SAGE Science Publishing

Next-Gen AI Architectures for Telecom: Federated Learning, Graph Neural Networks, and Privacy-First Customer Automation

Rahul Khurana [1]

[1] Bothell, WA, USA

Submitted: 2022-08-29
Accepted: 2022-11-20
Published: 2022-11-23

Abstract

The telecom commerce industry generates volumes of customer interaction data across diverse channels-voice, text, and digital platforms-on an unprecedented scale. Traditional modes of processing data are insufficient to deal with the complexity and real-time needs of this high-dimensional and unstructured data. The present paper reviews advanced AI and machine learning models being applied for improving customer interactions and automation in the domain of telecom commerce. The present work is meant to discuss deep learning architecture applications, namely CNN and RNN, in processing textual and speech data with the intent of sentiment analysis and intent recognition. Reinforcement learning algorithms are adopted in optimizing customer engagement strategies, by learning policies that maximize customer satisfaction and increased revenue generation. GNNs have also been used to model complex relationships among customers for personalized recommendations and targeting marketing efforts. Actual deployment of such models requires robust system architectures, using API-driven platforms with microservices for handling scalability, modularity, and interoperability. Optimization techniques, like model quantization and pruning, leverage the computationally efficient nature for their deployment on resource-constrained platforms such as edge devices. With this respect, different techniques such as differential privacy and federated learning are discussed that can preserve the security concerns without compromising on model performance. It clearly appears that integrations of these advanced models and algorithms within API-driven systems may further improve customer interaction and automation capabilities in telecom commerce.

Keywords: AI models, customer interaction, deep learning, reinforcement learning, sentiment analysis, telecom commerce, unstructured data

Introduction

The unparalleled growth in customer interaction data within the telecom commerce sector is brought about by the rapid evolution of digital communication technologies (Bigliardi *et al.* 2012; Chin Wei *et al.* 2009). This happens to be exponential in nature and is driven by many forms of digital interactions between service providers and customers. The unique characteristics of the generated data include volume, velocity, and variety; each scenario presents its own challenges with respect to data handling, analysis, and interpretation (Amin *et al.* 2019; Johnson and Sirikit 2002).

Volume, in this context, refers to huge scales at which information is continuously generated from a myriad of sources such as call records, text messages, customer support interactions, and telemetry data streams from IoT devices. The scale of the data is huge and often runs into terabytes or even petabytes for relatively short periods of time. This is because of the continuous nature and high frequency of interactions by consumers, hence reflecting all minute usage patterns, service feedback, and engagement across various digital touchpoints. On the volume end, it requires significant infrastructure capacity to store the data in question and handle distributed datasets that might span across many servers or cloud storage units (Shafei and Tabaa 2016; Bigliardi *et al.* 2012).

The velocity characteristic refers to the speed at which data is generated and at which it flows into data systems with incredible speed. Many of these items of data actually reach the center in real time or near real-time in the field of telecom, showing how dynamic all customer interactions could be via real-time live support chats, real-time service requests, and network usage data. The high velocity of the generation brings in a lot of complexity regarding data ingestion, processing, and storage. Data needs to be processed in a very short period of time in order for it to be relevant for monitoring network performance or answering customer queries. In fact, continuous in nature, it requires systems that can handle high-throughput input streams with not so much lag.

Variety in data refers to the range of data type and format that characterizes customer interactions. Traditional structured data, which includes clearly defined formats such as databases and spreadsheets, are unlike telecom data, that includes unstructured and semi-structured data types. Unstructured data includes nat-

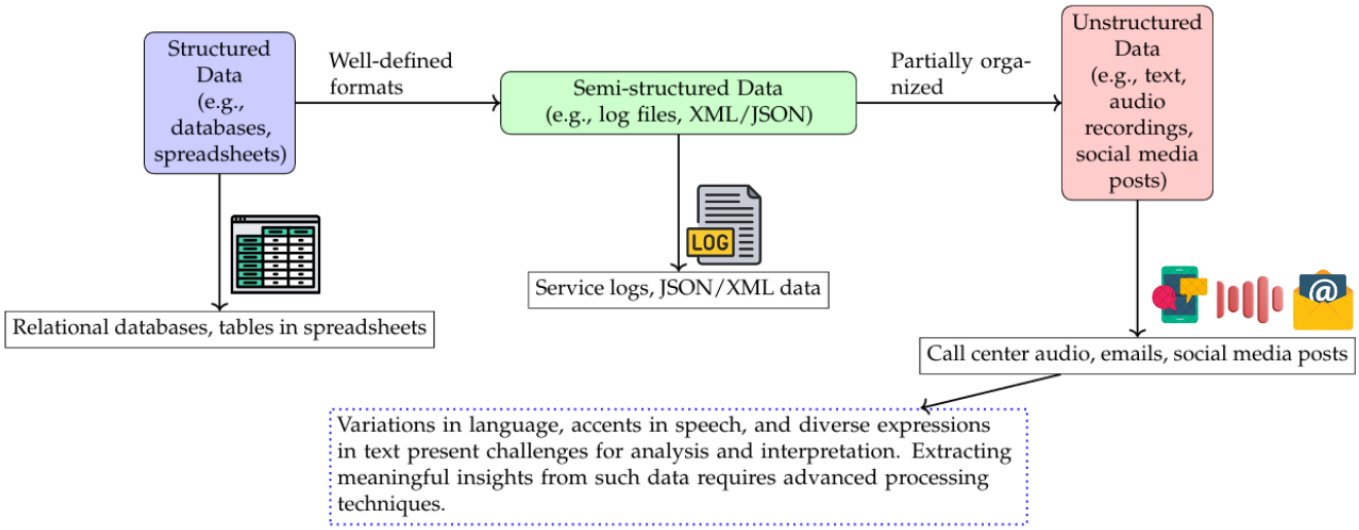

Figure 1 Variety in Telecom Data: Data Types and Formats. Structured data would be any information that may come in well-defined formats, such as relational databases or spreadsheets. Semi-structured data has a partially organized structure and would include things like service logs, XML/JSON files, and so on. Unstructured data includes everything from natural language text to audio recordings, emails, and social media posts, where the data is not preformatted.

ural language text, audio recordings from call centers, emails, and social media posts, which have no pre-defined format (**??**). Semi-structured data may involve log files or XML/JSON data from the interactions of services that are somewhat organized but do not fit neatly into relational database schemas. The unstructured data complexity involves the variation of languages, accents in speech, or different ways of expressing similar ideas in text that makes data interpretation and analysis hard when extracting meanings from raw inputs.

Moreover, the nature of unstructured data includes problems inherent in the noisiness and variability of the data. For example, speech may be interfered with by noise in the background, a quality-of-voice issue, or even dialects depending on the region; all these contribute to adding difficulties in its processing. Similarly, text data is full of colloquial, slang, and abbreviations that differ across demographics of customers, further complicating its standardization process. Converting audio to text or textual data to a common format requires thoughtful data preprocessing to tease out useful information from this variability (**?**Velmurugan 2014).

Data from customer interactions often arises from many disparate systems, ranging from CRM databases, network operation centers, and customer feedback channels to external social media platforms. Each of these sources may use different data models, metadata standards, and refresh cycles, which makes the integration and harmonization of data for any form of analysis a nightmare. How fast consistency is achieved in these diverse data sets is challenging due to conflicts in time stamps, different data structures, and variation in data accuracy and reliability coming from different systems.

Another important characteristic of customer interaction data in telecom is its contextual sensitivity. Usually, this kind of data is heavily rich in context about user behavior, perception of service quality, and usage patterns; at the same time, it may or may not clearly indicate this context. For instance, an uptick in call center interactions might point to a general problem with the service, but grasping what this is due to calls for deep analysis of conversation content and user sentiment. Similarly, a spike in the social media mentions for a service may point to a viral complaint, or it could be the start of something positive. But making sense of meaning and intent from the interactions themselves, if only for the nuance and context enwrapped in human-to-human communication, is burdensome. In such cases, this interpretation will inherently be complicated and variable.

The key challenges for the analytics method of real-time processing are immense scale and complexity concerning customer interaction data in the telecom industry. This is because of the basic nature of the data itself-scale, speed at which it is generated, and diversity in formats and structure-and also limitations of older data processing and analytical systems in managing these characteristics.

Most of the databases or data warehouses conventionally work for structured data with fixed schemas. However, the telecom industries have to handle huge volumes both in structured and unstructured data that are continuously generated. CDRs, metadata of mobile usages, and logs from network operations are high-frequency data sources that can bring traditional relational database management systems to their knees. These databases are more tuned for transactional processing and cannot scale to the high ingestion rates typical in telecom environments (Kiefer 2016). As a result, the traditional systems cannot store, retrieve, and process such large-scale datasets efficiently. The overall processing hence becomes slow and hence difficult to make timely decisions.

Real-time data processing is another challenge that is needed. Real-time data generation is a common feature in the telecom sector, as customer interactions flow into systems, together with network usage patterns and service performance metrics, in a relentless and rapid manner. The traditional batch processing systems are designed to process data in periodic intervals in large batches. They do lag in handling such streams of data continuously flowing within typical modern telecom environments. Batch processing involves time lags between data generation and analysis, which may be impossible in situations requiring

Table 1 Characteristics of Customer Interaction Data in Telecom

Characteristic	Overviewn	Challenges	Implications
Volume	Massive scale of data generation from diverse sources	Infrastructure capacity for storage	High resource allocation for data management
Velocity	Speed of data influx in real-time or near real-time	Complex data ingestion and processing	Necessity for high-throughput systems
Variety	Diversity in data types and formats	Difficulties in standardization and integration	Need for flexible processing techniques

Table 2 Comparison of Data Processing Methods in Telecom

Method	Characteristics	Strengths	Limitations
Traditional Databases	Optimized for structured data with fixed schemas	Reliable for transactional data	Struggles with high-frequency, unstructured data
Batch Processing	Processes data in large, periodic intervals	Efficient for historical data analysis	High latency; unsuitable for real-time applications
Real-Time Processing	Continuous data stream analysis	Immediate understandings for dynamic situations	Requires significant computational resources

immediacy. Examples include network outages, security threats, spikes in customer service requests, or anything else that requires immediate attention and, therefore, analysis. Traditional methods, which analyze the data at the end of a batch cycle, are not capable of such timely understandings for these dynamic situations (Hilas *et al.* 2006).

This is further complicated in real-time analytics by the kind of data formats involved in telecom. This means that the data on customer interaction involves everything from audio recordings of call center interactions to text messaging, chat transcripts, and emails-and, in some instances, social media interactions that may be in images or videos. Traditional methods of analytics are usually optimized for structured data, where each record follows a uniform format, such as entries in a table. They struggle with unstructured or semi-structured data where the format is inconsistent, such as natural language text or voice recordings (Tanwar *et al.* 2015). In such cases, meaningful understandings are usually obtained from unstructured data through complex preprocessing steps, such as speech to text or sentiment parsing in social media posts. Traditional methods do not have the flexibility for these various preprocessing needs in near real time and thus result in incomplete or delayed analyses.

The other major challenge is integration across diverse sources of information. The telecom companies require data that is generated from different systems, such as CRM systems, billing platforms, network monitoring utilities, and third-party social media platforms. Such systems often tend to operate in isolation from one another, with the data maintained in formats and structures that are incompatible with each other. Traditional techniques are inefface able to combine and align data from these diverse sources; even worse, the data is required to be analyzed in real time. Inconsistent timestamps, different data schemas, and misaligned metadata further complicate the aggregation task. Besides this, such inconsistencies cannot contribute to forming one single view of customer behavior or service performance. Due to these, understandings remain fragmented and inhibit the decision-making process by not being well-informed.

High-dimensional data, as often present in the telecom sector, is also poorly handled by traditional analytics. A good example might be customer behavior analysis, which could comprise a long feature set including call duration, frequency, geo-data, service usage patterns, device types, and even sentiment extracted from interactions. Many high-dimensional data sets need only higher-order data analytical techniques- dimensionality reduction or feature selection-to uncover relevant patterns. Traditional approaches relying on simpler statistical techniques or rule-based logic can easily become swamped by such complexity; this risks losing valuable information or perpetuates spurious correlations, with resulting loss of power in the analysis.

The computational demands for real-time analysis of large data sets are immense. Such a process requires enormous computational resources, including high memory bandwidth and processing power to deliver low latency with fast data throughput. Traditional analytic systems, which were designed under the assumption of slower, batch-oriented processing, do not have the capability to assume such high computational loads that are required by streaming analytics. This engenders bottlenecks in processing, thus slowing down the analysis and even causing delays that undermine reacting to fast-changing conditions in the telecom environment, such as traffic spikes or service degradation events.

Another needed problem is the latency that is introduced by traditional data architectures. Many traditional data architectures require the need to transfer data across different storage layers, moving from data warehouses to analytics engines for

Table 3 Sources of Customer Interaction Data in Telecom

Source	Type of Data	Data Characteristics
CRM Systems	Customer profiles and transaction history	Structured, consistent format
Network Operations	Call records and usage metrics	High-frequency, structured data
Social Media Platforms	User-generated content and feedback	Unstructured, diverse formats

example, adding extra latency in the data processing pipelines. That is, data transferred, loaded into the analytical systems, processed, and then extracted for understandings can take anything from minutes to hours. Because of this fact, it is not applicable for applications that need to provide almost instant understandings. This latency has been a problem in use cases like network optimization or real-time customer service, where the identification of the issue and subsequent response delay directly affects customer satisfaction and service reliability.

Noise and variability in unstructured data of telecoms are poorly handled by traditional analytics methods. For example, audio recordings of customer calls may include background noise, interruptions, and nonstandard speech patterns that can affect the quality of data processing. Informal language comprising spelling variations and idiomatic expressions hides true sentiment or intent in text-based data like chat logs or social media posts. Such variations may be too flexible for traditional rule-based or statistical models to accommodate and thus result in errors of interpretation or a need for extensive hand cleaning of data. This makes it harder for telecom companies to extract actionable understandings from raw interaction data and limits the depth of understanding that can be derived from customer feedback.

Deep Learning Architectures for Processing Unstructured Data

Unstructured data generated by customer interactions require deep learning models that can really understand patterns in contextual information across textual sequences Zhang *et al.* (2020); Adnan and Akbar (2019). In addition, Convolutional Neural Networks are used in the analysis of text material through convolutional filters that hunt for local features in the textual data, such as phrases and sentiment patterns. While the multi-channel CNN extends this capability to include word embeddings from multiple sources, a model can use these to represent semantic details required for subtle sentiment analysis. On the other hand, RNNs and LSTMs are designed for sequential data, hence becoming effective in modeling the temporal dynamics of customer dialogues and predicting intent over time. Bidirectional LSTMs extend this to process sequences in both directions, both forward and backward, capturing both past and future context, which becomes crucial for the correct interpretation of meaning in conversation Zhang *et al.* (2018); Al-Doulat *et al.* (2019).

That would involve elaborate data preprocessing, serving of models, and scalability to integrate into real-time API-driven systems. Preprocessing techniques, such as tokenization, stemming, and word embeddings, would prepare the raw text into forms suitable for deep learning models. Models have to be served using frameworks such as TensorFlow Serving or PyTorch Serve, which support serving at low latency with scalability Geraci *et al.* (2017); Gheisari *et al.* (2017). Such model deployments are quite effective under a microservices architecture with the use of Docker for containerization and Kubernetes for orchestration. This ensures that services can be scaled up or down with ease against variable loads of customer interaction data.

Convolutional Neural Networks (CNNs) for Text Analysis

CNNs have been adapted for text analysis by leveraging word embeddings to transform text into a spatial representation. Each word w_i is mapped to a vector $x_i = \text{Embed}(w_i) \in \mathbb{R}^d$, where d denotes the dimensionality of the embedding space. The convolution operation is defined as

$$c_i = f(w \cdot x_{i:i+k-1} + b),$$

where w represents a filter of size k, $x_{i:i+k-1}$ is a sequence of word embeddings, and f is a non-linear activation function. This operation slides the filter across the input matrix, detecting localized features that characterize the text. Pooling mechanisms, such as max-pooling or average pooling, follow the convolutional layers to retain the most significant features while reducing the dimensionality of the resulting feature maps. Pooling ensures that the CNN retains needed patterns while being computationally efficient, allowing it to generalize across varying text lengths Trask *et al.* (2019); Balducci and Marinova (2018).

In sentiment analysis, CNNs convert extracted features into a final prediction by passing them through fully connected layers. These layers integrate the localized patterns identified by convolutional filters, mapping them to specific sentiment categories. The optimization of such models is driven by the cross-entropy loss function,

$$L = -\sum_{i=1}^{N} y_i \log(\hat{y}_i),$$

where y_i represents the true label and $\hat{y}_i$ is the predicted probability. The cross-entropy loss penalizes misclassifications, guiding the model toward improved accuracy. CNNs are advantageous in this context due to their ability to detect phrase-level sentiment indicators, making them suitable for analyzing short texts, such as customer reviews or social media comments.

Recurrent Neural Networks (RNNs) for Sequential Data

RNNs have been designed to model the sequential dependencies inherent in time-series data, making them suitable for tasks like analyzing customer dialogues Bouziat *et al.* (2020); Spandorfer *et al.* (2019). A defining feature in an RNN is its hidden state, h_t, which captures information about past time steps. This update to the hidden state is given by the equation:

$$h_t = \sigma(W_{hh}h_{t-1} + W_{xh}x_t + b_h),$$

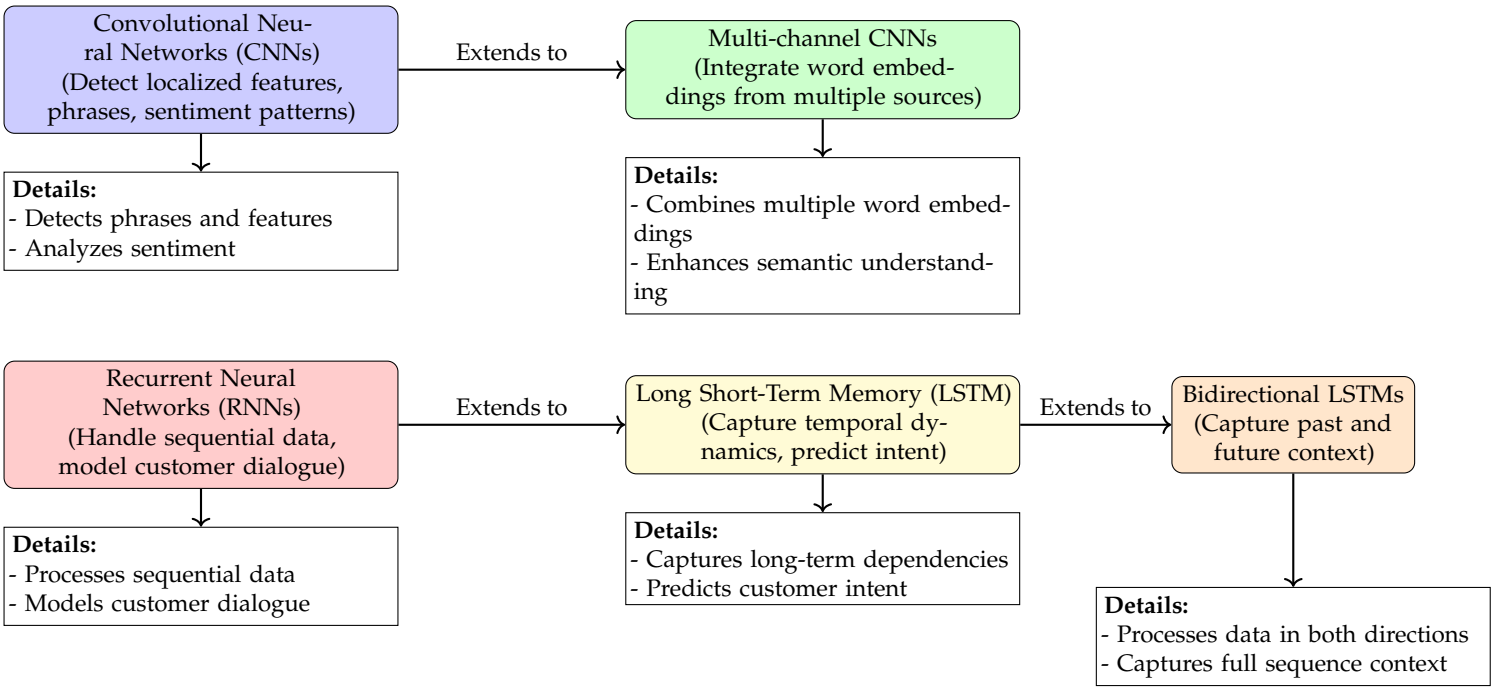

Figure 2 Deep learning architectures for processing unstructured data: CNNs and multi-channel CNNs extract unstructured data of localized features and semantic nuances from texts. RNN and LSTM networks model sequential data, capturing the temporal dynamics of customer interactions. Bidirectional LSTMs enrich this with added depth that comes from processing sequences in both directions, hence capturing the full context and allowing for more accurate interpretation of meaning.

where W_{hh} and W_{xh} are weight matrices, x_t is the input at time t, and b_h is a bias term. The activation function σ, typically a non-linearity such as tanh, enables the network to capture complex dependencies in the data. Despite their capability to learn sequential patterns, RNNs are still plagued by difficulties in handling long-term dependencies due to the vanishing gradient problem: gradients may get smaller during backpropagation and reduce the effectiveness of the model to learn from the earlier time steps Ramadhani and Goo (2017); Brunner and Stockinger (2019).

Standard RNNs have limitations mitigated by LSTM networks through the use of a gating mechanism. The LSTM architecture consists of four major components: an input gate, forget gate, output gate, and a memory cell. These four components work together in order to control the information flow regarding what information to keep and what to discard. The input gate is defined as:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i),$$

determining how much of the new input x_t should be stored in the memory cell. The forget gate,

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f),$$

modulates which parts of the previous memory should be kept. The output gate is given,

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o),$$

controls the information output from the memory cell. The cell state is updated via

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c),$$

combining the retained memory with new information. The hidden state

$$h_t = o_t \odot \tanh(c_t)$$

is then produces the hidden vector, computed based on the updated cell state. In such a setup, LSTMs are able to maintain information for a very long period of time while rejecting irrelevant information, thus being practical in the modeling of customer interactions in several turns Chen and Lin (2014); Qu (2020).

LSTM networks are useful in applications that require recognizing intent through catching the flow of time in customer dialogues. Since they can remember things from really far back, they become really good at picking up changes in customer intent over time. It will be taken into consideration that sequence-to-sequence models, usually developed with the LSTM architecture, are successful in providing responses to a client's queries. This model encodes an input sequence into a context vector, which is afterwards decoded into an output sequence; this way, they enable contextually fitting answers. In chatbots or virtual assistants, for example, it is of primary importance, since the quality of an answer depends directly on understanding and keeping the context of a conversation Nancy and Maheswari (2020); Navarro-Almanza *et al.* (2020).

Reinforcement Learning for Customer Engagement Optimization

The reinforcement learning algorithms that iteratively learn to take the best actions via interaction with their environment can considerably help in adaptive customer engagement strategies. For instance, in telecom commerce, it will be considered an environment in which customer states are derived from their interaction histories and the various range of actions correspond

to engagement strategies including personalized discounts and service modifications. These will require the concomitant reward function designs, which must be aligned with business objectives, capturing metrics including customer retention rates, conversion probabilities, and overall lifetime value. In return, RL models will aim to maximize long-term customer satisfaction and profitability via such a reward mechanism.

Reinforcement Learning

The basic framework for most reinforcement learning problems is the Markov Decision Process, which consists of a state space, action space, transition dynamics, reward function, and discount factor. In an MDP, the state space contains all the relevant information about the world, comprising customer profiles, behavioral history, and contextual data. This state space then forms the basis on which one makes decisions-once again, the action space-to send targeted offers to the users. Transition dynamics explain what changes in the environment after each action has been taken and denote the probability that, given an action selected, a new state will be reached based on the current state. The reward function quantifies how much immediate payoff an action will create in moving a person toward desirable benefits, such as increased engagement or revenue; the discount factor balances the importance between immediate versus future rewards when one is making decisions.

A policy in this framework defines the probability of choosing each action at every state and, in essence dictates the behavior. The state value function gives the expected return to be obtained by starting from a certain state and thereafter following the policy. Analogously, the action-value function returns the expected value of taking an action at a particular state and thereafter following the policy. The following enables the different assessments and refinements of the policies in reinforcement learning, hence allowing it to focus on those actions that might maximize the cumulative expected rewards over time.

Reinforcement Learning Algorithms

Q-Learning is a kind of off-policy RL, which updates the action-value function iteratively as follows:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a) \right],$$

where α is the learning rate, s' is the next state, and $\max_{a'} Q(s',a')$ represents the estimated future reward. DQN learns high-dimensional customer state spaces using neural networks that approximate the Q-function with parameters θ, represented as $Q(s,a;\theta)$. The networks are then trained to minimize the TD error between the predicted and the target Q-values, enabling the model to directly learn from raw interaction data. DQNs are suited for environments with discrete action spaces, where the number of possible engagement actions is limited.

Policy gradient methods directly optimize the policy $\pi_\theta(a \mid s)$ by maximizing the expected reward:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \right],$$

where τ represents trajectories sampled from the policy. Gradient ascent is used to update the policy parameters:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t \mid s_t) G_t \right],$$

with G_t being the return from time t. This method performs well in continuous action spaces, allowing the engagement strategy to be fine-tuned such that even minor variations in customer behavior are considered. In general, policy gradient methods are more suitable than value-based methods when the action space is large or continuous, allowing flexible adjustment of engagement tactics as required.

PPO is an advanced policy gradient method, introducing a clipped surrogate objective that enhances training stability. The deviation between the old and updated policy is bounded, preventing large updates that might destabilize training. By balancing exploration and policy refinement, PPO targets more reliable convergence, essential in dynamic environments like telecom, where customer preferences and behaviors may shift over time. This, combined with ease of implementation, makes PPO one of the most widely applied algorithms in commercial RL applications.

Implementation in Telecom Commerce

In the context of telecom, the state s_t can embed deep information on customers, ranging from basic demographic information to interaction history, current actual activities, and situational context, such as recent patterns of service usage. The action a_t consists of feasible engagement strategies ranging from sending personalized offers and service plan adjustments to retention campaigns; this is in regard to the maximization of customer satisfaction and key business metrics by proper tailoring of the service experience to the current needs of the customer.

The reward functions such as $R(s_t, a_t)$ are designed to align with specific business goals, which could be customer retention or maximization of revenue per user. Rewards can be defined in terms of changes in the customer satisfaction score, NPS, or monetary impact of certain engagement action. Whereas positive rewards could be given for actions that increase customer lifetime value, negative rewards would penalize action leading to customer churn or dissatisfaction. Reward function design should balance the short-term incentives, such as immediate revenue from upselling, with long-term objectives of building customer loyalty.

But actually deploying RL models in a telecom environment faces the dichotomy of choice between online learning, where the model keeps adapting continuously to newer data, and batch updates, where the learning happens periodically. Online learning serves better for highly dynamic environments where changes in customer behavior need to be picked up fast. This, however, requires strong mechanisms to ensure stability and avoid overfitting to recent data. Among the various exploration strategies are ϵ-greedy policies, which enable a trade-off for an RL agent between exploiting already known successful actions for engagement and exploring new strategies which may prove more effective. This is important when deploying RL in live environments since it allows, through exploration, new engagement tactics to be found that are better aligned with evolving customer preferences.

Graph Neural Networks for Modeling Customer Relationships

The key to targeted marketing strategy formulation and personalized services is basically understanding the relationships of customers (Xhonneux *et al.* 2020; Wu *et al.* 2020). Graph Neural Networks provide a very effective framework for the modelling

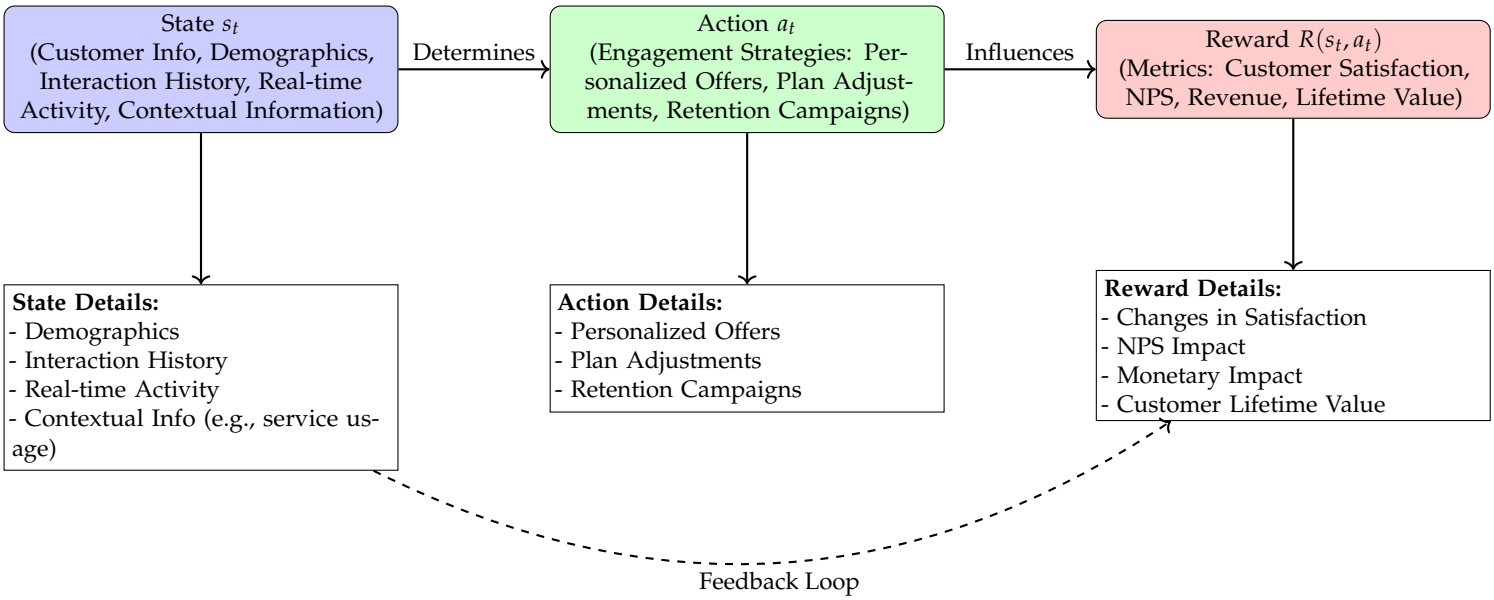

Figure 3 State-Action-Reward Representation in Telecom Commerce: The state, s_t, summarizes detailed customer information, which would inform the selection of actions a_t in terms of personalized offers or retention strategies. The reward function $R\left(s_t, a_t\right)$ implements action outcomes concerning customer satisfaction metric, NPS, and revenue impacts. It optimally considers the trade-off between short-term gains with long-term customer loyalty.

of such relational dynamics, with a representation of each customer as a node and their interactions with every other using edges in a structured graph. This can potentially be accomplished with the help of GNNs-such as Graph Convolutional Networks and Graph Attention Networks-to aggregate information from neighboring nodes, capturing both direct interactions and the more complex indirect relationships that shape customer behaviors (Fan *et al.* 2019; Franceschi *et al.* 2019).

Graph Representation of Customer Networks

In the customer graph, nodes V represent individual customers, while edges E model various forms of interactions, similarities, or social relationships. For example, an edge between any two customers can represent a similarity in their purchase patterns, frequency of communication, or interaction via social media (Gama *et al.* 2020). These graphs are more so constructed by the integration of social media data, CDRs, and transaction histories. In turn, this multi-sourced data lets the graph model the complexity of the interaction between the customers, giving a very useful view to the downstream analyses for the customer network.

Feature engineering has a needed role in enriching both nodes and edges with relevant attributes, which is needed in effective graph learning (Zhou *et al.* 2020; Xu *et al.* 2020). Node features x_v can range from demographic data such as age or location to behavioral metrics including average purchase frequency or service usage patterns. Edge features x_{uv} include metrics like frequency of communication, transaction amount by customers, and possibly co-purchase behavior. This somewhat fine-grained feature set equips the GNN model with greater power in learning from the graph (Gong and Cheng 2019), since this forms the necessary input to capture the subtleties of customer relationships.

Graph Neural Network Models

Graph Convolutional Networks generalize the convolution operation to graph data. This allows for aggregating feature information of a node's local neighborhood. The layer-wise propagation rule in GCNs is defined as:

$$H^{(k+1)} = \sigma(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(k)}W^{(k)}),$$

where $\widetilde{A} = A + I$ is the adjacency matrix A of the graph with added self-loops, $\widetilde{D}$ is the degree matrix of $\widetilde{A}$, $H^{(k)}$ is the node representation matrix on the k-th layer and $W^{(k)}$ is the learnable weight matrix. The self-loops ensure that each node can incorporate its own features during aggregation. The non-linear activation function σ, such as ReLU, introduces the non-linearity into the model so it can learn complex patterns on the customer graph. GCNs do particularly well in a task where information from a node's immediate neighborhood-like mutual connections or shared behaviors-is informative of the predictive power of the model (Hu *et al.* 2019).

GATs extend GCNs by applying attention mechanisms, allowing the model to assign different importance weights to different neighbors. This is especially useful in heterogeneous graphs, where not all connections bear equal relevance. The attention coefficients α_{ij} are computed as

$$\alpha_{ij} = \text{LeakyReLU}\left(a^T[Wh_i \| Wh_j]\right),$$

where h_i and h_j are the feature representations of nodes i and j, W is a learnable weight matrix, and a is a weight vector. The coefficients are then normalized using the softmax function over all neighbors $N(i)$ of node i:

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(a^T[Wh_i \| Wh_j]\right)\right)}{\sum_{k \in N(i)} \exp\left(\text{LeakyReLU}\left(a^T[Wh_i \| Wh_k]\right)\right)}.$$

Then, the new representation of node i is computed as:

$$h_i' = \sigma\left(\sum_{j \in N(i)} \alpha_{ij} W h_j\right).$$

This mechanism allows GATs to focus on the most informative neighbors while aggregating information, capturing intricate patterns in relationships that may not be captured through uniform aggregation. This property makes GATs suitable for applications where not every relationship of customers is indicative of the pattern of behavior, such as identifying key influencers in a social network (Liu *et al.* 2019, 2020).

Applications in Telecom Commerce

GNNs can be used in telecom to develop personalized recommendation systems by generating rich node embeddings. The embedding will encode each customer in a low-dimensional vector representative of their interaction history and similarities to other customers, hence suggesting relevant services or offers. For instance, the embedding for a customer may reveal that this customer has recently shown a similarity to a segment of other customers who have upgraded their service plan and could therefore be targeted with upselling.

Another important use case of GNNs is community detection and customer segmentation (Qu *et al.* 2020; Scarselli *et al.* 2008). Cluster the learned representations of GNNs to find communities in the customer base with similar interaction patterns or preference. These clusters can then be used for targeted marketing campaigns where each community receives personalized communication based on the shared characteristics of the cluster members. Thus, telcos are in a position to come up with much more focused marketing strategies aligned with the needs and behaviors of distinct customer groups.

GNNs perform well in the area of predictive maintenance and churn prediction. These models can find customers displaying patterns similar to those that have already churned by analyzing the structure of customer interactions within the graph. Therefore, anomaly detection techniques applied to the learned node embeddings can flag customers at risk of leaving the service. In turn, proactive measures of engagement become possible through personalized communications, for example, offering retention incentives to high-risk customers. Moreover, analysis of network structures can indicate bottlenecks or failures in the services provided. This ensures improvements in network reliability and customer satisfaction.

System Architectures for AI Integration

Advanced AI models require system architectures that can allow modularity, scalability, and good communication between components for their deployments in a production environment. Microservices architectures go a long way in helping achieve these by decomposing applications into loosely coupled, independently deployable services. Each service performs a distinct function-data ingestion, model inference, or results dissemination-and communicates with other services through well-defined APIs. This methodology allows teams to create, test, and scale discrete components independently of the larger system, enabling more agile development cycles and smooth updates.

Loose coupling and high cohesion are the basis of microservices architectures. Loose coupling implies that the dependencies among the services are minimized; hence, the update or replacement of services does not cascade changes to other parts of the system. High cohesion within the services themselves means each service has a clear and focused responsibility; thus, maintenance and debugging become easier. This separation of concerns enables components to be developed and scaled independently, hence optimizing resource utilization and enhancing system resilience.

Communication among microservices can be enabled through synchronous and asynchronous protocols, depending on the requirements for latency and fault tolerance. Synchronous protocols, such as REST and gRPC, serve well for real-time interactions where low-latency communication is required. REST brings ease of use and integration with HTTP-based APIs, whereas gRPC boasts the use of protocol buffers, hence higher performance, and provides features such as streaming and bidirectional communication. For services needing to decouple in time, event-driven protocols like AMQP-Advanced Message Queuing Protocol-and Kafka will be used. These protocols also allow services to communicate without having to wait for immediate responses, thus helping in improving scalability and fault tolerance in scenarios such as event-driven data processing or the execution of background tasks.

Containerization: This is mainly using Docker as a standard way of deploying AI models and their dependencies. Models can be encapsulated in containers so that they are always running coherently, from local testing to cloud-based production. The Docker images package the AI model with its runtime, libraries, and configurations in one deployment package that makes it behave precisely the same no matter where it is deployed. This means it reduces testing and debugging because the exact container can be run against different environments without changes.

Orchestration tools like Kubernetes manage the deployments and scaling of such containers. Kubernetes automates the deployment, scaling, and operations of containerized applications. It supplies a lot of features around load balancing, service discovery, and the process of rollout and rollback. It is able to scale services horizontally through the automated increase or decrease in the number of container instances according to the workload, hence assuring that the computational resources are being used efficiently. Kubernetes is particularly fit for deploying these AI services with variable workloads, such as user queries that fluctuate or sudden spikes in data ingested, because it handles complex clusters of containers.

Specialized model-serving frameworks like TensorFlow Serving, TorchServe, and NVIDIA Triton fundamentally make the deployment of AI models smooth, thus performing real-time inferences with efficiency. TensorFlow Serving and TorchServe allow deployment of models trained in TensorFlow and PyTorch, respectively, by exposing them via RESTful and gRPC interfaces for performing inferences. These systems are designed for high-throughput environments with features such as batch processing, model versioning, and automatic resource allocation. NVIDIA Triton is optimized for deploying models on GPUs, supporting a wide variety of frameworks, and allowing multi-model deployment on one single GPU, thus leveraging hardware to the maximum and reducing inference latency.

Real-time processing of data is very important for AI models that require incessant input from streaming data sources. Apache Kafka finds extensive usage for real-time data streaming due to its high throughput and fault tolerance. Kafka provides the capability of collecting data streams and distributing them,

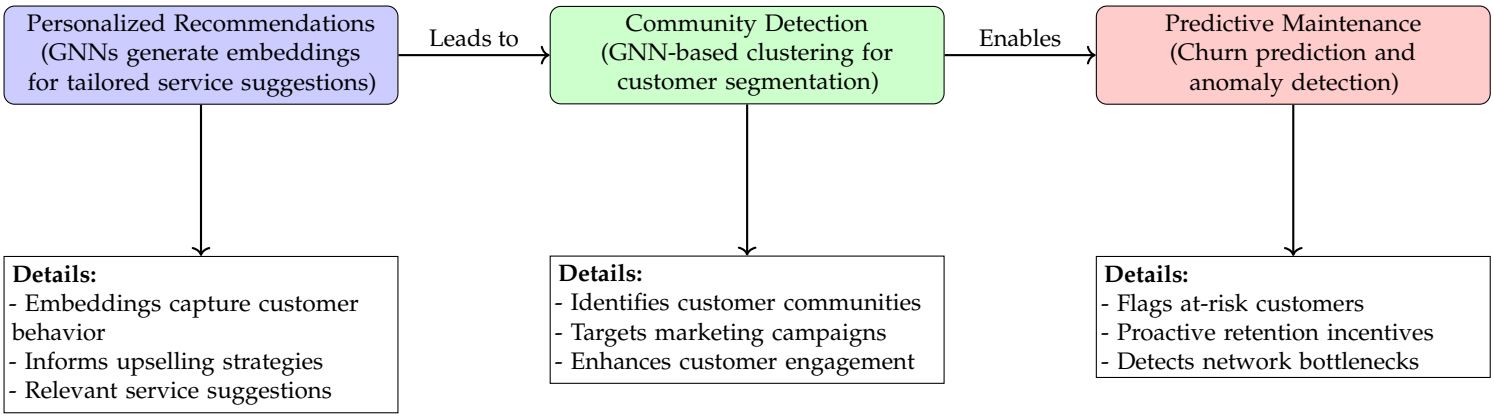

Figure 4 Applications of GNNs in Telecom Commerce: GNNs support personalized recommendation systems by inferring customer embeddings that convey knowledge to be used for service suggestion. Community detection and segmentation with GNNs spot clusters of customers who behave similarly and enhance targeting marketing efforts. Predictive maintenance and churn prediction leveraging GNNs find at-risk customer patterns that permit proactive retention and enhancement of network reliability.

Table 4 Comparison of Communication Protocols in Microservices Architecture

Protocol	**Type**	**Use Case**	**Latency**
REST	Synchronous	Real-time interactions	Moderate
gRPC	Synchronous	High-performance, streaming	Low
AMQP	Asynchronous	Event-driven data processing	High
Kafka	Asynchronous	Background task execution	High

Table 5 Model Deployment Strategies and Tools

Strategy	**Tool**	**Key Feature**	**Use Case**
Containerization	Docker	Environment consistency	Local to cloud deployment
Orchestration	Kubernetes	Auto-scaling	Large-scale deployment
Model Serving	TensorFlow Serving	Batch processing	Real-time inference
GPU Acceleration	TensorRT	Latency reduction	High-traffic inference

so the AI models receive the latest information upon which to base inference. For more advanced stream processing tasks, such as filtering, aggregation, and windowed operations, Kafka can be combined with other frameworks like Apache Flink. Flink provides very low latency while processing data with stateful computation. It's a perfect framework for applications like anomaly detection systems or real-time recommendations in customer engagement systems.

These AI applications generate various data formats that definitely need efficient data storage solutions. NoSQL databases, such as MongoDB and Cassandra, have been utilized to store unstructured data in the form of logs, customer feedback, and transaction records. These support horizontal scalability and distributed data architecture, hence proving to be scalable for large volumes of data. Graph databases like Neo4j are intended for use when data is in a relational format or the entities involved are interrelated. These databases are optimized for complex relationships between stored data and querying features, including advanced analytics such as social network analysis or customer segmentation based on interaction patterns.

A feature store is a repository that stores and manages features both during training and at inference. This ensures consistency between training and serving features by providing a single source of truth for models to make sure they use the same data representation during both training and real-time inference. Feature stores also make feature reuse easy across different models by centralizing feature storage, thus reducing the redundancy of feature engineering efforts. This becomes important for complex AI systems where feature transformation needs to remain consistent and access to the features needs to be of low latency for model performance. Given the complexity and hence computational demands of deep learning models, hardware acceleration is often employed in order to make the inference speeds acceptable. The typical go-to solution for accelerating the inherent parallel computations in neural networks is GPUs; these make the training and inference times considerably

Table 6 Data Storage Solutions for AI Pipelines

Storage Type	Database	Data Format	Scalability
NoSQL	MongoDB	Unstructured	High
NoSQL	Cassandra	Wide-column	High
Graph Database	Neo4j	Relational Graph	Moderate
Relational Database	PostgreSQL	Structured	Moderate

lesser. One can go even further in efficiency with specialized AI accelerators such as Google's TPU or NVIDIA's TensorRT. The accelerators are highly optimized for deep learning workloads, ranging from matrix operations to reduction of latency during inference. Deployment of the models in such hardware is of prime importance in high-traffic environments where response time is crucial for the user experience.

Edge computing is a strategic deployment choice for scenarios where models need to be run close to the origin of data capture to preserve latency and bandwidth consumption. In this paradigm, AI models are deployed on edge devices like routers or IoT gateways, offering real-time inference without having to send data to a centralized cloud. This is useful in telecom applications where devices generate large volumes of data, sending them over the network, thus being expensive. By processing data locally, edge computing reduces latency, enhances data privacy, and it also enables making decisions in real time, for instance, adjusting the service quality concerning local network conditions.

It demands an integral architecture of integration that effectively finds a balance between modularity and scalability with computational efficiency. A microservices architecture lets AI components be deployed independently as well as scale independently of the others, while model-serving frameworks and orchestration tools streamline how such services are managed. Real-time data pipelines and efficient data storage ensure the most current information for the models to operate on, while hardware acceleration with edge computing provides the performance required for responsive user experiences. These architectures are the essential foundation to deploy advanced AI models that can adapt to dynamic workloads and deliver value in real-time applications.

Optimization Techniques for Computational Efficiency

Typically, resource constraint in deployment requires several optimization techniques in order to reduce the model size and improve the speed of inference. Such techniques are particularly useful in situations when there are hardware limitations, memory capacity, and latency requirements, such as in edge computing or mobile devices. Strategies related to quantization, pruning, and knowledge distillation remain some of the most common approaches. Each one of these techniques has at its core the objective of reducing the computational demand in models with minimum loss regarding predictive performance.

Model Quantization

Quantization is any technique that reduces the precision of a model's weights and activations by changing them from high-precision, floating-point representations-usually 32-bit-to lower-bit formats such as 8-bit integers (Chen *et al.* 2017; Esser *et al.* 2019). This reduction significantly decreases both the memory footprint of the model and the computation, especially for hardware optimized for lower precision arithmetic. There are two major approaches to quantization:

By contrast, post-training quantization takes a pre-trained model and applies quantization to it. That is, during training the weight updates are done in full precision, but right afterwards, their precision is decreased by the quantization process. While easy to apply in practice, this simple technique suffers from accuracy drops on the models that are sensitive to the reduction of weight precision (Jacob *et al.* 2018; Kim and Rush 2016).

QAT trains the model with quantization effects by emulating lower precision arithmetic during forward passes. As such, the model learns from the lower precision it gets exposed to during training. This generally leads to a more accurate quantized model compared to others. Although QAT is more computationally expensive during training, the results are far better compared to post-training full integer quantization in terms of accuracy.

Quantization methods achieve good trade-offs between model size and accuracy. While decreasing the precision significantly reduces the memory used by the models for storage, and also accelerates the inference, it might introduce quantization error, which finally leads to decreased model accuracy. On the other hand, with techniques such as QAT, one could train models where this error is mitigated to keep the performance degradation very minimal. That makes quantization useful for deployment on devices when memory is limited or energy efficiency is critical, such as in mobile applications or IoT devices.

Model Pruning

Model pruning has been centered around the removal of superfluous parameters, which in return reduces model complexity and improves computational efficiency. This technique is achieved by removing those weights or neurons that contribute little to the output of a model, thus allowing the network architecture to be leaner. Pruning can be performed at two levels: unstructured and structured pruning.

Unstructured pruning: This technique removes single weights in the model, selected according to their relative importance. Examples include removing weights whose magnitudes fall below a certain threshold. While unstructured pruning can achieve very high sparsities, it typically results in irregular patterns of memory access that are less efficient on standard hardware.

The structured pruning targets the model components such as neurons, channels, and convolution filters. Structured pruning is more amenable to optimization on hardware platforms through the removal of entire filters or layers since the complexity of the matrix multiplications involved in the forward passes reduces. It is, therefore, practical for real-world appli-

Table 7 Optimization Techniques for Deep Learning Models

Technique	Description	Use Cases
Quantization	Reduces weight precision (e.g., to 8-bit)	Mobile, edge devices
Pruning	Removes redundant parameters to improve efficiency	Hardware-limited inference
Distillation	Transfers knowledge from a larger to a smaller model	On-device, IoT inference

cations where both speedups and compatibility with hardware accelerators are desired.

Multiple algorithms guide the pruning process, considering different criteria on which to base their decision about which parameters to remove. There are a few types of magnitude-based pruning, but most are based on ranking weights or filters based on their absolute values, after which those with the smallest magnitudes are pruned. These methods assume that relatively small absolute-value parameters contribute little to the overall performance of the model. This is then followed by a retraining phase in which model accuracy can be recovered through the re-adjustment of the remaining weights.

Iterative pruning and retraining cycles gradually remove a fraction of the parameters during several cycles, while retraining is conducted between every pruning step. It allows the model to adapt gradually to the loss of the parameters and hence often retains better performance compared to aggressive single-step pruning.

Knowledge Distillation

Knowledge distillation is an optimization technique whereby knowledge from a huge and complex model, which is the teacher, is transferred into a much simpler small model, which is the student. Herein, learning by the meta-model student is to approximate the distribution of outputs coming from the teacher model by capturing its generalizations (Li and Wang 2019; Polino *et al.* 2018; Zhu *et al.* 2016). This process generally involves training the student model by minimizing two incorporated loss functions:

$$L = \alpha L_{KD} + (1 - \alpha) L_{CE},$$

where L_{KD} is the knowledge distillation loss, and L_{CE} is the cross-entropy loss w.r.t. the true labels. The knowledge distillation loss L_{KD} is computed based on the Kullback-Leibler divergence between the softened outputs of the teacher and student models:

$$L_{KD} = KL(\sigma(z_T/T), \sigma(z_S/T)),$$

where z_T and z_S are the logits from the teacher model and the student model respectively and T is a temperature parameter which smooths the output distributions. A higher temperature allows the student to learn from the relative probabilities of the incorrect classes-to capture richer information than simply matching the hard labels.

Knowledge distillation enables the use of smaller models that retain much performance of larger models; therefore, it could be useful in environments where only limited computational resources are available. These combine the interpretability of the teacher model's decision boundary with efficiency for the student, and this allows for such things as on-device inference where both memory and processing power are limited.

Privacy-Preserving Methods in AI Systems

Dealing with sensitive customer information within AI systems implies strict adherence to privacy regimes and robust measures of security. The aim would be toward the protection of customer information in a data processing and model training perspective that is law-compliant, such as GDPR and CCPA. The main techniques proposed in this direction include differential privacy and federated learning for privacy preservation in machine learning.

Differential Privacy is a technique protecting individual data points by adding random noise into the dataset or to the results of queries performed on the data, in such a way that any result of any analysis does not tell whether a record contributed to a dataset. The privacy-utility trade-off in differential privacy is often characterized by a parameter commonly called the privacy budget, denoted as, which denotes the strength of the privacy guarantee; smaller means more private since more noise is added, it often results in lower accuracy for data analysis. On the other hand, a larger provides more accurate analysis with less privacy protection. One needs to handle the privacy budget: that basically amounts to adding a certain amount of noise so that there bodes a balance between privacy and utility of data processed. Differential privacy, for example, when applied to the training of machine learning models, allows for the addition of noise to the gradient updates during their training in such a way as to make individual data points not disproportionately influential with respect to the model parameters.

Federated Learning is another technique that enhances privacy by enabling models to be trained across multiple decentralized devices or servers without actually transferring the data from those devices. Instead of having all the data from a central location, each one computes locally the updates with its data, and the updates sent-like gradients or model parameters-to a central server for aggregation. With this, the chances of data breaches become very minimal since the raw data stays within the local devices. This adherence to data locality by not having to move data across borders or regions minimizes regulatory risks. Federated learning is applicable in industries such as telecommunication and healthcare, which contain sensitive data that should remain within jurisdictions.

However, federated learning introduces such challenges as communication overhead because, in every round of training, the model updates need to be exchanged between the centralized server and the decentralized devices. This may easily become a bottleneck when the number participating is huge or the network condition is poor. Besides, heterogeneous data distributions across devices make model convergence more complicated. This

Table 8 Quantization Methods and Their Impacts

Method	Description	Accuracy
Post-training	Quantizes after training	May reduce accuracy
Quantization-aware	Simulates quantization during training	Retains higher accuracy

Table 9 Pruning Strategies for Model Optimization

Strategy	Type	Advantages
Unstructured Pruning	Removes individual weights based on importance (e.g., magnitudes).	Achieves high sparsity but may lead to irregular memory access.
Structured Pruning	Targets entire structures (neurons, channels, filters).	More efficient on hardware, reduces matrix multiplication complexity.

may lead to difficulties in generalizing the aggregated model across all sources of data, since each device may be representative of a particular set of local data that is not representative of the overall distribution. These issues are sometimes mitigated using techniques such as learning rate adjustment and personalized models.

Secure aggregation protocols are additionally employed in an attempt to enhance privacy during federated learning. These protocols will have the central server aggregate model updates from various devices in such a way that it will not be able to access individual updates, hence preventing sensitive data from being reconstructed from each of the updates submitted by various devices. Secure aggregation relies on cryptographic techniques like homomorphic encryption or secret sharing, where updates become encrypted before being sent to the server, while enabling the server to compute the sum of the updates without decrypting them. Here, even a compromised server cannot access data from individual devices, and so a higher standard of privacy is guaranteed.

Conclusion

The rapid growth in communication technologies has led to an exponential explosion of customer interaction data for the telecom commerce industry. The diversity and unstructured nature of this data create a lot of problems regarding real-time analytics and decision-making. Traditional methods for the processing of data are mostly ineffective in extracting actionable understandings from such high-dimensional datasets. Such solutions involve AI and machine learning models that, until now, have provided advanced tools for data interpretation and automation, improvement of customer service, and efficiency of operations.

Working with unstructured data requires models which will extract complex patterns and contextual details from such datasets. CNN works effectively in the analysis of textual data by making use of convolutional layers to detect hierarchical features. That makes them helpful for sentiment analysis as they identify minor expressions and emotions in customer messages. RNNs, and LSTMs in particular, are useful for treating sequential data, like customer dialogues, and predict future interactions. This integration with API-driven systems for model deployment and scaling enhances responsiveness in customer support services.

Adaptive customer behavior requires adaptive strategies of engagement. Reinforcement Learning algorithms optimize an action through interaction with the environment. By modeling customer interactions as a Markov decision process, the RL agents can optimize policies that improve the metrics of customer retention and revenue. RL in telecom commerce could be done by designing reward functions reflecting business objectives and integrating RL agents into existing systems via APIs for real-time decision support.

It will be important to understand the relationships and interactions between customers for personalized marketing and service recommendations. GNNs can model these by representing customers and their interactions in graph form, where nodes and edges represent customers and their interactions, respectively. GNNs make use of message-passing algorithms to aggregate information from other connected nodes, catering to both local and global patterns. The deployment of GNN involves handling graph-structured data and integrating with API-driven platforms for real-time understandings.

System architectures that allow scalability and performance make for efficient deployment of AI models. Microservices enable component development in a modular fashion, allowing for independent scaling, thus easing the integration of complex AI models more easily. Techniques like quantization reduce the size and computational load of neural networks, hence making them fit better for resource-constrained devices. Pruning further enhances efficiency by removing redundant connections within neural networks without any major loss in accuracy-a common concern when trying to maintain latency low for customer-facing applications.

It requires treating customer data in a privacy and security-respectful way. This mostly means differential privacy techniques that add noise either to the datasets or to the model outputs while protecting the utility of the individual data points at a higher level. Federated learning allows training a model across decentralized devices in a manner that raw data remains on local devices, while model updates are aggregated. These techniques shall be implemented by carefully balancing privacy with performance using appropriate design choices considering

secure communication protocols and compliance with personal data protection regulations.

Therefore, several challenges are faced while integrating advanced AI models into commerce systems in telecommunication systems, both from a technical and organizational point of view. Besides data heterogeneity, the major factor that presently affects the performance of AI models is that their source arises from many diversified platforms, including call records, customer interaction logs, and network usage logs-all potentially with different formats, structures, and quality of data. This heterogeneity negatively impacts model performance as it introduces inconsistencies during training or causes inaccuracies during inference. These challenges put forth strong demands for establishing good data governance frameworks that ensure quality, consistency, and integrity of data. Such frameworks also ensure compliance with data privacy regulations-including those outlined by GDPR and CCPA-that are a must when handling sensitive customer data.

Real-time decision making for telecom applications, such as customer engagement or network optimization, requires processing and analyzing data in a short latency period. These models are envisioned to meet such real-time processing needs by being integrated into some form of efficient data pipelines that can handle high-throughput streaming data. This includes the use of stream platforms like Apache Kafka for streaming data and Apache Flink for complex event processing; such will enable the real-time ingestion and transformation of data. Deployment of AI models for real-time inference also requires frameworks like TensorFlow Serving or NVIDIA Triton, optimized for low latency response times. These frameworks can also facilitate rapid deployment to a production environment so that model predictions can be made quickly and acted upon.

The integration of AI models into the conventional infrastructure of telecom is even more daunting due to the required interoperability between the legacy systems and the new AI components. Legacy systems tend to operate on older protocols and data formats, most of which will not interface directly with newer AI technologies. The collaboration among these systems should be possible with the use of standard communication protocols, like RESTful APIs, and data exchange formats such as JSON, which would make structured data transfer between systems possible. RESTful APIs support a flexible way of embedding the output of AI models into several already functioning applications that either provide automatic responses to customer queries or adjust prices dynamically depending on model predictions.

Model explainability goes beyond technical integration; it is a must-have requirement in a quest to build trust among internal teams and regulators. Many deep learning models are complex to explain, sometimes even impossible to interpret, raising concerns about transparency and accountability. This is where techniques like SHAP values and LIME are useful for providing insight into how models make certain decisions. SHAP values provide the quantity of each feature's contribution to a certain prediction, enabling globally consistent explanations of feature importance. LIME describes complex models in terms of locally interpretable models around each prediction. These various techniques make the models more interpretable at significant computational costs, since extra computation is required for the interpretability calculation, especially on big datasets or when using complex models. The art of balance between being transparent and the computational costs of interpretability lie in between. In essence, all these factors play an important role in making the AI model efficient at the same time complying with various regulatory requirements.

References

Adnan K, Akbar R. 2019. An analytical study of information extraction from unstructured and multidimensional big data. Journal of Big Data. 6:1–38.

Al-Doulat A, Obaidat I, Lee M. 2019. Unstructured medical text classification using linguistic analysis: A supervised deep learning approach. In: . pp. 1–7. IEEE.

Amin A, Al-Obeidat F, Shah B, Adnan A, Loo J, Anwar S. 2019. Customer churn prediction in telecommunication industry using data certainty. Journal of Business Research. 94:290–301.

Balducci B, Marinova D. 2018. Unstructured data in marketing. Journal of the Academy of Marketing Science. 46:557–590.

Bigliardi B, Ivo Dormio A, Galati F. 2012. The adoption of open innovation within the telecommunication industry. European Journal of Innovation Management. 15:27–54.

Bouziat A, Desroziers S, Feraille M, Lecomte J, Divies R, Cokelaer F. 2020. Deep learning applications to unstructured geological data: From rock images characterization to scientific literature mining. In: . volume 2020. pp. 1–5. European Association of Geoscientists & Engineers.

Brunner U, Stockinger K. 2019. Entity matching on unstructured data: an active learning approach. In: . pp. 97–102. IEEE.

Chen G, Choi W, Yu X, Han T, Chandraker M. 2017. Learning efficient object detection models with knowledge distillation. Advances in neural information processing systems. 30.

Chen XW, Lin X. 2014. Big data deep learning: challenges and perspectives. IEEE access. 2:514–525.

Chin Wei C, Siong Choy C, Kuan Yew W. 2009. Is the malaysian telecommunication industry ready for knowledge management implementation? Journal of knowledge management. 13:69–87.

Esser SK, McKinstry JL, Bablani D, Appuswamy R, Modha DS. 2019. Learned step size quantization. arXiv preprint arXiv:1902.08153. .

Fan W, Ma Y, Li Q, He Y, Zhao E, Tang J, Yin D. 2019. Graph neural networks for social recommendation. In: . pp. 417–426.

Franceschi L, Niepert M, Pontil M, He X. 2019. Learning discrete structures for graph neural networks. In: . pp. 1972–1982. PMLR.

Gama F, Bruna J, Ribeiro A. 2020. Stability properties of graph neural networks. IEEE Transactions on Signal Processing. 68:5680–5695.

Geraci J, Wilansky P, de Luca V, Roy A, Kennedy JL, Strauss J. 2017. Applying deep neural networks to unstructured text notes in electronic medical records for phenotyping youth depression. BMJ Ment Health. 20:83–87.

Gheisari M, Wang G, Bhuiyan MZA. 2017. A survey on deep learning in big data. In: . volume 2. pp. 173–180. IEEE.

Gong L, Cheng Q. 2019. Exploiting edge features for graph neural networks. In: . pp. 9211–9219.

Hilas CS, Goudos SK, Sahalos JN. 2006. Seasonal decomposition and forecasting of telecommunication data: A comparative case study. Technological Forecasting and Social Change. 73:495–509.

Hu W, Liu B, Gomes J, Zitnik M, Liang P, Pande V, Leskovec J. 2019. Strategies for pre-training graph neural networks. arXiv preprint arXiv:1905.12265. .

Jacob B, Kligys S, Chen B, Zhu M, Tang M, Howard A, Adam H, Kalenichenko D. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In: . pp. 2704–2713.

Johnson WC, Sirikit A. 2002. Service quality in the thai telecommunication industry: a tool for achieving a sustainable competitive advantage. Management Decision. 40:693–701.

Kiefer C. 2016. Assessing the quality of unstructured data: An initial overview. In: . pp. 62–73.

Kim Y, Rush AM. 2016. Sequence-level knowledge distillation. arXiv preprint arXiv:1606.07947. .

Li D, Wang J. 2019. Fedmd: Heterogenous federated learning via model distillation. arXiv preprint arXiv:1910.03581. .

Liu M, Gao H, Ji S. 2020. Towards deeper graph neural networks. In: . pp. 338–348.

Liu Q, Nickel M, Kiela D. 2019. Hyperbolic graph neural networks. Advances in neural information processing systems. 32.

Nancy AM, Maheswari R. 2020. A review on unstructured data in medical data. J. Crit. Rev. 7:2202–2208.

Navarro-Almanza R, Juárez-Ramírez R, Licea G, Castro JR. 2020. Automated ontology extraction from unstructured texts using deep learning. Intuitionistic and Type-2 fuzzy logic enhancements in neural and optimization algorithms: Theory and applications. pp. 727–755.

Polino A, Pascanu R, Alistarh D. 2018. Model compression via distillation and quantization. arXiv preprint arXiv:1802.05668. .

Qu S. 2020. Combining structured and unstructured data in electronic health record for readmission prediction via deep learning. The Ohio State University. .

Qu X, Li Z, Wang J, Zhang Z, Zou P, Jiang J, Huang J, Xiao R, Zhang J, Gao J. 2020. Category-aware graph neural networks for improving e-commerce review helpfulness prediction. In: . pp. 2693–2700.

Ramadhani AM, Goo HS. 2017. Twitter sentiment analysis using deep learning methods. In: . pp. 1–4. IEEE.

Scarselli F, Gori M, Tsoi AC, Hagenbuchner M, Monfardini G. 2008. Computational capabilities of graph neural networks. IEEE Transactions on Neural Networks. 20:81–102.

Shafei I, Tabaa H. 2016. Factors affecting customer loyalty for mobile telecommunication industry. EuroMed Journal of Business. 11:347–361.

Spandorfer A, Branch C, Sharma P, Sahbaee P, Schoepf UJ, Ravenel JG, Nance JW. 2019. Deep learning to convert unstructured ct pulmonary angiography reports into structured reports. European radiology experimental. 3:1–8.

Tanwar M, Duggal R, Khatri SK. 2015. Unravelling unstructured data: A wealth of information in big data. In: . pp. 1–6. IEEE.

Trask N, Patel RG, Gross BJ, Atzberger PJ. 2019. Gmls-nets: A framework for learning from unstructured data. arXiv preprint arXiv:1909.05371. .

Velmurugan T. 2014. Performance based analysis between k-means and fuzzy c-means clustering algorithms for connection oriented telecommunication data. Applied Soft Computing. 19:134–146.

Wu Z, Pan S, Chen F, Long G, Zhang C, Philip SY. 2020. A comprehensive survey on graph neural networks. IEEE transactions on neural networks and learning systems. 32:4–24.

Xhonneux LP, Qu M, Tang J. 2020. Continuous graph neural networks. In: . pp. 10432–10441. PMLR.

Xu K, Zhang M, Li J, Du SS, Kawarabayashi Ki, Jegelka S. 2020. How neural networks extrapolate: From feedforward to graph neural networks. arXiv preprint arXiv:2009.11848. .

Zhang D, Yin C, Zeng J, Yuan X, Zhang P. 2020. Combining structured and unstructured data for predictive models: a deep learning approach. BMC medical informatics and decision making. 20:1–11.

Zhang Q, Yang LT, Chen Z, Li P. 2018. A survey on deep learning for big data. Information Fusion. 42:146–157.

Zhou J, Cui G, Hu S, Zhang Z, Yang C, Liu Z, Wang L, Li C, Sun M. 2020. Graph neural networks: A review of methods and applications. AI open. 1:57–81.

Zhu C, Han S, Mao H, Dally WJ. 2016. Trained ternary quantization. arXiv preprint arXiv:1612.01064. .