user retrieves the data can lead to a single point of failure and scalability issues. Big Data transfers can also have consistency issues due to the longer duration of data transfers when data is shared.

To address these data sharing issues in a scalable way and improve the quality of data sharing with third-party vendors or internal departments of an organization, investments can be made in building standardized interfaces for accessing relevant data and event-based applications and architectures for complex events. Another possible solution for data sharing is to integrate the latest developments in blockchain-based systems into telecommunication networks [219] so that intelligence and data can be shared between the owners of the individual network domains in a secure and reliable manner.

(xiii) Coexistence with non-AI/ML capable systems: In a traditional telecommunications infrastructure, not all deployed equipment will be intelligent. In some cases, the AI/ML-enabled systems will need to interact with non-AI/ML-enabled systems. For example, in some situations, some of the UEs/edge devices may be used for model training while others act as normal mobile/edge devices. In this scenario, the AI/ML platform should be able to distinguish AIML-enabled nodes. This can help AI/ML nodes to participate in distributed model training or model service and ensure unexpected interventions (e.g., interference, traffic, congestion, etc.) by non-AI/ML nodes.

Moreover, human-motivated actions or misleading behaviours that can be performed on nodes not operated by AI/ML may negatively affect the learning process of AI/ML-enabled nodes. On the other hand, in some cases, especially during the autonomous learning process (e.g., during the

TABLE 7. Comparisons of architectural choices.

Architecture Options	Description & Advantages		Drawbacks
Lambda	—Suggested by Nathan Maz and is utilized to support systems that require both streaming and batch pipelines [217]. —The aim is to make streaming systems to aid batch systems to be as close as possible to real-time. —Enables reliable streaming pipeline establishments.		—May cause maintenance problems due to two separate codebases (one for streaming and another for batch) which may need to be maintained for consistency during software updates and fixes together. —Increased complexity, cost as well as inaccuracy due to potential loss or duplication of data. —May cause system integration and data centralization difficulties. —Not suitable for use cases requiring correct and low-latency results.
Kappa	—Suggested by Jay Kreps and addresses some of the challenges and limitations of Lambda architecture, —Replay the data from a structure data source into a stream (e.g turning tables into unbounded stream) to provide unified processing capabilities. —Unifies both batch and streaming pipelines under a unified codebase and facilitates the use of batch and streaming data to drive business innovation.		—Requires fast stream processing engine —Not efficient storage for large data sets —Less efficient processing capabilities for batch —Limited cloud native support
Microservice compatible	**Serverless data pipeline**	—Promoted by AWS Lambda and one of the latest architectural trend which exploit serverless framework to have an infrastructure and orchestrate data pipeline using configuration files. —More serverless platform examples with Google Cloud Functions, Nuclio, Pivotal Function Service, Azure Functions for automated data science. —Defines serverless events and functions as a service —Based on function model where each operation is short-lived and does single operation —Significantly simplifies the infrastructure data pipeline instantiation process and accelerates ML deployment cycle by providing high elasticity via Cloud Provider. —No manual configurations, e.g. on API Gateways, etc. —Low cost since only pay what you need —Scalability is instant and handled by cloud providers. (Scale up and down based on traffic patterns) —Cloud providers handle maintenance —Deployment time is in milliseconds —Simplified and very fast application development process (easy Minimum Viable Products (MVPs) launch)	—Testing is difficult since backend architecture needs to be replicated locally —Portability is difficult from one cloud provider to another (needs substantial codebase change) —Limited freedom for resource management and application control —Latency is high (cold start is common issue) —Problematic for long running batch processing applications (serverless functions have time limitations) —Limited language support —Rapid function scaling is problematic for connection oriented data sources such as relational databases and message brokers. —Standardized choices —Complex configurations —Issues with interfacing with storage/databases —Does not scale large well
	Container based data pipeline	—Custom code and services —Deployment time is in seconds —Testing is very simple as the same container is in both local and cloud environment —Agnostic to cloud provider (easy to port the codebase) —Provides low latency —Good options for long running batch processing application deployments —Full resource and application control is possible —Language support in unlimited —Lots of choices of frameworks and API mechanisms	—High cost since containers always run —Scalability is handled by developers. —Developers need to constantly update the containers security, etc. for maintenance purposes. —Slower development process due to management and processing for setting up environment

exploration phase of reinforcement learning algorithms), the actions performed by AI/ML-enabled devices are unreliable. To avoid unexpected behaviour of telecommunication systems in this case, non AI/ML-enabled nodes can be used until the learning process is successfully completed.

(xiv) Small dataset: One of the components for building a data engineering pipeline is the data storage ecosystem, in which HDFS plays a key role. However, HDFS also brings practical limitations in storing a large amount of small data. For this reason, the size of files that need to be stored in databases, or the amount of data that can be sent to a web service that feeds the data into the database must be adjusted accordingly via configuration parameters. As a result, applications that rely on HDFS (e.g., Spark jobs) slow down because the applications spend most of its time on I/O operations instead of focusing on data processing or analysis aspects. A possible solution to this problem in Hadoop would be to store data in SequenceFile format where each small file is stored in a larger single file.

(xv) Testing distributed systems: Testing a distributed system with multiple components operating in spatially separated locations is usually more difficult and complicated. In a typical data engineering workflow used to develop a system, determining where the system fails requires additional investigative work as the components involved in the data pipeline grow larger. For a typical system, several proven types of traditional software testing must be performed before the system is put into production. Similar procedures can be used when testing data engineering pipelines. These procedures are: (i) unit testing (to test a small part or subset of the functionality of a data engineering component), (ii) regression testing (to reproduce the previously found bugs and fix), (iii) integration testing (to test the system, when individual data engineering components are integrated with each other), (iv) E2E testing (to test full functionality of a data engineering system in a staging environment) and (v) stress testing (to test the scalability limits of the data engineering system on a large scale, e.g., number of users supported, data traffic support, number of commands executed, etc.).

(xvi) Lack of standardization: There are many data-driven telecommunication use cases in the standardization community, but no implementation details for real-world telecommunication network environments (e.g., in 5G and beyond). This can lead to significant challenges in building robust data engineering pipelines when extending enterprise building blocks. Although some strategies are presented in [220], they are limited to procedures of BDA techniques in IT with limited applications in the network domain. However, extensive standardization efforts are needed to generalize these concepts for used by a large community.

(xvii) Backlogged data pipelines: A common challenge for all pipelines is the delays that occur in the ingestion pipeline. Note that in telecommunication networks, especially in 5G and beyond mobile networks, SLAs are very strict when time-critical AI/ML-based decision making processes need to be made. Thus, if any component of the data pipeline fails, it can lead to serious SLA misses. In networks, for example, the packet transmission times are on the order of milliseconds and the inference time of the developed models should be an order of magnitude shorter, otherwise there will be overhead as traffic increases.

(xviii) Limited data availability: Unavailability of sufficient data for AI/ML training and model building purposes is a major challenge in almost all industrial use cases [221]. Every time new data emerges, it also brings new knowledge and hence needs to be integrated into the training process. This is also true for the telecommunication industry. At the same time, recent advances such as semi-supervised learning, federated learning or active learning can help to create larger training datasets or to introduce new knowledge into the training process.

For a good summary of existing ML approaches that work with limited data see [222]. Among these algorithms, semi-supervised learning aims to train ML models that use both labelled and unlabeled data [223]. These methods use a large amount of unlabeled data and proportional lack of labelled data to achieve an optimal result. For example, the labelling of unlabelled examples can be done by a semi-supervised algorithm based on their proximity to known labelled examples. The main advantage of this approach is that it generates additional labelled data that can be used to train the ML model. Therefore, semi-supervised learning is particularly beneficial for scenarios where more training data is needed.

The concept of federated learning aims to distribute the copies of the ML algorithm to the distributed sites/devices where the data is kept, perform the training iterations locally, and finally send the computational results (e.g., updated neural network weights) to the central repository to update the main algorithm [212]. The main advantage is that the data remains with the owner and the algorithms can still be trained on the distributed data. Active learning aims to reduce the amount of data required for human labelling [224]. In this learning method, a query-based strategy is used to select the most informative examples that a human operator can label. Once new examples are labelled, the ML model is updated based on the newly labelled examples and this process is repeated to train the model and improve performance.

In telecommunications, unlabeled instances can be selected for active labeling. For example, when it is difficult to obtain enough labelled network fault data to find the root cause of faults in cellular networks, an active learning strategy can be used [225], [226]. In semi-supervised learning cases, auto-encoder based approaches can be used to find the root cause of faults in cellular networks [182]. Finally, federated learning in wireless communication allows each UE to build local federated learning models based on their local measurements and send them to BSs to build a global federated learning model [227].

Finally Table 8 provides a summary of the challenges described above.

TABLE 8. Summary of the challenges.

Challenges (I)	Description (I)
(i) Interworking between different programming languages, tools, computation runtimes	The field of data tools and systems is inherently heterogeneous, diverse and fragmented, as multiple workflows are involved in the process of creating data engineering pipelines.
(ii) Choosing the right toolset	It can be difficult to find a suitable set of tools for building a data pipeline given different use case requirements
(iii) Support for containerization	There are still many components in the Hadoop ecosystem that have not moved away from YARN or do not provide standard support.
(iv) Lack of a unified framework for data processing and analysis	It is a difficult task to integrate all separate analytics nodes with traditional systems in a single framework
(v) Use of multiple AI/ML frameworks	In cross-domain applications where many different computational frameworks or patterns need to be combined, serious challenges can arise

Challenges (II)	Description (II)
(vi) Data security and privacy	A large data set distributed across an enterprise can cause major headaches for data owners in terms of security, authentication, authorization, privacy and information integrity.
(vii) Event streaming support	Traditional event streaming in OSS rely on various vendor-specific protocols which also bring their own challenges.
(viii) Architecture decisions	Different data architectures need to be chosen depending on the use case and SLAs
(ix) Operational complexity	There will not be a single platform, system or compute runtime that can handle all the entire underlying heterogeneous data engineering infrastructure
(x) Batch computing challenge	Batch computing queries become problematic when resources are limited because aggregations are not additive when new elements are added to the computed results.

Challenges (III)	Description (III)
(xi) System stability	If the same data engineering pipeline cluster is used by multiple use cases, the entire cluster may become unstable during sudden traffic spikes
(xii) Data sharing	Implementing a secure and distributed approach to data sharing between different network nodes is challenging
(xiii) Coexistence with non-AI/ML capable systems	In cases when the AI/ML-enabled systems will need to interact with non-AI/ML-enabled system, the AI/ML platform should be able to distinguish AI/ML-enabled nodes
(xiv) Small dataset	The size of files that need to be stored in databases, or the amount of data that can be sent to a web service that feeds the data into the database must be adjusted accordingly

Challenges (IV)	Description (IV)
(xv) Testing distributed systems	In a typical data engineering workflow used to develop a system, determining where the system fails requires additional investigative work as the components involved in the data pipeline grow larger.
(xvi) Lack of standardization	Many data-driven telecommunication use cases in the standardization community, but no implementation details for real-world telecommunication network environments
(xvii) Backlogged data pipelines	Delays that occur in the ingestion pipeline can cause problems in SLAs in telecommunication networks when time-critical AI/ML-based decision making processes need to be made.
(xviii) Limited data availability	Unavailability of sufficient data for AI/ML training and model building purposes is a major challenge

C. FUTURE DIRECTIONS AND ROAD AHEAD

Today, we have enormously large datasets, increased computing power (GPUs, cloud, etc.), extensive open source software tools and increased industry investment as well as a large community developing new data science/engineering applications and services. Similarly, data applications are attracting large scale number of users and the data engineering ecosystem has the potential to support a larger number of users. The involvement of telecommunication industry in data value chain would provide strategic business value to telecommunication infrastructure and service providers. For this reason, telecommunication providers are looking forward to interacting with and benefiting from the data engineering ecosystem more frequently as it is open source, royalty-free and community-driven.

At the same time, there is still much to be done to develop, deploy, enable operation, debug/test and extend the data applications within the telecommunication infrastructure. It is critical to identify the applications, services, and products that will benefit most from transformations of data engineering in the networks and IT organizations of telecommunications providers. When designing a data engineering pipeline in an enterprise based on use case requirements, both telecommunication and data engineering experts should be consulted, as their perspective on each use case is different and shared ideas can be of great benefit.

Data infrastructure is already undergoing a significant architectural change [228]. Traditional data warehouses are moving from on-premise to cloud-based data warehouses to increase scalability, wide expansion, flexibility and ease of use (e.g. e-commerce data migration case to Google BigQuery[121]), and next-generation Data Lakes are beginning to include more ACID-like features and interactive SQL query capabilities (e.g. Presto [83]). More flexible and consistent Extract-Load-Transform (ELT) pipelines are taking the place of traditional ETL processes, (e.g., dbt[122]). In the area of data management and orchestration, several hundred data pipelines are orchestrated using dataflow automation tools (e.g., with AirFlow, Dagster[123]).

Tools like Superset help provide self-service insights (reports, dashboards, etc.) and are also accessible to non-technical users. Our survey results shows that telecommunication providers can move beyond the traditional boundaries of telecommunication networks (e.g., RAN or OSS/BSS operations, etc.) to reap the benefits of deploying data engineering frameworks on an evolving data infrastructure. They can leverage the power of data engineering systems deployed in a distributed, scalable and optimized architecture for their own business needs. This would also result in lower Operating Expenditures (OPEX) and Capital Expenditures (CAPEX), simplify network deployment and management, and ensure high customer satisfaction in addition to improved value-added services.

For optimized network management and orchestration, the coexistence of the data engineering frameworks described above with traditional systems at different layers of the network infrastructure is critical. The integration of the tools and frameworks of the data engineering ecosystem should serve as a complement to the traditional systems. For example, if the deployed data processing and analysis framework is not able to adequately handle the dynamic changes in the network environment, existing non-AI/ML-based solutions (e.g., predefined hand-crafted, and rule-based approaches that do not consider model-based approaches and take reactive actions based on human experience) can meet the new requirements. This coexistence is also important for security reasons. Enabling such hybrid approaches can help ensure rapid response in production environments.

Typically, individual service technologies in the telecommunications world are implemented by multiple vendors and devices running on their network are usually locked-in and expensive. Compared to telecommunication infrastructure, the data engineering infrastructure is young, innovative, and growing rapidly. Data engineering technologies and platforms are evolving and improving at a rapid pace. In addition, most of the innovative and disruptive technologies being developed in the data engineering community are being released as open source. For this reason, telecommunication systems must be prepared to adopt and deal with these new data engineering technologies rather than remain in legacy systems that cannot take advantage of the data.

For example, by starting with a simple, small E2E data engineering pipeline in a production environment, rather than working on a more complex data pipeline, can help to avoid numerous mistakes, detect errors early, and solve integration issues with traditional telecommunication infrastructure. In addition, AI/ML solutions do not have to found for every task and every problem. In many cases, simple solutions such as rule-based systems instead of ML systems can also help to find an intermediate solution. These results can be used later and iterated step by step to collect more data needed for more complex data engineering solutions. Moreover, the functionality of the data engineering pipeline can be progressively extended during this process through a series of iterations. For new and untested frameworks, it makes sense for organizations to use public cloud resources first (e.g., AWS, GCP or Azure) and then move to on-premises resources once a stable definition of workload pipeline is in place.

AI/ML algorithms are predicted to be integrated into telecommunication networks in the next decade. There are more and more number of use cases for real-time data, and the systems that process this data should be mature for the requirements of telecommunication systems. Network management and orchestration based on data engineering can be used to track evolving traffic patterns, user behaviour, etc., and take these trends into account in the planning and

[121] https://cloud.google.com/blog/products/data-analytics/e-commerce-data-warehouse-migration, accessed December-2021
[122] https://www.getdbt.com/, accessed December-2021
[123] https://dagster.io/, accessed December-2021

operational phases. It is important to choose a modular system that can cover multiple use cases.

As a starting point, there is no need to reinvent the wheel, as there is a good chance that an existing tool/framework can support initial efforts to integrate AI/ML systems into the telecommunication specific applications (both in IT and network). Familiarity with the data engineering ecosystem and tools/frameworks, diversity of expertise across technologies, and integration skills in bringing disparate pieces together into new telecommunication applications will be very useful for network-focused product and service development teams. Moving forward, a few useful questions to consider are:

- How easily can a new framework or approach be integrated and tested on large scale in the telecommunication infrastructure?
- How accurately can the impact of the new changes/updates in the data engineering pipeline be measured in telecommunication infrastructure to avoid system complexity, poor resource utilization or degradation of the KPIs for a particular service?
- Does the improvement of a framework in the pipeline affect or degrade other components in the data engineering pipeline and telecommunication infrastructure while keeping maintenance tasks at a low level?
- How does the addition of each framework in the data engineering pipeline in an integrated environment impact an organization's policy standards (e.g., data storage, compliance and regulations, security, network, operations, management, data traffic flow, servers, workloads, legacy applications, or reporting)?
- How quickly can the network engineers of the telecommunication world and the data engineers of the data engineering world would be brought together to accelerate the process?

XIII. CONCLUSION

The data engineering ecosystem will inevitably play an important role in next generation network management and orchestration systems. In this tutorial paper, we highlight the recent advances in data engineering based networks to meet the needs of network management and orchestration. We first provide a comprehensive analysis of existing frameworks and platforms, and then focus on recent standardization activities. Finally, we discuss the gaps, challenges, and future directions in building a data engineering-oritented networking system for telecommunication networks. Our tutorial analysis shows that data engineering frameworks can be used for a variety of purposes, ranging from data ingestion to data visualization, enabling telecommunication network operators to leverage the data generated by their users, environment, or network equipment.

REFERENCES

[1] *Experiential Networked Intelligence (ENI) Use Cases*, Standard ETSI GR ENI 001, Version 1.1(2018-04), ETSI, 2020.

[2] *Study of Enablers for Network Automation for 5G*, document TR 23.791, Version 16.2.0(2019-06), Release 16, 3GPP, 2020.

[3] *Machine Learning in Future Networks Including IMT-2020: Use Cases*, document Y.Sup55 ITU-T Y.3170-series, ITU, 2020.

[4] GSMA. (2019). *AI in Network Use Cases in China*. Accessed: Mar. 2022. [Online]. Available: https://bit.ly/3k8l2K7

[5] Nokia. (2020). *5G Use Cases and Requirements*. Accessed: Mar. 2022. [Online]. Available: https://bit.ly/2H7mLl4

[6] S. E. Elayoubi, M. Fallgren, P. Spapis, G. Zimmermann, D. Martin-Sacristan, C. Yang, S. Jeux, P. Agyapong, L. Campoy, Y. Qi, and S. Singh, "5G service requirements and operational use cases: Analysis and METIS II vision," in *Proc. Eur. Conf. Netw. Commun. (EuCNC)*, Jun. 2016, pp. 158–162.

[7] J. Navarro-Ortiz, P. Romero-Diaz, S. Sendra, P. Ameigeiras, J. J. Ramos-Munoz, and J. M. Lopez-Soler, "A survey on 5G usage scenarios and traffic models," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 2, pp. 905–929, 2nd Quart., 2020.

[8] G. Nguyen, S. Dlugolinsky, M. Bobák, V. Tran, A. L. García, I. Heredia, P. Malík, and L. Hluchý, "Machine learning and deep learning frameworks and libraries for large-scale data mining: A survey," *Artif. Intell. Rev.*, vol. 52, no. 1, pp. 77–124, 2019.

[9] *AI and ML—Enablers for Beyond 5G Networks*, 5GPPP Technology Board, Version 1, May 2021, doi: 10.5281/zenodo.4299895

[10] C. Zhang, P. Patras, and H. Haddadi, "Deep learning in mobile and wireless networking: A survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 3, pp. 2224–2287, 3rd Quart., 2019.

[11] A. Zappone, M. Di Renzo, and M. Debbah, "Wireless networks design in the era of deep learning: Model-based, AI-based, or both?" *IEEE Trans. Commun.*, vol. 67, no. 10, pp. 7331–7376, Oct. 2019.

[12] Q. Mao, F. Hu, and Q. Hao, "Deep learning for intelligent wireless networks: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 2595–2621, 4th Quart., 2018.

[13] M. Usama, J. Qadir, A. Raza, H. Arif, K.-L.-A. Yau, Y. Elkhatib, A. Hussain, and A. Al-Fuqaha, "Unsupervised machine learning for networking: Techniques, applications and research challenges," *IEEE Access*, vol. 7, pp. 65579–65615, 2019.

[14] B. Ma, W. Guo, and J. Zhang, "A survey of online data-driven proactive 5G network optimisation using machine learning," *IEEE Access*, vol. 8, pp. 35606–35637, 2020.

[15] P. V. Klaine, M. A. Imran, O. Onireti, and R. D. Souza, "A survey of machine learning techniques applied to self-organizing cellular networks," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2392–2431, 4th Quart., 2017.

[16] M. E. Morocho-Cayamcela, H. Lee, and W. Lim, "Machine learning for 5G/B5G mobile and wireless communications: Potential, limitations, and future directions," *IEEE Access*, vol. 7, pp. 137184–137206, 2019.

[17] J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, and Y. Liu, "A survey of machine learning techniques applied to software defined networking (SDN): Research issues and challenges," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 1, pp. 393–430, 1st Quart., 2019.

[18] Y. Zhao, Y. Li, X. Zhang, G. Geng, W. Zhang, and Y. Sun, "A survey of networking applications applying the software defined networking concept based on machine learning," *IEEE Access*, vol. 7, pp. 95397–95417, 2019.

[19] C.-X. Wang, M. D. Renzo, S. Stanczak, S. Wang, and E. G. Larsson, "Artificial intelligence enabled wireless networking for 5G and beyond: Recent advances and future challenges," *IEEE Wireless Commun.*, vol. 27, no. 1, pp. 16–23, Feb. 2020.

[20] Y. Sun, M. Peng, Y. Zhou, Y. Huang, and S. Mao, "Application of machine learning in wireless networks: Key techniques and open issues," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 4, pp. 3072–3108, 4th Quart., 2019.

[21] M. Kulin, C. Fortuna, E. De Poorter, D. Deschrijver, and I. Moerman, "Data-driven design of intelligent wireless networks: An overview and tutorial," *Sensors*, vol. 16, no. 6, p. 790, 2016.

[22] H. Muccini and K. Vaidhyanathan, "Software architecture for ML-based systems: What exists and what lies ahead," in *Proc. IEEE/ACM 1st Workshop AI Eng., Softw. Eng. AI (WAIN)*, May 2021, pp. 121–128.

[23] B. Mao, F. Tang, K. Yuichi, and N. Kato, "AI based service management for 6G green communications," 2021, *arXiv:2101.01588*.

[24] P. Casas, "Two decades of AI4NETS—AI/ML for data networks: Challenges & research directions," in *Proc. IEEE/IFIP Netw. Oper. Manage. Symp. (NOMS)*, Apr. 2020, pp. 1–6.

[25] H. Fourati, R. Maaloul, and L. Chaari, "A survey of 5G network systems: Challenges and machine learning approaches," *Int. J. Mach. Learn. Cybern.*, vol. 12, no. 2, pp. 385–431, Feb. 2021.
[26] J. Moysen and L. Giupponi, "From 4G to 5G: Self-organized network management meets machine learning," *Comput. Commun.*, vol. 129, pp. 248–268, Sep. 2018.
[27] A. Asghar, H. Farooq, and A. Imran, "Self-healing in emerging cellular networks: Review, challenges, and research directions," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 3, pp. 1682–1709, 3rd Quart., 2018.
[28] S. Yu, X. Lin, J. Misic, and X. S. Shen, *Networking for Big Data*, vol. 2. Boca Raton, FL, USA: CRC Press, 2015.
[29] C. Benzaid and T. Taleb, "AI-driven zero touch network and service management in 5G and beyond: Challenges and research directions," *IEEE Netw.*, vol. 34, no. 2, pp. 186–194, Mar. 2020.
[30] A. D'Alconzo, I. Drago, A. Morichetaa, M. Mellia, and P. Casas, "A survey on big data for network traffic monitoring and analysis," *IEEE Trans. Netw. Service Manag.*, vol. 16, no. 3, pp. 800–813, Sep. 2019.
[31] S. Verma, Y. Kawamoto, Z. M. Fadlullah, H. Nishiyama, and N. Kato, "A survey on network methodologies for real-time analytics of massive IoT data and open research issues," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1457–1477, 3rd Quart., 2017.
[32] M. S. Hadi, A. Q. Lawey, T. E. H. El-Gorashi, and J. M. H. Elmirghani, "Big data analytics for wireless and wired network design: A survey," *Comput. Netw.*, vol. 132, pp. 180–199, Feb. 2018.
[33] D. Geng, C. Zhang, C. Xia, X. Xia, Q. Liu, and X. Fu, "Big data-based improved data acquisition and storage system for designing industrial data platform," *IEEE Access*, vol. 7, pp. 44574–44582, 2019.
[34] Y. Cui, S. Kara, and K. C. Chan, "Manufacturing big data ecosystem: A systematic literature review," *Robot. Comput.-Integr. Manuf.*, vol. 62, Apr. 2020, Art. no. 101861.
[35] M. Chen, D. Gündüz, K. Huang, W. Saad, M. Bennis, A. V. Feljan, and H. V. Poor, "Distributed learning in wireless networks: Recent progress and future challenges," 2021, *arXiv:2104.02151*.
[36] W. Inoubli, S. Aridhi, H. Mezni, M. Maddouri, and E. M. Nguifo, "An experimental survey on big data frameworks," *Future Gener. Comput. Syst.*, vol. 86, pp. 546–564, Sep. 2018.
[37] H. Zahid, T. Mahmood, A. Morshed, and T. Sellis, "Big data analytics in telecommunications: Literature review and architecture recommendations," *IEEE/CAA J. Autom. Sinica*, vol. 7, no. 1, pp. 18–38, Jan. 2020.
[38] H.-N. Dai, R. C.-W. Wong, H. Wang, Z. Zheng, and A. V. Vasilakos, "Big data analytics for large-scale wireless networks: Challenges and opportunities," *ACM Comput. Surv.*, vol. 52, no. 5, pp. 1–36, 2019.
[39] W. Xia, P. Zhao, Y. Wen, and H. Xie, "A survey on data center networking (DCN): Infrastructure and operations," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 1, pp. 640–656, 1st Quart., 2016.
[40] A. A. Gebremariam, M. Usman, and M. Qaraqe, "Applications of artificial intelligence and machine learning in the area of SDN and NFV: A survey," in *Proc. 16th Int. Multi-Conf. Syst., Signals Devices (SSD)*, Mar. 2019, pp. 545–549.
[41] T. White, *Hadoop: The Definitive Guide*. Sebastopol, CA, USA: O'Reilly Media, 2012.
[42] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.
[43] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache Hadoop YARN: Yet another resource negotiator," in *Proc. 4th Annu. Symp. Cloud Comput.*, Oct. 2013, pp. 1–16.
[44] S. E. Solmaz, B. Gedik, H. Ferhatosmanoğlu, S. Sözüer, E. Zeydan, and Ç. Ö. Etemoğlu, "ALACA: A platform for dynamic alarm collection and alert notification in network management systems," *Int. J. Netw. Manage.*, vol. 27, no. 4, p. e1980, Jul. 2017.
[45] N. Garg, *Apache Kafka*. Birmingham, U.K.: Packt Publishing, 2013.
[46] S. Hoffman, *Apache Flume: Distributed Log Collection for Hadoop*. Birmingham, U.K.: Packt Publishing, 2013.
[47] X. Cheng, L. Fang, and L. Yang, "Mobile big data based network intelligence," *IEEE Internet Things J.*, vol. 5, no. 6, pp. 4365–4379, Dec. 2018.
[48] G. J. Chen, J. L. Wiener, S. Iyer, A. Jaiswal, R. Lei, N. Simha, W. Wang, K. Wilfong, T. Williamson, and S. Yilmaz, "Realtime data processing at Facebook," in *Proc. Int. Conf. Manage. Data*, Jun. 2016, pp. 1087–1098.
[49] L. Johansson and D. Dossot, *RabbitMQ Essentials: Build Distributed and Scalable Applications with Message Queuing Using RabbitMQ*. Birmingham, U.K.: Packt Publishing, 2020.
[50] A. E. Bagaskara, S. Setyorini, and A. A. Wardana, "Performance analysis of message broker for communication in fog computing," in *Proc. 12th Int. Conf. Inf. Technol. Electr. Eng. (ICITEE)*, Oct. 2020, pp. 98–103.
[51] X. J. Hong, H. S. Yang, and Y. H. Kim, "Performance analysis of RESTful API and RabbitMQ for microservice web application," in *Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, Oct. 2018, pp. 257–259.
[52] Y. Fu and C. Soman, "Real-time data infrastructure at Uber," in *Proc. Int. Conf. Manage. Data*, Jun. 2021, pp. 2503–2516.
[53] T. Akidau, S. Chernyak, and R. Lax, *Streaming Systems: The What, Where, When, and How of Large-Scale Data Processing*. Sebastopol, CA, USA: O'Reilly Media, 2018.
[54] A. Jain and A. Nalya, *Learning Storm*. Birmingham, U.K.: Packt Publishing, 2014.
[55] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica, "Apache spark: A unified engine for big data processing," *Commun. ACM*, vol. 59, no. 11, pp. 56–65, 2016.
[56] C. King, A. Higham, and S. Guo. (2020). *Pulsar vs. Kafka—Part 1—A More Accurate Perspective on Performance, Architecture, and Features*. Accessed: Mar. 2022. [Online]. Available: https://bit.ly/3tCIgj6
[57] T. Dunning and E. Friedman, *Streaming Architecture: New Designs Using Apache Kafka and MapR Streams*. Sebastopol, CA, USA: O'Reilly Media, 2016.
[58] Hughes Network Systems. (2017). *OSS/BSS Services*. Accessed: Mar. 2022. [Online]. Available: https://bit.ly/332K86U
[59] F. Yang, E. Tschetter, X. Léauté, N. Ray, G. Merlino, and D. Ganguli, "Druid: A real-time analytical data store," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, Jun. 2014, pp. 157–168.
[60] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu, and R. Murthy, "Hive—A petabyte scale data warehouse using Hadoop," in *Proc. IEEE 26th Int. Conf. Data Eng. (ICDE)*, 2010, pp. 996–1005.
[61] C. Gormley and Z. Tong, *Elasticsearch: The Definitive Guide: A Distributed Real-Time Search and Analytics Engine*. Sebastopol, CA, USA: O'Reilly Media, 2015.
[62] M. Fu, A. Agrawal, A. Floratou, B. Graham, A. Jorgensen, R. Li, N. Lu, K. Ramasamy, S. Rao, and C. Wang, "Twitter heron: Towards extensible streaming engines," in *Proc. IEEE 33rd Int. Conf. Data Eng. (ICDE)*, Apr. 2017, pp. 1165–1172.
[63] J. Carlson, *Redis in Action*. New York, NY, USA: Simon and Schuster, 2013.
[64] K. Ting and J. J. Cecho, *Apache Sqoop Cookbook: Unlocking Hadoop for Your Relational Database*. Sebastopol, CA, USA: O'Reilly Media, 2013.
[65] C. Ibsen and J. Anstey, *Camel in Action*. Shelter Island, NY, USA: Manning Publications, 2010.
[66] G. Fu, Y. Zhang, and G. Yu, "A fair comparison of message queuing systems," *IEEE Access*, vol. 9, pp. 421–432, 2021.
[67] M. Zhang. (2020). *A Curated List of Awesome Streaming (Stream Processing) Frameworks, Applications, Readings and Other Resources*. Accessed: Mar. 2022. [Online]. Available: https://github. com/manuzhang/awesome-streaming
[68] D. L. Quoc, R. Chen, P. Bhatotia, C. Fetzer, V. Hilt, and T. Strufe, "StreamApprox: Approximate computing for stream analytics," in *Proc. 18th ACM/IFIP/USENIX Middleware Conf.*, Dec. 2017, pp. 185–197.
[69] D. Merkel, "Docker: Lightweight Linux containers for consistent development and deployment," *Linux J.*, vol. 2014, no. 239, p. 2, 2014.
[70] C. Olston, N. Fiedel, K. Gorovoy, J. Harmsen, L. Lao, F. Li, V. Rajashekhar, S. Ramesh, and J. Soyke, "TensorFlow-Serving: Flexible, high-performance ML serving," 2017, *arXiv:1712.06139*.
[71] F. Balali, J. Nouri, A. Nasiri, and T. Zhao, "Data analytics," in *Data Intensive Industrial Asset Management*. Cham, Switzerland: Springer, 2020, pp. 105–113.
[72] D. L. Olson and G. Lauhoff, "Descriptive data mining," in *Descriptive Data Mining*. Singapore: Springer, 2019, pp. 129–130.
[73] H. Cao, M. Wachowicz, C. Renso, and E. Carlini, "Analytics everywhere: Generating insights from the Internet of Things," *IEEE Access*, vol. 7, pp. 71749–71769, 2019.
[74] P. S. Deshpande, S. C. Sharma, and S. K. Peddoju, "Predictive and prescriptive analytics in big-data era," in *Security and Data Storage Aspect in Cloud Computing*. Singapore: Springer, 2019, pp. 71–81.
[75] J. Hermann and M. D. Balso. (2020). *Meet Michelangelo: Uber's Machine Learning Platform*. Accessed: Mar. 2022. [Online]. Available: https://ubr.to/3upw6cL

[76] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache Flink: Stream and batch processing in a single engine," Bull. IEEE Comput. Soc. Tech. Committee Data Eng., IEEE, NY, USA, Tech. Rep. 4, 2015, vol. 36, no. 4.

[77] J. J. Dai, Y. Wang, X. Qiu, D. Ding, Y. Zhang, Y. Wang, X. Jia, C. L. Zhang, Y. Wan, Z. Li, J. Wang, S. Huang, Z. Wu, Y. Wang, Y. Yang, B. She, D. Shi, Q. Lu, K. Huang, and G. Song, "BigDL: A distributed deep learning framework for big data," in *Proc. ACM Symp. Cloud Comput.*, 2019, pp. 50–60.

[78] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan, and I. Stoica, "Ray: A distributed framework for emerging AI applications," in *Proc. 13th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2018, pp. 561–577.

[79] S. A. Noghabi, K. Paramasivam, Y. Pan, N. Ramesh, J. Bringhurst, I. Gupta, and R. H. Campbell, "Samza: Stateful scalable stream processing at LinkedIn," *Proc. VLDB Endowment*, vol. 10, no. 12, pp. 1634–1645, 2017.

[80] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig Latin: A not-so-foreign language for data processing," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2008, pp. 1099–1110.

[81] J. Russell, *Cloudera Impala*. Sebastopol, CA, USA: O'Reilly Media, 2013.

[82] M. Hausenblas and J. Nadeau, "Apache drill: Interactive ad-hoc analysis at scale," *Big Data*, vol. 1, no. 2, pp. 100–104, Jun. 2013.

[83] R. Sethi, M. Traverso, D. Sundstrom, D. Phillips, W. Xie, Y. Sun, N. Yegitbasi, H. Jin, E. Hwang, N. Shingte, and C. Berner, "Presto: SQL on everything," in *Proc. IEEE 35th Int. Conf. Data Eng. (ICDE)*, Apr. 2019, pp. 1802–1813.

[84] A. Kylin. (2020). *Analytical Data Warehouse for Big Data*. Accessed: Mar. 2022. [Online]. Available: http://kylin.apache.org/

[85] J.-F. Im, K. Gopalakrishna, S. Subramaniam, M. Shrivastava, A. Tumbde, X. Jiang, J. Dai, S. Lee, N. Pawar, J. Li, and R. Aringunram, "Pinot: Realtime OLAP for 530 million users," in *Proc. Int. Conf. Manage. Data*, May 2018, pp. 583–594.

[86] N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, P. Selmer, and A. Taylor, "Cypher: An evolving query language for property graphs," in *Proc. Int. Conf. Manage. Data*, May 2018, pp. 1433–1445.

[87] T. Lipcon, T. Lipcon, D. Alves, D. Burkert, J.-D. Cryans, A. Dembo, M. Percy, S. Rus, D. Wang, M. Bertozzi, C. P. McCabe, and A. Wang, "Kudu: Storage for fast analytics on fast data," Cloudera, Palo Alto, CA, USA, Tech. Rep. 1, 2015, vol. 28.

[88] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proc. 7th Symp. Oper. Syst. Design Implement.*, 2006, pp. 307–320.

[89] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," *ACM SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, pp. 205–220, 2007.

[90] J. Lennon, *Beginning CouchDB*. New York, NY, USA: Apress, 2010.

[91] A. Vukotic, N. Watt, T. Abedrabbo, D. Fox, and J. Partner, *Neo4j in Action*, vol. 22. Shelter Island, NY, USA: Manning, 2015.

[92] B. Fitzpatrick, "Distributed caching with memcached," *Linux J.*, vol. 2004, no. 124, pp. 72–78, 2004.

[93] A. Lakshman and P. Malik, "Cassandra: A decentralized structured storage system," *ACM SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35–40, Apr. 2010.

[94] L. George, *HBase: The Definitive Guide: Random Access to Your Planet-Size Data*. Sebastopol, CA, USA: O'Reilly Media, 2011.

[95] M. Armbrust *et al.*, "Delta lake: High-performance acid table storage over cloud object stores," in *Proc. Int. Conf. Very Large Data Bases (VLDB)*, 2020, pp. 1–14.

[96] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "ZooKeeper: Wait-free coordination for internet-scale systems," in *Proc. USENIX Annu. Tech. Conf.*, 2010, vol. 8, no. 9, pp. 1–14.

[97] L. Calcote and Z. Butcher, *Istio: Up and Running: Using a Service Mesh to Connect, Secure, Control, and Observe*. Sebastopol, CA, USA: O'Reilly Media, 2019.

[98] J. Turnbull, *Monitoring With Prometheus*. New York, NY, USA: Turnbull Press, 2018.

[99] Amazon Web Services. (2020). *Serverless Application Lens—AWS Well-Architected Framework*. Accessed: Oct. 2020. [Online]. Available: https://go.aws/3wx0nca

[100] Linux Containers. (2020). *Infrastructure for Container Projects*. Accessed: Mar. 2022. [Online]. Available: https://linuxcontainers.org/

[101] B. Burns, J. Beda, and K. Hightower, *Kubernetes: Up and Running: Dive Into the Future of Infrastructure*. Sebastopol, CA, USA: O'Reilly Media, 2019.

[102] D. Kakadia, *Apache Mesos Essentials*. Birmingham, U.K.: Packt Publishing, 2015.

[103] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. A. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe, F. Xie, and C. Zumar, "Accelerating the machine learning lifecycle with MLflow," Bull. IEEE Comput. Soc. Tech. Committee Data Eng., IEEE, NY, USA, Tech. Rep. 4, 2018, pp. 39–45, vol. 41, no. 4.

[104] E. Bisong, "Kubeflow and kubeflow pipelines," in *Building Machine Learning and Deep Learning Models on Google Cloud Platform*. New York, NY, USA: Apress, 2019, pp. 671–685.

[105] M. Islam, A. K. Huang, M. Battisha, M. Chiang, S. Srinivasan, C. Peters, A. Neumann, and A. Abdelnur, "Oozie: Towards a scalable workflow management system for Hadoop," in *Proc. 1st ACM SIGMOD Workshop Scalable Workflow Execution Engines Technol.*, 2012, pp. 1–10.

[106] B. Saha, H. Shah, S. Seth, G. Vijayaraghavan, A. Murthy, and C. Curino, "Apache tez: A unifying framework for modeling and building data processing applications," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, May 2015, pp. 1357–1369.

[107] E. Begoli, J. Camacho-Rodríguez, J. Hyde, M. J. Mior, and D. Lemire, "Apache calcite: A foundational framework for optimized query processing over heterogeneous data sources," in *Proc. Int. Conf. Manage. Data*, May 2018, pp. 221–230.

[108] T. Grainger and T. Potter, *Solr in Action*. Shelter Island, NY, USA: Manning, 2014.

[109] A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 1–12.

[110] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar, "MLlib: Machine learning in Apache Spark," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 1235–1241, 2016.

[111] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, A. Müller, J. Nothman, G. Louppe, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Nov. 2011.

[112] J. M. Hellerstein, C. Ré, F. Schoppmann, D. Z. Wang, E. Fratkin, A. Gorajek, K. S. Ng, C. Welton, X. Feng, K. Li, and A. Kumar, "The MADlib analytics library: Or MAD skills, the SQL," *Proc. VLDB Endowment*, vol. 5, no. 12, pp. 1700–1711, Aug. 2012.

[113] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Operating Syst. Design Implement. (OSDI)*, 2016, pp. 265–283.

[114] R. Collobert, S. Bengio, and J. Mariéthoz. (2002). *Torch: A Modular Machine Learning Software Library, Idiap*. Accessed: Mar. 2022. [Online]. Available: https://bit.ly/3iD2p2c

[115] S. Tokui, R. Okuta, T. Akiba, Y. Niitani, T. Ogawa, S. Saito, S. Suzuki, K. Uenishi, B. Vogel, and H. Y. Vincent, "Chainer: A deep learning framework for accelerating the research cycle," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2019, pp. 2002–2011.

[116] B. C. Ooi, K.-L. Tan, S. Wang, W. Wang, Q. Cai, G. Chen, J. Gao, Z. Luo, A. K. H. Tung, Y. Wang, Z. Xie, M. Zhang, and K. Zheng, "SINGA: A distributed deep learning platform," in *Proc. 23rd ACM Int. Conf. Multimedia*, Oct. 2015, pp. 685–688.

[117] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems," in *Proc. Neural Inf. Process. Syst., Workshop Mach. Learn. Syst.*, 2015, pp. 1–6.

[118] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: A CPU and GPU math compiler in Python," in *Proc. Python Sci. Conf. (SciPy)*, Austin, TX, USA, 2010, vol. 4, no. 3, pp. 1–7.

[119] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proc. 22nd ACM Int. Conf. Multimedia*, Nov. 2014, pp. 675–678.

[120] F. Chollet, *Deep Learning With Python*, vol. 361. New York, NY, USA: Manning, 2018.
[121] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica, "RLlib: Abstractions for distributed reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 3053–3062.
[122] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI gym," 2016, *arXiv:1606.01540*.
[123] A. Sergeev and M. Del Balso, "Horovod: Fast and easy distributed deep learning in TensorFlow," 2018, *arXiv:1802.05799*.
[124] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, "Clipper: A low-latency online prediction serving system," in *Proc. 14th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2017, pp. 613–627.
[125] J. Bergstra, D. Yamins, and D. Cox, "Hyperopt: A Python library for optimizing the hyperparameters of machine learning algorithms," in *Proc. 12th Python Sci. Conf.* Princeton, NJ, USA: Citeseer, 2013, p. 20.
[126] Y. Vasiliev, *Natural Language Processing With Python and SpaCy: A Practical Introduction*. San Francisco, CA, USA: No Starch Press, 2020.
[127] T. Wolf *et al.*, "HuggingFace's transformers: State-of-the-art natural language processing," 2019, *arXiv:1910.03771*.
[128] M. Gardner, J. Grus, M. Neumann, O. Tafjord, P. Dasigi, N. Liu, M. Peters, M. Schmitz, and L. Zettlemoyer, "AllenNLP: A deep semantic natural language processing platform," 2018, *arXiv:1803.07640*.
[129] T. B. Brown *et al.*, "Language models are few-shot learners," 2020, *arXiv:2005.14165*.
[130] C. Ré, F. Niu, P. Gudipati, and C. Srisuwananukorn, "Overton: A data system for monitoring and improving machine-learned products," 2019, *arXiv:1909.05372*.
[131] J. Dunn, "Introducing FBLearner flow: Facebook's AI backbone," Eng. Blog, Facebook Code, Facebook, CA, USA, Tech. Rep. 1, 2016.
[132] *Zero-Touch Network and Service Management (ZSM) Reference Architecture*, Standard ETSI GS ZSM 002, Version 1.1.1(2019-08), ETSI, 2020.
[133] (Mar. 2022). *ZSM Architectural Framework for End-to-End Service and Network Automation*. Accessed: Mar. 2022. [Online]. Available: https://bit.ly/3qykGSU
[134] OSM End User Advisory Group. (2019). *OSM Scope, Functionality, Operation and Integration Guidelines*. Accessed: Mar. 2022. [Online]. Available: https://bit.ly/3iz3Q1K
[135] D. Soldani and S. A. Illingworth, "5G AI-enabled automation," in *Wiley 5G Ref: The Essential 5G Reference Online*. Hoboken, NJ, USA: Wiley, 2019, pp. 1–38.
[136] ORAN-Alliance. (2020). *Operator Defined Next Generation RAN Architecture and Interfaces*. Accessed: Mar. 2022. [Online]. Available: https://www.o-ran.org/
[137] *O-RAN Working Group 2: AI/ML Workflow Description and Requirements*. Alfter, Germany: O-RAN Alliance, 2019.
[138] *O-RAN Working Group 3: O-RAN Near-Real-Time RAN Intelligent Controller Architecture and E2 General Aspects and Principles—V1.01*, 2020.
[139] ONAP. (2020). *A Comprehensive Platform for Orchestration, Management, and Automation of Network and Edge Computing Services*. Accessed: Mar. 2022. [Online]. Available: https://www.onap.org/
[140] *Requirements Based on Documented Scenarios*, Standard ETSI GS ZSM 001, Version 1.1.1(2019-08), ETSI, 2019.
[141] *Autonomic Network Engineering for the Self-Managing Future Internet (AFI); Generic Autonomic Network Architecture; Part 2: An Architectural Reference Model for Autonomic Networking, Cognitive Networking and Self-Management*, Standard ETSI TS 103 195-2, Version 1.1.1(2018-05), 2020.
[142] *Experiential Networked Intelligence (ENI) System Architecture*, Standard ETSI GS ENI 005, Version 1.1.1(2019-09), 2020.
[143] Y. Wang, R. Forbes, C. Cavigioli, H. Wang, A. Gamelas, A. Wade, J. Strassner, S. Cai, and S. Liu, "Network management and orchestration using artificial intelligence: Overview of ETSI ENI," *IEEE Commun. Standards Mag.*, vol. 2, no. 4, pp. 58–65, Dec. 2018.
[144] L. Frost, T. B. Meriem, J. M. Bonifacio, S. Cadzow, F. da Silva, M. Essa, R. Forbes, P. Marchese, M. Odini, N. Sprecher, C. Toche, and S. Wood, "Artificial intelligence and future directions for ETSI," ETSI, Sophia Antipolis, France, ETSI White Paper #34 (2020-06), 2020.
[145] *Architectural Framework for Machine Learning in Future Networks Including IMT-2020*, document ITU-T R Y.3172 (2019-06), ITU, 2020.
[146] R. Ferrus, O. Sallent, and J. Perez-Romero, "Data analytics architectural framework for smarter radio resource management in 5G radio access networks," *IEEE Commun. Mag.*, vol. 58, no. 5, pp. 98–104, May 2020.
[147] *Architecture Enhancements for 5G System (5GS) to Support Network Data Analytics Services*, Standard TS 23.288, Version 16.1.0, 3rd Generation Partnership Project (3GPP), 2019.
[148] A. Imran, A. Zoha, and A. Abu-Dayya, "Challenges in 5G: How to empower SON with big data for enabling 5G," *IEEE Netw.*, vol. 28, no. 6, pp. 27–33, Nov./Dec. 2014.
[149] R. Li, Z. Zhao, X. Zhou, G. Ding, Y. Chen, Z. Wang, and H. Zhang, "Intelligent 5G: When cellular networks meet artificial intelligence," *IEEE Wireless Commun.*, vol. 24, no. 5, pp. 175–183, Oct. 2017.
[150] I. Malaka and I. Brown, "Challenges to the organisational adoption of big data analytics: A case study in the South African telecommunications industry," in *Proc. Annu. Res. Conf. South Afr. Inst. Comput. Scientists Inf. Technol.*, 2015, pp. 1–9.
[151] M. I. Baig, L. Shuib, and E. Yadegaridehkordi, "Big data adoption: State of the art and research challenges," *Inf. Process. Manage.*, vol. 56, no. 6, Nov. 2019, Art. no. 102095.
[152] H. Gacanin and M. Wagner, "Artificial intelligence paradigm for customer experience management in next-generation networks: Challenges and perspectives," *IEEE Netw.*, vol. 33, no. 2, pp. 188–194, Mar. 2019.
[153] S. Mwanje, G. Decarreau, C. Mannweiler, M. Naseer-ul-Islam, and L. C. Schmelz, "Network management automation in 5G: Challenges and opportunities," in *Proc. IEEE 27th Annu. Int. Symp. Pers., Indoor, Mobile Radio Commun. (PIMRC)*, Sep. 2016, pp. 1–6.
[154] W. Xu, Y. Xu, C.-H. Lee, Z. Feng, P. Zhang, and J. Lin, "Data-cognition-empowered intelligent wireless networks: Data, utilities, cognition brain, and architecture," *IEEE Wireless Commun.*, vol. 25, no. 1, pp. 56–63, Feb. 2018.
[155] T. Zhang, K. Zhu, and E. Hossain, "Data-driven machine learning techniques for self-healing in cellular wireless networks: Challenges and solutions," 2019, *arXiv:1906.06357*.
[156] A. Lavin, C. M. Gilligan-Lee, A. Visnjic, S. Ganju, D. Newman, A. G. Baydin, S. Ganguly, D. Lange, A. Sharma, S. Zheng, E. P. Xing, A. Gibson, J. Parr, C. Mattmann, and Y. Gal, "Technology readiness levels for machine learning systems," 2021, *arXiv:2101.03989*.
[157] F. Martínez-Plumed, E. Gómez, and J. Hernández-Orallo, "Futures of artificial intelligence through technology readiness levels," *Telematics Informat.*, vol. 58, May 2021, Art. no. 101525.
[158] T. Granlund, A. Kopponen, V. Stirbu, L. Myllyaho, and T. Mikkonen, "MLOps challenges in multi-organization setup: Experiences from two real-world cases," 2021, *arXiv:2103.08937*.
[159] C. Papagianni, J. Mangues-Bafalluy, P. Bermudez, S. Barmpounakis, D. De Vleeschauwer, J. Brenes, E. Zeydan, C. Casetti, C. Guimaraes, P. Murillo, A. Garcia-Saavedra, D. Corujo, and T. Pepe, "5Growth: AI-driven 5G for automation in vertical industries," in *Proc. Eur. Conf. Netw. Commun. (EuCNC)*, Jun. 2020, pp. 17–22.
[160] X. Li, A. Garcia-Saavedra, X. Costa-Perez, C. J. Bernardos, C. Guimaraes, K. Antevski, J. Mangues-Bafalluy, J. Baranda, E. Zeydan, D. Corujo, P. Iovanna, G. Landi, J. Alonso, P. Paixao, H. Martins, M. Lorenzo, J. Ordonez-Lucena, and D. R. Lopez, "5Growth: An end-to-end service platform for automated deployment and management of vertical services over 5G networks," *IEEE Commun. Mag.*, vol. 59, no. 3, pp. 84–90, Mar. 2021.
[161] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, "Hidden technical debt in machine learning systems," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 2503–2511.
[162] J.-P. Joutsenlahti, T. Lehtonen, M. Raatikainen, E. Kettunen, and T. Mikkonen, "Challenges and governance solutions for data science services based on open data and Apis," in *Proc. IEEE/ACM 1st Workshop AI Eng. Softw. Eng. AI (WAIN)*, May 2021, pp. 1–4.
[163] M. Giordani, M. Polese, M. Mezzavilla, S. Rangan, and M. Zorzi, "Toward 6G networks: Use cases and technologies," *IEEE Commun. Mag.*, vol. 58, no. 3, pp. 55–61, Dec. 2020.
[164] E. C. Strinati, S. Barbarossa, J. L. Gonzalez-Jimenez, D. Ktenas, N. Cassiau, L. Maret, and C. Dehos, "6G: The next frontier: From holographic messaging to artificial intelligence using subterahertz and visible light communication," *IEEE Veh. Technol. Mag.*, vol. 14, no. 3, pp. 42–50, Sep. 2019.
[165] N. Rajatheva *et al.*, "Scoring the terabit/s goal: Broadband connectivity in 6G," 2020, *arXiv:2008.07220*.

[166] A. Burkov, *Machine Learning Engineering Book*, vol. 1. Quebec City, QC, Canada: Andriy Burkov, 2020.
[167] B. Derakhshan, A. Mahdiraji, T. Rabl, and V. Markl, "Continuous deployment of machine learning pipelines," in *Proc. 22nd Int. Conf. Extending Database Technol. (EDBT)*, 2019, p. 397—408.
[168] Google Cloud. (2020). *MLOps: Continuous Delivery and Automation Pipelines in Machine Learning*. Accessed: Mar. 2022. [Online]. Available: https://bit.ly/3tztmu2
[169] E. Acurna and C. Rodriguez, "The treatment of missing values and its effect on classifier accuracy," in *Proc. Meeting Int. Fed. Classification Soc. (IFCS)*, 2004, pp. 639–647.
[170] S. García, J. Luengo, and F. Herrera, *Data Preprocessing in Data Mining*, vol. 72. Cham, Switzerland: Springer, 2015.
[171] M. J. Azur, E. A. Stuart, C. Frangakis, and P. J. Leaf, "Multiple imputation by chained equations: What is it and how does it work?" *Int. J. Methods Psychiatric Res.*, vol. 20, no. 1, pp. 40–49, Mar. 2011.
[172] J. Pagès, *Multiple Factor Analysis by Example Using R*. Boca Raton, FL, USA: CRC Press, 2014.
[173] F. Biessmann, T. Rukat, P. Schmidt, P. Naidu, S. Schelter, A. Taptunov, D. Lange, and D. Salinas, "DataWig: Missing value imputation for tables," *J. Mach. Learn. Res.*, vol. 20, no. 175, pp. 1–6, 2019.
[174] S. Schelter, F. Biessmann, T. Januschowski, D. Salinas, S. Seufert, and G. Szarvas, "On challenges in machine learning model management," Bull. IEEE Comput. Soc. Tech. Committee Data Eng., IEEE, NY, USA, Tech. Rep. 4, 2018, pp. 5–15, vol. 41, no. 4.
[175] T. Rukat, D. Lange, S. Schelter, and F. Biessmann, "Towards automated data quality management for machine learning," in *Proc. ML Ops Workshop Conf. ML Syst. (MLSys)*, 2020, pp. 1–3.
[176] M. Cutler, T. J. Walsh, and J. P. How, "Real-world reinforcement learning via multifidelity simulators," *IEEE Trans. Robot.*, vol. 31, no. 3, pp. 655–671, Jun. 2015.
[177] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, Oct. 2009.
[178] G. Yang, Y. Zhang, Z. He, J. Wen, Z. Ji, and Y. Li, "Machine-learning-based prediction methods for path loss and delay spread in air-to-ground millimetre-wave channels," *IET Microw., Antennas Propag.*, vol. 13, no. 8, pp. 1113–1121, 2019.
[179] C. T. Nguyen, N. Van Huynh, N. H. Chu, Y. M. Saputra, D. T. Hoang, D. N. Nguyen, Q.-V. Pham, D. Niyato, E. Dutkiewicz, and W.-J. Hwang, "Transfer learning for future wireless networks: A comprehensive survey," 2021, *arXiv:2102.07572*.
[180] L. Baier, F. Jöhren, and S. Seebacher, "Challenges in the deployment and operation of machine learning in practice," in *Proc. 27th Eur. Conf. Inf. Syst. (ECIS)*, 2019, pp. 1–15.
[181] A. Paleyes, R.-G. Urma, and N. D. Lawrence, "Challenges in deploying machine learning: A survey of case studies," 2020, *arXiv:2011.09926*.
[182] H. D. Trinh, E. Zeydan, L. Giupponi, and P. Dini, "Detecting mobile traffic anomalies through physical control channel fingerprinting: A deep semi-supervised approach," *IEEE Access*, vol. 7, pp. 152187–152201, 2019.
[183] F. D. Calabrese, P. Frank, E. Ghadimi, U. Challita, and P. Soldati, "Enhancing RAN performance with AI," *Ericsson Technol. Rev.*, vol. 12, no. 12, pp. 36–46, 2019. Accessed: Mar. 2022. [Online]. Available: https://bit.ly/3aDKJQB
[184] A. Zheng, *Evaluating Machine Learning Models: A Beginner's Guide to Key Concepts and Pitfalls*. Sebastopol, CA, USA: O'Reilly Media, 2015.
[185] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015, *arXiv:1503.02531*.
[186] L. E. Lwakatare, A. Raj, J. Bosch, H. H. Olsson, and I. Crnkovic, "A taxonomy of software engineering challenges for machine learning systems: An empirical investigation," in *Proc. Int. Conf. Agile Softw. Develop.* Cham, Switzerland: Springer, 2019, pp. 227–243.
[187] F. Belqasmi, R. Glitho, and C. Fu, "RESTful web services for service provisioning in next-generation networks: A survey," *IEEE Commun. Mag.*, vol. 49, no. 12, pp. 66–73, Dec. 2011.
[188] G. Brown. (2017). *Service-Based Architecture for 5G Core Networks, Heavy Reading White Paper*. Accessed: Jan. 2021. [Online]. Available: https://bit.ly/3o6i7nU
[189] R. Laigner, M. Kalinowski, P. Diniz, L. Barros, C. Cassino, M. Lemos, D. Arruda, S. Lifschitz, and Y. Zhou, "From a monolithic big data system to a microservices event-driven architecture," in *Proc. 46th Euromicro Conf. Softw. Eng. Adv. Appl. (SEAA)*, Aug. 2020, pp. 213–220.
[190] *Technical Specification Group Services and System Aspects; Release 15 Description; Summary of Rel-15 Work Items*, document TR21.915, Version 0.5.0, Release 15, 3GPP, 2020.
[191] A. Katsifodimos and M. Fragkoulis, "Operational stream processing: Towards scalable and consistent event-driven applications," in *Proc. 22nd Int. Conf. Extending Database Technol. (EDBT)*, 2019, pp. 682–685.
[192] E. Zeydan, O. Dedeoglu, and Y. Turk, "Experimental evaluations of TDD-based massive MIMO deployment for mobile network operators," *IEEE Access*, vol. 8, pp. 33202–33214, 2020.
[193] Y. Turk, E. Zeydan, and C. A. Akbulut, "Experimental performance evaluations of CoMP and CA in centralized radio access networks," *Telecommun. Syst.*, vol. 72, no. 1, pp. 115–130, Sep. 2019.
[194] Y. Turk, E. Zeydan, and C. A. Akbulut, "On performance analysis of single frequency network with C-RAN," *IEEE Access*, vol. 7, pp. 1502–1519, 2019.
[195] A. S. Tan and E. Zeydan, "Performance maximization of network assisted mobile data offloading with opportunistic device-to-device communications," *Comput. Netw.*, vol. 141, pp. 31–43, Aug. 2018.
[196] O. Narmanlioglu and E. Zeydan, "New era in shared cellular networks: Moving into open and virtualized platform," *Int. J. Netw. Manage.*, vol. 27, no. 6, p. e1986, Nov. 2017.
[197] O. Narmanlioglu and E. Zeydan, "Software-defined networking based network virtualization for mobile operators," *Comput. Electr. Eng.*, vol. 57, pp. 134–146, Jan. 2017.
[198] E. Zeydan, E. Bastug, M. Bennis, M. A. Kader, I. A. Karatepe, A. S. Er, and M. Debbah, "Big data caching for networking: Moving from cloud to edge," *IEEE Commun. Mag.*, vol. 54, no. 9, pp. 36–42, Sep. 2016.
[199] Y. Turk and E. Zeydan, "Satellite backhauling for next generation cellular networks: Challenges and opportunities," *IEEE Commun. Mag.*, vol. 57, no. 12, pp. 52–57, Dec. 2019.
[200] E. Zeydan and Y. Turk, "On the impact of satellite communications over mobile networks: An experimental analysis," *IEEE Trans. Veh. Technol.*, vol. 68, no. 11, pp. 11146–11157, Nov. 2019.
[201] E. Zeydan and Y. Turk, "Recent advances in intent-based networking: A survey," in *Proc. IEEE 91st Veh. Technol. Conf. (VTC-Spring)*, May 2020, pp. 1–5.
[202] B. Alturki, S. Reiff-Marganiec, and C. Perera, "A hybrid approach for data analytics for Internet of Things," in *Proc. 7th Int. Conf. Internet Things*, Oct. 2017, p. 8.
[203] H. Zou, Y. Yu, W. Tang, and H.-W.-M. Chen, "FlexAnalytics: A flexible data analytics framework for big data applications with I/O performance improvement," *Big Data Res.*, vol. 1, pp. 4–13, Aug. 2014.
[204] L. I. B. López, J. M. Vidal, and L. J. G. Villalba, "Orchestration of use-case driven analytics in 5G scenarios," *J. Ambient Intell. Hum. Comput.*, vol. 9, no. 4, pp. 1097–1117, Aug. 2018.
[205] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. 20th Int. Conf. Artif. Intell. Statist.*, 2017, pp. 1273–1282.
[206] A. Balaji, F. Corradi, A. Das, S. Pande, S. Schaafsma, and F. Catthoor, "Power-accuracy trade-offs for heartbeat classification on neural networks hardware," *J. Low Power Electron.*, vol. 14, no. 4, pp. 508–519, 2018.
[207] D. Preuveneers, I. Tsingenopoulos, and W. Joosen, "Resource usage and performance trade-offs for machine learning models in smart environments," *Sensors*, vol. 20, no. 4, p. 1176, Feb. 2020.
[208] R. Shwartz-Ziv and A. Armon, "Tabular data: Deep learning is not all you need," 2021, *arXiv:2106.03253*.
[209] I. D. Addo, S. I. Ahamed, and W. C. Chu, "A reference architecture for high-availability automatic failover between PaaS cloud providers," in *Proc. Int. Conf. Trustworthy Syst. Appl.*, Jun. 2014, pp. 14–21.
[210] F. Kalyoncu, E. Zeydan, A. Yildirim, and I. O. Yigit, "An experimental study of factor analysis over cellular network data," in *Proc. IEEE 87th Veh. Technol. Conf. (VTC Spring)*, Jun. 2018, pp. 1–5.
[211] D. Naboulsi, M. Fiore, S. Ribot, and R. Stanica, "Large-scale mobile traffic analysis: A survey," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 124–161, 1st Quart. 2015.
[212] M. Aledhari, R. Razzak, R. M. Parizi, and F. Saeed, "Federated learning: A survey on enabling technologies, protocols, and applications," *IEEE Access*, vol. 8, pp. 140699–140725, 2020.
[213] S. Sun, Z. Cao, H. Zhu, and J. Zhao, "A survey of optimization methods from a machine learning perspective," *IEEE Trans. Cybern.*, vol. 50, no. 8, pp. 3668–3681, Aug. 2019.

[214] M. A.-U.-D. Khan, M. F. Uddin, and N. Gupta, "Seven V's of big data understanding big data to extract value," in *Proc. Zone Conf. Amer. Soc. Eng. Educ.*, Apr. 2014, pp. 1–5.
[215] D. Gannon, R. Barga, and N. Sundaresan, "Cloud-native applications," *IEEE Cloud Comput.*, vol. 4, no. 5, pp. 16–21, Sep. 2017.
[216] Apache Flink. (2020). *Flink-TensorFlow: A Library for Machine Intelligence in Apache Flink, Using the TensorFlow Library and Associated Models*. Accessed: Mar. 2022. [Online]. Available: https://github.com/FlinkML/flink-tensorflow
[217] J. Warren and N. Marz, *Data: Principles and Best Practices of Scalable Real-Time Data Systems*. Shelter Island, NY, USA: Manning, 2015.
[218] Y. Türk and E. Zeydan, "On performance analysis of multioperator RAN sharing for mobile network operators," *TURKISH J. Electr. Eng. Comput. Sci.*, vol. 29, no. 2, pp. 816–830, Mar. 2021.
[219] L. Giupponi and F. Wilhelmi, "Blockchain-enabled network sharing for O-RAN in 5G and beyond," 2021, *arXiv:2107.02005*.
[220] D. Aneato, "Strategies to implement big data analytics in telecommunications organizations," Ph.D. dissertation, College Manage. Technol., Walden Univ., Minneapolis, MN, USA, 2020.
[221] A. L'Heureux, K. Grolinger, H. F. Elyamany, and M. A. M. Capretz, "Machine learning with big data: Challenges and approaches," *IEEE Access*, vol. 5, pp. 7776–7797, 2017.
[222] Defence Science and Technology Laboratory (DSTL). (2020). *Machine Learning With Limited Data—Future of AI for Defence Project Autonomy Programme*. Accessed: Mar. 2022. [Online]. Available: https://bit.ly/3gSmguR
[223] J. E. Van Engelen and H. H. Hoos, "A survey on semi-supervised learning," *Mach. Learn.*, vol. 109, no. 2, pp. 373–440, 2020.
[224] B. Settles, "Active learning literature survey," Dept. Comput. Sci., Univ. Wisconsin-Madison, Madison, WI, USA, Tech. Rep. #1648, 2009.
[225] M. Chen, K. Zhu, and B. Chen, "Root cause analysis for self-organizing cellular network: An active learning approach," *Mobile Netw. Appl.*, vol. 25, no. 6, pp. 2506–2516, Dec. 2020.
[226] M. Chen, K. Zhu, R. Wang, and D. Niyato, "Active learning-based fault diagnosis in self-organizing cellular networks," *IEEE Commun. Lett.*, vol. 24, no. 8, pp. 1734–1737, Aug. 2020.
[227] M. Chen, Z. Yang, W. Saad, C. Yin, H. V. Poor, and S. Cui, "A joint learning and communications framework for federated learning over wireless networks," *IEEE Trans. Wireless Commun.*, vol. 20, no. 1, pp. 269–283, Jan. 2021.
[228] P. Brous, M. Janssen, and P. Herder, "Next generation data infrastructures: Towards an extendable model of the asset management data infrastructure as complex adaptive system," *Complexity*, vol. 2019, Jan. 2019, Art. no. 5415828.

ENGIN ZEYDAN (Senior Member, IEEE) received the B.Sc. and M.Sc. degrees from the Department of Electrical and Electronics Engineering, Middle East Technical University, Ankara, Turkey, in 2004 and 2006, respectively, and the Ph.D. degree from the Department of Electrical and Computer Engineering, Stevens Institute of Technology, Hoboken, NJ, USA, in February 2011. From 2011 to 2016, he has worked as a Research and Development Engineer at Avea, a mobile operator in Turkey. From 2016 to 2018, he was with Turk Telekom Labs working as a Senior Research and Development Engineer. From 2015 to 2018, he was also a part-time Instructor at the Electrical and Electronics Engineering Department, Ozyegin University. He is currently working as a Senior Researcher with the Communication Networks Division, Centre Tecnològic de Telecomunicacions de Catalunya (CTTC). His research interests include telecommunications and data engineering for networking. He received the Best Paper Award from the Network of Future Conference, in 2017.

JOSEP MANGUES-BAFALLUY received the bachelor's and Ph.D. degrees in telecommunications engineering from UPC, in 1996 and 2003, respectively. He was a Researcher and an Assistant Professor with UPC. He is currently a Senior Researcher and the Head of the Communication Networks Division, Centre Tecnològic Telecomunicacions Catalunya (CTTC), Barcelona. He has participated in various roles (including leadership) in several public funded and industrial research projects, such as 5GPPP 5Growth, 5G-Transformer, or Spanish 5G-REFINE. His research interests include NFV applied to mobile networks and autonomous network management. He was the Vice Chair of the IEEE WCNC, Barcelona, in 2018.