Edge LLM Hosting

A technical paper prepared for presentation at SCTE TechExpo25

Randy Levensalor
Distinguished Technologist
CableLabs
r.levensalor@cablelabs.com

Megan Collins
Lead Engineer
CableLabs
m.collins@cablelabs.com

Ryan Fuerst, CableLabs

Table of Contents

Title **Page Number**

List of Figures

List of Tables

1. Executive Summary

The rapid growth of AI applications demanding low latency, data privacy, and real-time responsiveness is driving a shift from centralized cloud computing toward edge-hosted AI. This transformation mirrors earlier trends in content delivery networks and is particularly relevant for telecom and cable operators, whose distributed infrastructure is well-positioned to support this evolution.

This paper explores the deployment of large language models (LLMs), vision-language models (VLMs), and other AI workloads at the network edge. By placing compute closer to users and data sources, operators can reduce latency, preserve data locality, meet regulatory requirements, and unlock new operational efficiencies and revenue streams.

This paper outlines a strategic approach for deploying AI models at different layers of the network—from customer premises to metro hubs and regional data centers. It highlights how matching AI model size with infrastructure capabilities can optimize performance, reduce latency, and improve service reliability. By processing data locally, operators can minimize bandwidth usage and enhance responsiveness. Techniques like Retrieval-Augmented Generation (RAG), when combined with edge-hosted embedding models, can further improve relevance and reduce the need for frequent retraining.

The paper also evaluates IaaS, PaaS, and SaaS models for edge AI, each offering varying degrees of control, complexity, and monetization potential. Operators can support third-party innovation by offering flexible hosting platforms, while simultaneously using edge AI internally for tasks such as network optimization, anomaly detection, and predictive maintenance. Reinforcement learning and fine-tuning at the edge enable continuous model adaptation and alignment with local conditions.

Future directions include multi-model hosting, large-model sharding across distributed systems, and federated learning across nodes. Integration with standards like ETSI MEC and CAMARA's edge APIs will help align edge AI capabilities with operator-grade requirements for orchestration and interoperability.

Ultimately, edge-hosted AI represents a strategic opportunity for operators to deliver differentiated, low-latency AI services while maintaining data control and reducing cloud dependency. This paper offers technical leaders a blueprint for deploying scalable, resilient AI at the edge and adapting to a rapidly evolving AI landscape.

2. Introduction

This white paper explores the deployment of edge-hosted AI within service provider networks, with a focus on large language models (LLMs), vision-language models (VLMs), and other AI systems. It delivers a practical and strategic framework for cable operators and telecom providers seeking to integrate AI at the edge, where compute resources are closer to users and data sources. Through technical insights and operational analysis, the paper highlights how edge AI can improve latency, enhance privacy, optimize infrastructure, and unlock new services. It is designed to equip technical leaders and strategists with the knowledge needed to evaluate edge AI opportunities, navigate implementation challenges, and drive innovation across both network operations and customer-facing applications.

2.1. The Rise of Edge-Hosted AI in Service Provider Networks

Today's customers demand instant AI responses, whether asking a voice assistant for directions, trying on virtual clothing, or receiving personalized recommendations. But when AI inference happens in distant cloud data centers, network latency impacts these experiences with delays exceeding 100 milliseconds. This problem intensifies as data gravity[41] takes hold: the massive volumes of sensor data, video streams, and user interactions generated at the edge become too expensive and slow to shuttle to the cloud for processing. The solution mirrors how content delivery networks solved video streaming by distributing infrastructure closer to users. Edge AI follows this same principle, pushing intelligence to where data originates and customers interact. By processing data locally rather than fighting physics and bandwidth constraints, edge computing finally enables the responsive, real-time AI experiences that define competitive advantage in modern applications.

Service providers are uniquely positioned to enable this edge AI ecosystem. With infrastructure that spans from homes to regional data centers, they can deliver "last-mile AI cloud" capabilities that minimize network transit and latency. Their experience in managing distributed infrastructure and meeting strict service level agreements (SLAs) translates well to hosting AI workloads. By moving models to the edge, operators not only improve user experience but also retain greater control over data and decision-making. This supports strategic goals around operational efficiency, customer retention, and service differentiation.

Data locality and privacy are central to this shift. Hosting AI at the edge ensures that sensitive user data remains within the operator’s network. This minimizes exposure to external cloud providers and supports compliance with data sovereignty laws (e.g., GDPR, HIPAA). For regulated sectors like healthcare or finance, keeping AI processing local to the region or jurisdiction is essential. State-level compliance requirements, such as California’s CCPA/CPRA, further emphasize the importance of edge-hosted AI. These regulations mandate strict controls over data collection, storage, and usage, making localized AI processing a critical strategy for adhering to privacy laws. Operators can also offer fine-tuning and retrieval-augmented generation (RAG) services in compliance-sensitive environments, adapting AI to local user needs without exporting data. CableLabs has done primary research that demonstrates they are highly trusted in terms of privacy and security by consumers. Hosting AI at the edge leverages this trust.

Internally, operators benefit from deploying AI at the edge to optimize networks and reduce manual intervention. Models trained on telemetry from DOCSIS, fiber, or 5G systems can proactively detect faults, adjust parameters, or assist technicians with diagnosis and repair. For instance, an LLM fine-tuned on plant maintenance logs could recommend RF settings to improve SNR, reducing outages and truck rolls[37]-[39].

Edge AI enables personalized services with minimal delay and strong privacy guarantees, driving improvements in customer experience. Imagine a smart home assistant running on a gateway or set-top box that responds instantly while keeping all data on-site. Such services not only improve responsiveness but also build trust and loyalty by avoiding unnecessary data exposure.

The convergence of low-latency requirements, data locality mandates, and the distributed infrastructure of telecom providers sets the stage for edge-hosted AI. Operators can reduce latency to just a few milliseconds, improve reliability by reducing dependence on public cloud, and offer differentiated services tailored to regulatory and performance requirements. The following sections detail how this paper explores the architectures, use cases, and deployment strategies that make this vision practical and scalable.

Table 1- Key Benefits of Edge AI for Telecom Operators

Benefit Category	Specific Benefits	Impact
Performance	Reduced Latency	Improved user experience and enables new time-sensitive applications
	High Throughput	Increased processing capacity without overwhelming backhaul networks
Data Privacy & Compliance	Enhanced Privacy Protection	Builds customer trust and loyalty
	Data Sovereignty Compliance	Enables operations in compliance-sensitive markets
Network Operations	Infrastructure Optimization	Lower operational costs and improved network efficiency
	Reduced Manual Intervention	Cost savings and faster issue resolution
Service Differentiation	Personalized & Tailored Services	New revenue opportunities, competitive advantage, and premium offerings
Strategic Advantages	Reduced Cloud Dependency	Enhanced autonomy and negotiating power
	Leverages Existing Infrastructure	Maximizes ROI on existing investments
Developer Ecosystem	API Integration	Accelerates innovation and partner ecosystem growth

2.2. Key Takeaways

- **Latency**: Reduces from 50-150ms (cloud) to just a few milliseconds
- **Compliance**: Supports GDPR, HIPAA, CCPA/CPRA and other regulations
- **Operations**: Enables predictive maintenance and automated optimization
- **Trust**: Leverages service providers' established privacy reputation
- **Strategic Position**: Creates differentiated AI capabilities at the network edge

2.3. Edge AI and the Role of Edge Computing

Edge computing has emerged as a foundational architecture for enabling low-latency, high-performance digital services by placing compute resources closer to end users and devices. The ETSI MEC (Multi-access Edge Computing) Industry Specification Group aims to establish an open, standardized framework that enables smooth and effective application integration from various sources, including vendors, service providers, and third parties, across diverse Multi-access Edge Computing platforms from multiple vendors [1]. The Linux Foundation's CAMARA project is developing Edge Cloud APIs that offer developers standardized interfaces for discovering and accessing edge capabilities provided by operators, with these APIs now being incorporated into CableLabs' Network-as-a-Service project[2].

Edge AI is a natural extension of these architectures. By deploying AI inference workloads on edge compute platforms, operators can deliver real-time AI services that benefit from proximity, reduced backhaul traffic, and data locality. Extending existing APIs like CAMARA's to expose access to AI-optimized hardware and inference services can accelerate adoption by offering a seamless path for developers to host and invoke AI models at the edge. As demand for intelligent, secure, private, and latency-sensitive applications grows, Edge AI will drive further investment and standardization in edge compute infrastructure.

3. Use Case Analysis

This section explores **what** use cases are suitable for deployment at the edge and **why** the edge represents the optimal environment. For each use case, we rigorously align the design with clearly defined **Selection Criteria** (latency, data gravity, burst profile, connectivity, privacy/regulatory compliance, cost, and energy efficiency). Each case explicitly details the recommended topology, tuning parameters, and relevant KPIs.

3.1. Selection Criteria

These criteria help identify which use-cases are suitable for edge deployments. Further details on the benefits of deploying LLMs at the edge can be found in §3.1.

- **Latency Budget**: Control loops (<100 ms), interactive tasks (100–500 ms), batch processing (>1 s).
- **Data Gravity**: If raw data cannot leave the premises, process locally and transmit only structured events [40].
- **Burst Profile**: Assess whether workloads are steady or bursty; apply disaggregated prefill/decode techniques for intermittent workloads.
- **Connectivity**: Intermittent connections demand offline-first architectures and local storage/registry mirrors.
- **Privacy/Regulatory**: PII/PHI and sensitive logs require in-situ data masking, redaction, and retention policies.
- **Cost & Energy**: Aim for GPU utilization above 60%; prefer INT8/INT4 precision and smaller models augmented by techniques like LoRA when appropriate.

3.2. Exemplary Edge AI Use Cases

Service providers can leverage edge inference across access, aggregation, and core network segments to enhance reliability, efficiency, and customer satisfaction. While there are dozens of potential use cases, we focus on a few key examples that illustrate the breadth of edge AI applications and how the selection criteria apply.

3.2.1. Predictive Network Maintenance and Optimization

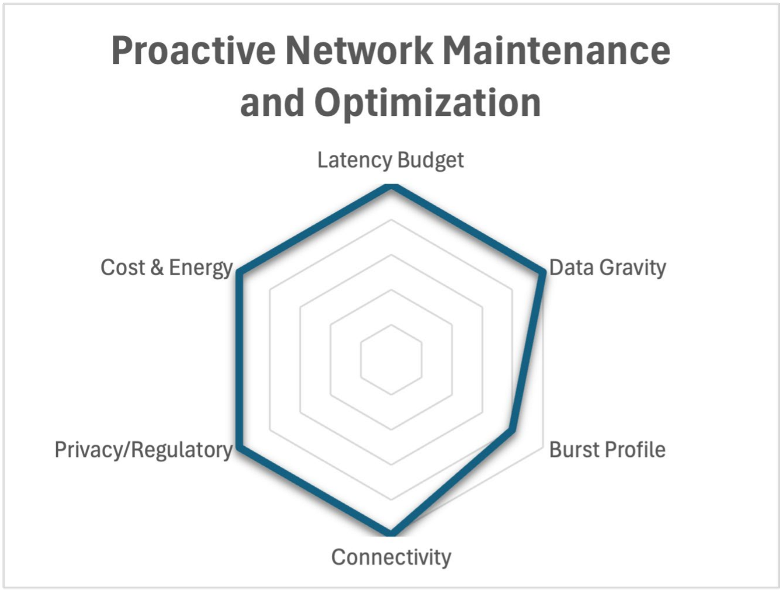

Figure 1- Predictive Network Maintenance Architecture for Edge AI Applications

Telecom operators need effective methods to quickly detect and resolve network anomalies, RF impairments, and Quality-of-Experience (QoE) issues to minimize disruptions, reduce operational costs, and shorten Mean Time To Repair (MTTR) [3]. Edge AI technologies can proactively identify and remediate such issues, significantly enhancing network reliability and customer satisfaction[37]-[39].

A hierarchical AI approach leverages Neural Processing Units (NPUs) embedded in modems, smart amplifiers, and network nodes. These NPUs process high-frequency RF and operational data locally in real-time, swiftly detecting intermittent impairments often missed by lower-frequency monitoring. Streaming such high-fidelity, voluminous data centrally is impractical due to bandwidth constraints and latency challenges. Local analytics at the edge thus enable efficient event detection and reporting of only actionable insights.

GPU-based edge servers at regional headends correlate information across multiple network segments, accurately localizing RF impairments and forecasting potential network congestion or QoE deterioration. This localized processing significantly reduces backhaul traffic by forwarding only critical, summarized data to centralized management systems.

In scenarios with intermittent or absent WAN connectivity, embedded NPUs and GPUs maintain autonomous monitoring and initiate immediate corrective actions, preserving network functionality and service availability independently of central oversight.

Distributing AI tasks strategically between NPUs at edge devices and GPUs at headends significantly improves operational efficiency, responsiveness, and overall network resilience.

3.2.2. Edge-Based Video Analysis

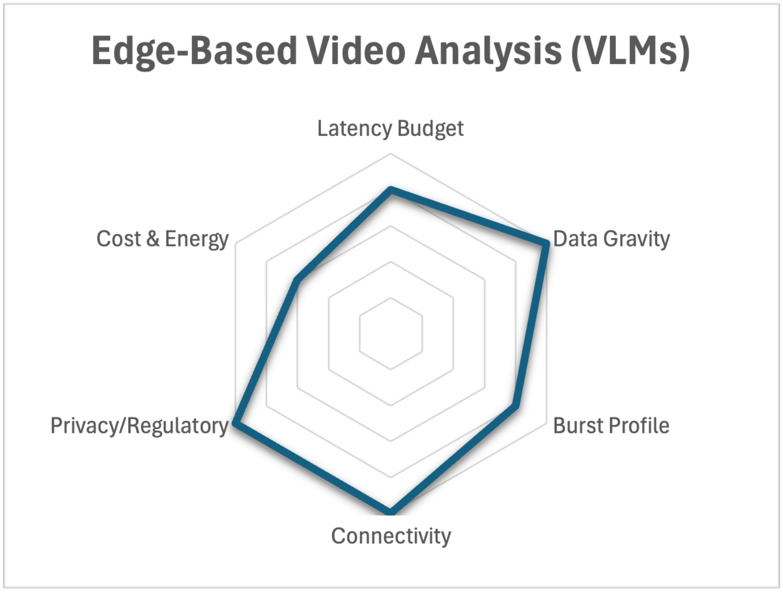

Figure 2- Edge-Based Video Analysis System for Real-Time Visual Intelligence

Real-time visual analysis with natural language understanding is transforming operations across multiple sectors. Security systems can now instantly identify and describe anomalies in CCTV feeds while keeping sensitive footage within the facility. Retail environments gain immediate inventory insights and interactive customer assistance. Industrial settings benefit from real-time quality assurance and safety monitoring that swiftly detects compliance violations and equipment issues directly onsite.

These capabilities come from vision-language models (VLMs), a rapidly emerging edge AI application that combines visual perception with natural language understanding through multimodal processing deployed directly at edge locations. This local deployment enables instantaneous analysis without cloud dependencies.

The edge deployment of these models directly addresses critical latency challenges inherent in cloud-based processing. By situating AI inference at the network edge, close to where data is generated, latency drops dramatically, eliminating the delays associated with distant data centers and enabling true real-time performance. Advanced model compression techniques now make it possible to run sophisticated VLMs on resource-constrained edge hardware[4], such as optimized models on edge gateways that can analyze and articulate real-time video feeds. These systems can maintain local databases storing images of household members and pets, enabling recognition of familiar faces while alerting users to security threats and deliveries, all processed locally.

This millisecond-level responsiveness proves essential in scenarios where delays can compromise safety or user experience, including industrial automation, healthcare diagnostics, live streaming analytics, autonomous vehicle safety systems, personalized content creation, and immersive gaming experiences. In these applications, the near-instantaneous processing enabled by edge-hosted VLMs transforms what's possible, making real-time AI a practical reality rather than a distant goal.

4. Background: Edge Computing for LLMs

With the explosion of large language models (LLMs) and vision-language models (VLMs), operators are exploring how to host these AI workloads at the edge of their networks. This section provides an overview of the motivations, benefits, and technical considerations for deploying LLMs at the edge. While this paper focuses on LLMs, many of the principles apply to other AI models as well.

Not all AI workloads are suitable for edge hosting, so we will also discuss the selection criteria that help identify which use-cases are best suited for edge deployments. The goal is to provide a foundation for understanding how edge computing can enhance AI capabilities while addressing the unique challenges of provider networks.

Training and serving large AI models requires significant computational resources. Traditionally, these workloads have been hosted in centralized cloud data centers, which offer high performance but can introduce latency and bandwidth challenges for real-time applications. Edge AI can be used to support federated or reinforcement learning, where models are trained on distributed data sources while maintaining privacy and reducing the need to transfer large datasets to a central location.

4.1. Benefits of Edge Hosting for LLMs

Deploying LLM inference workloads on edge infrastructure provides significant operational and strategic advantages, directly aligning with key scoring mechanisms:

Latency Reduction: Deploying AI models closer to end-users and data sources substantially reduces response times, enabling near-instantaneous interaction critical for applications such as conversational agents, AR/VR, and network control systems. Localized hosting achieves response times within milliseconds and significantly outperforming traditional cloud infrastructures.

Bandwidth Optimization: Local edge processing substantially reduces network traffic by transmitting only essential results or metadata, rather than raw, high-volume data streams (e.g., video feeds, sensor data). This optimizes bandwidth usage, reduces transit costs, and maintains core network capacity.

Data Privacy and Security: Edge deployments keep sensitive information such as voice recordings, personal data, and enterprise records onsite, aligning with regulatory compliance and data sovereignty principles. This significantly reduces data breach risks and facilitates adherence to local and international privacy laws, enhancing trust and marketability of AI-driven services.

Reliability and Availability: Edge infrastructure inherently improves system robustness and operational continuity by ensuring local nodes continue processing AI requests during network outages or cloud disruptions. This is critical during emergencies or connectivity interruptions, ensuring uninterrupted service for essential applications.

Energy Efficiency: Localized AI processing can be energy-efficient, especially when employing specialized, low-power AI accelerators and optimization techniques.

These edge AI deployment benefits collectively empower operators to enhance performance, optimize operational efficiency, and strengthen compliance, positioning edge computing as a foundational component for future-ready network infrastructures.

4.2. Edge and Cloud Hosting Models: Deployment Paradigms for Operators

As operators build out AI capabilities across their networks, they face a growing array of deployment choices. Understanding the hosting paradigms and their associated trade-offs is critical for aligning infrastructure, use case requirements, and operational goals. These models support both applications consumed internally by the operator or delivered as services to end customers.

The business of hosting compute and storage is similar to the content delivery network (CDN) model, which has traditionally depended on network peering between service providers and CDN operators. Today, multiple system operators (MSOs) are moving into this space by monetizing CDNs and edge computing, hosting them directly within their networks through edge compute platforms [45]. Some model hosting use cases must be deployed locally by the operator, while others may allow for either service provider hosting or peering arrangements with a hyperscaler. Each approach should be evaluated carefully, weighing the benefits and challenges of the available options.

4.2.1. *Hosting Models: IaaS, PaaS, and SaaS*

Infrastructure as a Service (IaaS):

- Provides virtualized or bare-metal access to edge or cloud compute.
- Offers maximum flexibility and performance tuning where AI application providers can deploy custom models, leverage specific AI accelerators, and tailor model quantization, batch sizes, and runtime behavior.
- Trade-off: IaaS requires significant engineering investment and expertise across AI frameworks (e.g., PyTorch, TensorRT), orchestration, and hardware optimization.
- Margin potential is typically the lowest among the three models, as confirmed by industry analysts due to the commoditization of infrastructure and high operational overhead [5][6].
- IaaS also enables third-party developers to innovate directly on operator edge platforms, creating use cases and services the operator may not have envisioned and opening new revenue opportunities through robust APIs.

Platform as a Service (PaaS):

- Offers turn key environments to deploy containerized models or inference services (e.g., via KServe, NVIDIA Triton).
- Simplifies deployment and scaling but still allows use of proprietary models and model selection.
- Ideal for teams that want to maintain control over models without managing the full infrastructure stack.
- Margin is moderate; support and maintenance responsibility is shared. Compared to IaaS, PaaS often achieves higher margins due to value-added service abstraction [5][6].
- PaaS enables a broader ecosystem by allowing external developers and vendors to deploy on shared operator infrastructure, fostering innovation and ecosystem expansion.

Software as a Service (SaaS):

- Provides turnkey AI solutions tailored to specific use cases—e.g., smart assistant APIs, log summarization, network anomaly detection.
- Lowest barrier to entry; best for rapid deployment and minimal integration effort.
- Trade-off: Users have limited control over models, parameters, and data handling logic.
- Margins are typically the highest, as SaaS enables recurring revenue at scale and captures more of the application layer value [5][6].

4.3. Edge Locations in Service Provider Networks

Edge locations within service provider networks play a pivotal role in enabling AI workloads closer to end-users. This section explores the diverse deployment options available across customer premises equipment (CPE), access network elements, headends, and regional data centers. Each location offers unique advantages and constraints, shaping how operators can optimize latency, bandwidth, and scalability for edge-hosted AI services.

4.3.1. Customer Premises Equipment (CPE)

Customer Premises Equipment (CPE), such as home gateways, cable modems, fiber ONTs, 5G fixed wireless routers and other user equipment, serve as strategic edge AI deployment points, enabling ultra-low latency and improved privacy through localized data processing [7].

Practical Considerations:

- **Resource Constraints:** Typical CPE hardware operates with limited power (<20 W), minimal cooling, memory, storage, and constrained computational resources, necessitating the deployment of compact, highly optimized models.
- **Integration Capability:** High-end DOCSIS 4.0 modems and fiber ONTs integrate advanced SoCs, supporting containerized AI services directly deployed by operators via initiatives like the Reference Design Kit (RDK) and prpl Foundation.
- **Operational Management:** Efficient AI operations require techniques such as duty cycling, low-bit quantization (e.g., INT4), and leveraging specialized low-power AI accelerators, ensuring uninterrupted primary device functionality.

Utilizing CPE for edge AI hosting enables telecom operators to deliver responsive, secure, and privacy-conscious AI applications directly to end-users, enhancing customer experience and trust. Importantly, data processed by AI solutions at the CPE edge may not need to leave the customer premises

4.3.2. Street Cabinets and Remote Nodes

Beyond customer premises equipment (CPE), telecom operators can leverage components of the access network—such as fiber nodes, smart amplifiers, street cabinets, and wireless towers—as valuable hosts for edge AI workloads. These elements, while resource-constrained, provide strategic venues for localized AI inference, especially for operational tasks managed by the operator.

Key Access Network Elements for Edge AI Hosting:

- **Fiber Nodes and Remote PHY Devices (RPDs):** Positioned at the neighborhood level, these can support light to moderate AI workloads. Emerging Distributed Access Architecture (DAA) allows for programmable compute at these points [8].
- **Smart Amplifiers:** Increasingly equipped with telemetry and digital control interfaces, smart amps can host embedded AI for real-time gain control, signal optimization, and predictive maintenance [9].
- **Wireless Towers and RAN Elements:** Radio access components, especially those enabled with MEC (Multi-access Edge Computing), can support AI workloads for mobile user analytics, video stream classification, and radio signal optimization.
- **Switches and Optical Transport:** For interconnecting fiber nodes, RPDs, and RAN elements to the core network.

Design Constraints:

- **Power and Environmental Limitations:** Most access elements are passively cooled and deployed in uncontrolled outdoor environments, requiring efficient and resilient AI workloads.
- **Compute Boundaries:** These locations could host compact processors, embedded NPUs, or GPUs suitable for quantized models.
- **Deployment Models:** Remote deployment and OTA updates are essential, often delivered as lightweight containers or model inference runtimes.
- **Not Always Available for User Workloads:** GPUs may be deployed at RAN elements specifically to support radio link processing/operations and may not be available for user facing AI applications.

Strategic Value: Access-layer AI enables ultra-local context-awareness and immediate response capabilities. Though limited in compute, these locations serve as a distributed, low-latency analytics mesh that supports proactive maintenance, network optimization, and resilient operations at scale. This positions the access network not only as a transport layer but as a foundation for autonomous and intelligent infrastructure.

4.3.3. Edge AI Deployment at Headends, Hubs, and Points of Presence (POPs)

Headends, hub sites, and regional points of presence (POPs) represent critical infrastructure within an operator's core network. These sites, sometimes called "metro edge" locations, provide an ideal balance of geographic proximity and computational capacity for AI workloads [7].

Advantages of Headend-Based AI Hosting:

- **Proximity to Users:** Located within tens of kilometers of subscribers, headends typically provide ~5-10 ms one-way latency over the access network, enabling responsive AI services without relying on distant hyperscalers [1].

- **Resource Pooling:** Unlike CPE or street cabinets, headends can host shared GPU clusters that serve thousands of users. This model improves utilization, allowing dynamic allocation of compute resources based on real-time demand.

- **Scalability:** Operators can scale inference capacity using server clusters with multiple GPUs, orchestrated via Kubernetes or container frameworks (e.g., KServe), similar to existing CDN infrastructure.

Model Size and Workload Capacity: With robust power, cooling, and rack space, headends can support LLMs in the 7-30B parameter range—or larger—using high-performance GPUs.

Operational Considerations:

- **Cluster Management:** Use container orchestration for autoscaling and high availability, aligned with platform practices already in place for CDN and edge caches.

- **Security and Compliance:** As operator-owned facilities, headends can support data sovereignty and GDPR-compliant data handling.

- **Hybrid Architectures:** Enable flexible workflows across edge tiers, including CPE, access nodes, and headends.

Strategic Role: Headends and POPs provide a scalable, resilient, and latency-sensitive compute tier between ultra-constrained edge devices and centralized cloud environments. As AI workloads grow in size and complexity, metro-edge sites will become a key pillar of telecom operators' generative AI strategy, enabling use cases that are too large for CPE but require lower latency than the public cloud can offer.

4.3.4. Regional Data Centers

At the **regional data center** tier—also known as the metro or core network data center—operators leverage their largest infrastructure sites to host advanced AI workloads. These facilities typically exist in major metropolitan areas, often functioning as interconnection hubs and peering points. While not as proximate to end-users as headends or CPE, they still offer significantly lower latency (10-30 ms RTT) than centralized cloud regions, enabling **metro-edge deployments** with a balance of responsiveness and computational scale.

4.3.4.1. High-Capacity Hosting

Regional data centers support the largest AI models or multi-modal models that require clusters of GPUs or TPUs. This tier enables:

- **Large model hosting**: Deploy models with tens of billions of parameters.

- **Batch processing**: Support large-scale data analytics, summarization, and training workflows.

- **Aggregation services**: Act as a hub to coordinate inference results from multiple edge locations.

4.3.4.2. Hardware and Operational Benefits

- **Scale and Efficiency**: Support for high-density AI racks (e.g., NVIDIA DGX, Graphcore IPU)

- **Robust Infrastructure**: High cooling and power budgets, strong security, and centralized monitoring

- **Economies of Scale**: Cost-effective hardware procurement and lifecycle management

4.3.4.3. Tradeoffs

- **Higher Latency than Edge**: ~10-30 ms RTT is suitable for most real-time use cases but may fall short for ultra-low-latency control loops.

- **Less Contextual Awareness**: Compared to street-level or CPE deployments, regional centers have a broader but less granular view of network and user context.

Regional data centers are a critical layer in the edge-cloud continuum, acting as powerful inference and coordination hubs. They offer:

- Superior computing power

- Scalable and secure multi-tenant hosting

- Flexible fallback for edge deployments

- Data gravity in dense population centers seems to be increasing, adding to the importance of AI capabilities at the metro edge [41]

As operators evolve their edge strategies, regional DCs will remain essential for delivering high-performance AI services, hosting large models, and bridging the gap between local processing and the hyperscale cloud.

4.4. Access Network Considerations for Edge AI

As operators deploy edge AI services, the access network plays a crucial role in determining performance, latency, and capacity. This section discusses how different access technologies such as DOCSIS for cable, PON for fiber, and 5G for mobile impact edge AI deployments. Each technology has unique characteristics that influence how edge compute can be optimized to deliver low-latency, high-bandwidth AI applications.

4.4.1. DOCSIS Networks

Cable operators are well-positioned to deliver edge AI services over DOCSIS broadband, especially with recent advancements in the DOCSIS standard.

Enterprises are prioritizing machine learning datafeeds along with other AI related traffic [42]. Low Latency DOCSIS (LLD) introduces dedicated low-latency service flows alongside traditional best-effort paths, significantly reducing round-trip time on the last mile allowing the prioritized to extend to the access network. When properly configured, LLD can enable inference requests and responses to traverse the network in just a few milliseconds. By classifying AI traffic (e.g., requests to LLM microservices) into these prioritized flows, operators can ensure consistent, low-latency performance.

Modern AI workloads often involve interactive upstream data, voice, video, sensor streams and making upstream bandwidth critical. DOCSIS 4.0 introduces Full Duplex and Extended Spectrum capabilities, enabling multi-gigabit symmetric speeds. DOCSIS 4.0 allows segmentation and dynamic profile management to ensure bandwidth availability for AI workloads.

Colocating edge compute nodes near DOCSIS infrastructure, such as in the headend next to the CMTS minimizes transport latency. Distributed Access Architectures (DAA) enable even tighter integration, with some operators exploring colocating compute with Remote PHY devices. While fully embedded solutions are still emerging, proximity between CMTS and edge compute (e.g., shared aggregation switches) offers immediate benefits.

DOCSIS networks, especially with DOCSIS 4.0 and LLD, are well-suited to support low-latency AI services. By combining latency optimization, upstream bandwidth provisioning, and thoughtful edge compute placement, cable operators can deliver AI experiences competitive with fiber. Proactive tuning and monitoring tools, including AI-powered analytics, further enhance performance and reliability.

4.4.2. Fiber Access (PON/FTTP)

Fiber access networks such as GPON, XGS-PON, and newer 25G/50G-PON standards offer high bandwidth and low latency, which are valuable characteristics for edge AI deployments. Typical one-way delays are in the range of 1-2 ms. This provides a strong foundation for supporting real-time AI services like interactive gaming, voice assistants, and object recognition, particularly when edge compute is located nearby.

PON systems provide bandwidth symmetry, in standards like XGS-PON that offer 10 Gbps in both upstream and downstream directions. This symmetry is beneficial for AI use cases that involve substantial upstream data, such as video frames from security cameras, sensor streams from IoT devices, or large user input files. It also allows edge AI systems to send larger result sets or generated content with minimal delay.

Fiber access networks support edge AI with robust bandwidth, symmetric data rates, and consistent low-latency performance. When combined with well-placed edge compute and appropriate QoS configurations, they can deliver a broad range of next-generation AI services.

4.4.3. Wireless Access (5G/FWA) and Edge AI

Wireless access, particularly 5G and Fixed Wireless Access (FWA), presents a mix of opportunities and constraints for edge-hosted AI. While 5G introduces low-latency and high-throughput capabilities, wireless links are inherently more variable than fiber or cable networks due to signal quality, interference, and mobility.

For mobile 5G, ultra-reliable low-latency communications (URLLC) theoretically allow sub-10 ms latencies. However, these features are primarily available in limited-scale private 5G deployments, mostly within enterprise or industrial environments [18]. Public 5G networks do not yet deliver consistent URLLC-grade performance due to shared spectrum and deployment constraints. As a result, latency and jitter in public wireless networks remain significantly higher and more variable than in wireline alternatives like DOCSIS or PON.

FWA offers more stable connectivity than mobile but still contends with wireless link dynamics. Environmental factors such as weather, obstruction, and cell congestion can introduce fluctuations in available bandwidth and delay. Average latencies for FWA connections range from 20-40 ms, with occasional spikes. While acceptable for many AI use cases, this latency is still higher than that of

DOCSIS 4.0 or XGS-PON, both of which offer lower round-trip latency and higher upstream bandwidth in practical deployments [26][27].

Bandwidth limitations also pose challenges. While FWA can support high downstream speeds, upstream capacity is often more constrained, especially under shared spectrum conditions. This can impact edge AI applications that require significant upstream data, such as live video analytics or real-time sensor streaming. Adaptive bitrate control, data compression, or edge caching strategies are often needed to ensure low-latency responses despite limited bandwidth.

While 5G and FWA can support edge-hosted AI, their suitability depends on the workload and network planning. Compared to DOCSIS and PON, wireless access introduces greater variability in latency and bandwidth. Effective deployment of AI services over 5G/FWA requires robust QoS mechanisms, proximity-aware compute placement, and adaptive inference pipelines to ensure consistent user experiences.

4.5. AI Placement Criteria

When deciding where to host AI workloads, a range of factors must be considered to balance performance, cost, and operational complexity. The following criteria provide a framework for evaluating edge, cloud, and hybrid hosting options:

- **Use Edge When**: Latency is critical, data must remain local, or cloud costs are prohibitive.
- **Use Cloud When**: Scale and elasticity are paramount, latency is tolerable, or workloads are transient.
- **Use Hybrid When**: A tiered model provides the best trade-off between responsiveness, cost, and resilience.

This framework enables operators to make informed decisions aligned with technical and business goals. The following table provides a comparative checklist to assist in evaluating AI workloads across latency, bandwidth, privacy, and cost dimensions.

Table 2- Edge, Cloud, and Hybrid Hosting Comparison Matrix

Criteria	Edge Hosting	Cloud Hosting	Hybrid Hosting
Latency Sensitivity	Excellent (<20 ms); ideal for real-time and interactive use cases	Higher latency (50-300+ ms); not suitable for ultra-low-latency tasks	Edge handles real-time, cloud supports heavy or background tasks
Data Locality	Keeps data on-prem or in-network; ideal for sensitive or regulated data	Requires data transfer to central locations; may trigger privacy concerns	Sensitive data filtered at edge, summarized results sent to cloud
Compute Intensity	Limited by power and thermal constraints; best for quantized or small models	Can host large LLMs and batch processes	Edge runs light models; cloud offloads complex or large tasks

Criteria	Edge Hosting	Cloud Hosting	Hybrid Hosting
Elasticity/Scalability	Fixed capacity; harder to scale dynamically	Highly elastic; scales with workload demand	Edge absorbs base load; cloud provides burst capacity
Deployment Speed	Slower; involves hardware provisioning	Fast; near-instant provisioning via public cloud	Use cloud first, then migrate to edge as usage stabilizes
Cost Efficiency (steady)	CapEx up front; potentially cheaper long-term for stable workloads	High OpEx; pay-as-you-go, but expensive for sustained usage	Balanced TCO; edge for base load, cloud for peaks
Cost Efficiency (variable)	Less efficient for bursty or sporadic traffic	Ideal for spiky workloads	Optimize load placement between layers
Operational Complexity	High; requires physical infrastructure, monitoring, updates	Low; managed by cloud vendor	Requires integration but spreads complexity
Resilience	Can operate standalone during outages	Impacted by network/cloud availability	Failover across layers improves reliability
Vendor Lock-in	Minimal; operator controls hardware and stack	High; risk of pricing or policy changes	Mitigates lock-in with multiple deployment options

5. Models and Fine-Tuning at the Edge

When deploying AI at the edge, understanding the types of models that can be effectively hosted is crucial. This section explores the various classes of AI models suitable for edge deployment, their characteristics, and considerations for fine-tuning to optimize performance in edge environments. The model size, architecture, and training methods all play significant roles in determining how well a model can perform in edge scenarios. Where memory and compute resources are constrained, selecting the right model class and size is essential for achieving low-latency, efficient inference [47].

5.1. Classes of Models to Host on Edge AI

As operators expand edge AI infrastructure, understanding the types of models suitable for edge deployment is essential. Each model class comes with distinct resource requirements, inference behaviors, and ideal use cases. This section introduces the major classes of AI models that can be hosted on edge infrastructure, including large language models (LLMs), vision-language models (VLMs), foundational machine learning (ML) models, fine-tuned variants, and emerging categories like media manipulation models.

5.1.1. Large Language Models (LLMs)

LLMs such as LLaMA, DeepSeek, and Mistral have become central to many AI workloads due to their ability to perform diverse natural language processing (NLP) tasks. These models enable local execution of tasks like summarization, Q&A, intent recognition, and conversational interfaces without cloud round trips [21][28].

5.1.2. Small Language Models (SLMs)

Much like LLMs, SLMs perform a variety of natural language processing (NLP) tasks, despite being substantially smaller than LLMs. SLMs typically have between 100 million and 7 billion parameters whereas LLMs typically have between 7 billion and 200 billion parameters; SLMs that are between 100 million and 2 billion parameters are usually optimized for edge hosting and more limited compute environments. SLMs are frequently created from LLMs using processes that include pruning, quantization, low-rank factorization, or distillation; the goal of each of these methods is to retain the most important learned representations from the LLM's training process so as to obtain similar performance on NLP tasks as the LLM while reducing the size of the model [43][44].

5.1.3. Vision-Language Models (VLMs)

VLMs combine text and image understanding, supporting use cases such as image captioning, visual anomaly detection, multimodal search, and augmented reality. VLMs are particularly effective in environments where visual data is abundant and needs to be interpreted quickly, such as security monitoring, retail shelf analysis, or manufacturing quality assurance [23].

5.1.4. Machine Learning Models (ML)

Smaller, task-specific ML models remain widely used on constrained edge devices. These include classifiers (e.g., decision trees, random forests), regressors, anomaly detectors, and time-series forecasters. Applications include sensor data monitoring, predictive maintenance, signal anomaly detection, and rule-based inference. These models often run on CPUs or NPUs embedded in CPE, smart amps, or access nodes, offering fast inference with minimal energy and memory footprint [31].

5.1.5. Fine-Tuned and Domain-Specific Models

Fine-tuned models apply pre-trained LLMs or VLMs to domain-specific tasks. For example, a general LLM might be adapted to understand cable network logs or technical support dialogues. Fine-tuning allows edge-hosted models to be more relevant, reducing the need for full-scale inference in the cloud. LoRA and QLoRA techniques allow this tuning to be deployed efficiently at the edge [22].

5.1.6. Media Manipulation and Generation Models

Emerging edge use cases involve real-time media processing: e.g., enhancing video quality, converting text to speech, generating synthetic audio/video responses, or translating spoken language. Models in this class include diffusion models (for image generation), voice cloning models, and video summarization tools. Due to compute demands, these are typically hosted in mid-tier edge locations (headends or regional data centers) and optimized via hardware acceleration or model pruning [26].

Together, these model classes enable a wide range of edge AI applications. Operators must consider hardware constraints, latency requirements, and workload characteristics when selecting model types for deployment. While LLMs and VLMs are often the most visible components of edge AI strategies, smaller ML models and specialized generative models also play critical roles in practical edge scenarios.

5.2. Fine Tuning and Model Size Considerations

Choosing the right model size is essential when deploying AI models at the edge. Model size impacts performance, latency, energy consumption, and the feasibility of fine-tuning. We categorize model sizes into three broad tiers: small, medium, and large, each suited for different edge scenarios and fine-tuning strategies.

Small Models (<7B parameters) Small language models are well-suited for low-power edge environments such as home gateways, IoT devices, or customer premises equipment. These models require minimal memory (often <8 GB with quantization), run efficiently on CPUs or entry-level GPU chips, and deliver fast response times. While they have limited generalization and linguistic capabilities, they are sufficient for simple applications like keyword recognition, FAQ bots, or on-device summarization.

Medium Models (7-30B parameters) Medium-sized models strike a balance between capability and efficiency. These models offer significantly better accuracy and reasoning, suitable for real-time customer interaction, language translation, or domain-specific inference. These models typically require a single high-memory GPU chips or a multi-GPU chip and multi-node setup, making them deployable at hub sites, headends, or micro data centers. Quantization can optimize their footprint further. Many operators find these models ideal for delivering powerful services within edge constraints.

Large Models (>30B parameters) Large models offer top-tier performance but are resource-intensive, often requiring dozens of GPU chips across many nodes and hundreds of gigabytes of memory. These models are generally reserved for regional data centers or centralized cloud platforms. Their deployment at the edge is typically limited to hybrid approaches, where the edge handles preprocessing and lighter inference while offloading complex queries to centralized infrastructure.

Fine-Tuning at the Edge Fine-tuning customizes models for specific domains or user contexts, enhancing performance without retraining from scratch. For privacy-sensitive or jurisdiction-specific workloads, edge-based fine-tuning ensures data locality and regulatory compliance. Operators are uniquely positioned to offer this as a managed service, using local compute infrastructure to adapt models close to where data is generated. This preserves data sovereignty while enabling rapid iteration.

Small models offer ease of deployment and privacy, medium models provide strong performance within realistic edge constraints, and large models require centralized resources or hybrid architectures. Operators can differentiate themselves by offering fine-tuning-as-a-service, enabling AI solutions that are tailored, compliant, and close to the user.

5.3. Retrieval Augmented Generation (RAG) at the Edge

Retrieval-Augmented Generation (RAG) is a powerful architecture that enhances language model performance by grounding responses in external data sources. Rather than relying solely on a language model's parametric knowledge, RAG incorporates a retrieval step that fetches relevant documents or embeddings, which are then passed as context to the model for more accurate and up-to-date generation. Deploying RAG at the edge unlocks low-latency, contextually enriched responses while preserving data locality and reducing reliance on cloud connectivity.

A typical RAG workflow involves two core components: a **retriever**, which finds semantically relevant documents or embeddings, and a **generator** (typically an LLM) that synthesizes the final output. At the edge, these components can be co-located on the same infrastructure used for general Edge AI workloads, such as headends, access nodes, or micro data centers. This co-location reduces round-trip time between the retrieval and generation steps, enabling responsive interactions that are tailored to locally relevant data.

Embedding Models and Vector Databases To support efficient retrieval, edge deployments must also include embedding models that convert incoming queries and documents into vector representations. These embeddings are stored in a vector database or semantic index, which supports fast similarity search. Running embedding models at the edge enables ingestion and indexing of local data in real time, keeping the knowledge base current without uploading data to the cloud. For example, logs, customer support tickets, or sensor data can be embedded and indexed locally, making them available for retrieval during RAG inference [30].

Operators can leverage this architecture for customer service assistants, proactive troubleshooting, or enterprise search across localized data stores. For example, a technician at a fiber node could ask, "Have there been similar RF issues in this area recently?" and receive a synthesized answer based on logs and diagnostics indexed at that node. RAG at the edge ensures this query never has to leave the local facility, preserving privacy and ensuring fast resolution.

Edge-deployed vector databases are optimized for constrained environments, and can be containerized alongside the embedding and generation models. Embedding models can be fine-tuned for local domain understanding, further improving the relevance of retrieved content [25][30].

Benefits for Edge Deployment

- **Low Latency**: Co-located compute and data retrieval reduces inference time.
- **Privacy and Compliance**: Sensitive data can remain onsite, satisfying data locality and regulatory requirements.
- **Up-to-date Knowledge**: Embedding pipelines at the edge can index recent logs, events, or sensor streams in real time.
- **Contextual Accuracy**: Retrieval grounds the model in real-world context, improving the reliability of output.

6. Benchmarking Methodology

This section outlines the methodology used to benchmark various edge LLM hosting platforms. Our goal is to provide a comprehensive and fair comparison of how well these platforms perform in real-world scenarios, particularly for service provider use cases.

6.1. Test Environment Setup

Our benchmarking was conducted in a controlled environment simulating edge compute conditions, using a server equipped with an NVIDIA L40 GPU. The L40 represents hardware that is now several generations behind the latest accelerator platforms. Nonetheless, its 48 GB of GDDR6 memory and efficient performance profile make it a practical reference point for deployments at edge data centers, such as telecom hub sites.

We defined AI agent planning as the key scenario for evaluation due to its essential role within agentic AI systems. This is especially pertinent as agentic AI systems move closer to the edge within MSO networks. While metrics like *time to first token* and *per-token latency* are important metrics, these metrics are limited to streaming responses found in chatbots, which are being abandoned in favor of more complex and flexible agentic AI systems. Within agentic AI systems, responses from tools to agents and between agents in the system are not streamed which makes *time to first token* and *per-token latency* largely irrelevant. Instead, the latency of the agent's response is the determining factor for how long it takes the entire agentic AI system to complete its purpose. While it takes some time for the agent to review its available tools and format its tool calls, the majority of the latency is introduced after all the tool calls are complete and the agent must generate its response. Additionaly, the latency introduced by the review of all available tools and for the tool calls to complete is primarily driven by the number of available tools, the quality of tool descriptions, and the complexity of the tools that are called instead of the location of the AI agent at the edge or its edge hosting mechanism. Consequently, we focus our tests on the latency introduced by the AI agent generating its response once all the tool calls are complete since this facet of the overall agentic AI system is most strongly influenced by the edge hosting mechanism of the AI agent and its location at the edge.

For testing, we used test cases of varying lengths and test cases which required varying lengths of outputs from the AI agent's response to obtain a comprehensive analysis of LLM response latency over a variety of real-world agentic AI applications. These test cases simulate varying complexity of tool responses and their impact on downstream AI agent response generation latency.

Our **test models and data** were chosen to represent realistic service provider use cases. We primarily benchmarked with Meta's Llama 3.1 8B Instruct model, which is a high-performing 8B parameter model designed for single-GPU use, as a proxy for a general-purpose LLM that might be deployed at edge. For evaluation, we tested 119 total queries related to the SCTE 280 document in an agentic RAG setting [28]. We selected agentic RAG as our testing scenario due to the varying levels of detail required by AI agent responses within an agentic RAG system. Additionally, agentic RAG is incorporated into the planning phase of more complex agentic AI systems, since planning is paramount to the success of agentic AI systems. Within larger agentic AI systems that utilize agentic RAG for their planning capabilities, the planning phase of the agentic AI system is the largest contributor to the latency of the entire agentic AI system. Therefore, conducting a comprehensive analysis of the most important component of complex agentic AI systems and the feasiblity of placing this component on edge devices determines the latency and feasibility of hosting the entire agentic AI system at the edge. The 119 queries that were provided to our edge-hosted AI agent were developed in conjunction with and were reviewed by, numerous industry experts across several SCTE and CableLabs working groups to ensure that these queries are representative of the types of queries that field technicians working in the field would need to be able to answer to diagnose and successfully resolve downstream cable impairments. In some cases, these queries can be answered by a single sentence, while others involve longer and more detailed responses. This variety in expected response length and the corresponding latency associated with the response generation by our edge-hosted AI agent determines the robustness of the edge-hosted AI agent across a variety of planning queries which are paramount to the success of a downstream impairment resolution agentic AI system. By using a consistent model across platforms, we isolate the overhead and optimization of the serving stack rather than differences in model.

When generating the responses using Llama 3.1 8B Instruct for each of the questions in the question set and each of the hosting methods, we used a standard set of arguments in the LLM call to ensure reproducibility and consistency across the evaluations. Specifically, we set the temperature of the model to 0 since we are using an agentic RAG context and we want the AI agent to stay as close to the facts provided by the expert retrieved information as possible, since this will result in the most accurate planning of tasks for proceeding AI agents within the agentic AI system. Observe that setting the temperature to 0 ensures that the LLM is as minimally creative as possible which produces more consistent LLM responses that take fewer creative liberties with the provided expert information in their response. Additionally, we set streaming to False to ensure that we are following an agentic RAG context where the responses from AI agents are returned in one piece once the generation process is complete, rather than streamed token-by-token during the generation process. Finally, we set the maximum tokens to 1024 as this provides a sufficient number of generated tokens across the 119 question set for a complete response from the AI agent but is not so expansive as to permit overly verbose and potentially hallucinated responses, an important consideration within agentic RAG and agentic AI systems.

6.2. Performance Metrics

We evaluated each platform on a set of **performance metrics** that together give a holistic picture of how well it meets the needs of edge deployment. The primary metric is related to latency, but we also monitored resource usage and other operational considerations like power, given their importance at the edge.

For **inference latency**, we measure the elapsed time from when an AI agent receives its tool message(s) and when the final response is generated by the AI agent. Crucially, this elapsed time reflects the time required for an AI agent to utilize information retrieved from expert sources, in this case from the SCTE 280 standard document, and synthesize that information into a comprehensive response that will guide the actions of the downstream AI agents within a larger agentic AI system. This inference latency determines the delay required for an AI agent to plan its actions such that the larger agentic AI system can accomplish its purpose. Inference latency is measured in seconds, where a lower inference latency times enable the other AI agents to begin acting faster, as opposed to waiting longer for the planning agent.

6.3. Test Models and Datasets

For our benchmarks, choosing appropriate **models and datasets** was crucial to ensure results that are relevant to service providers (MSOs) considering edge AI deployments.

Our **model selection criteria** revolved around using a model that is both widely used in the open-source community and realistically deployable at the edge. We focused on the Llama family of models by Meta as our primary LLMs under test: specifically the 8B parameter variant. This model has excellent tool-calling capabilties in the open domain, and importantly, it is available for research/benchmarking with reproducible performance characteristics. Llama 3.1 was chosen because it supports robust tool-calling, both of which are essential for edge-hosted AI agents. Additionally, many of the serving platforms (Ollama, llama.cpp, etc.) are highly optimized for Llama/LLM-type architectures. Additionally, Llama 3.1 has no built-in reinforcement learning fine-tuning (unlike ChatGPT/GPT-4 which include RLHF), meaning out-of-the-box it’s more of a raw model; this was useful to see how platform performance is on a raw model. We steered clear of models larger than 30B for most tests, as discussed, because those aren’t yet practical for single-edge-server deployment given our hardware.

To reflect service provider needs, we crafted **representative workloads** that mirror common use cases outlined in Section 2. For example:

- **Field Technician Support AI Agents**: We worked closely with several CableLabs and SCTE working groups to define a corpus of 119 questions related to the SCTE 280 standard document that define essential knowledge for understanding, diagnosing, and resolving a variety of downstream cable impairments. These questions were written, reviewed, and approved by numerous veteran field technicians and cable industry veterans. These questions were selected to reflect the breadth and depth of content covered by the SCTE 280 standard as well as the breadth and depth of knowledge required for field technicians to accurately and effectively diagnose and resolve downstream cable impairments in the field.

The real-world scenarios were drawn from actual data as described - e.g., questions that veteran field technicians consider essential to understanding, diagnosing, and resolving downstream cable impairments. We balanced these by difficulty: some straightforward ("What are water waves?") and some complex ("What is the problem with network designs that have an equalizer placed mid-span between the amplifier and the end-of-line?"). We also varied the expected length of responses: some should be one sentence, others a few paragraphs (for instance, summarizing the various manifestations of downstream cable impairments). This allowed us to measure platform behavior for both short and long outputs.

In summary, our test models and datasets were chosen to **represent key MSO use cases** and to cover a range of input/output lengths and complexities. By using a consistent model across platforms and a mix of synthetic stress tests plus real-world-like queries, we ensure that the comparisons highlight differences in the serving platforms themselves. The next section will detail those platform architectures and present the results using the metrics from Section 5.2 on the workloads described here.

7. Tools for Hosting Models

This section introduces the two roles in LLM edge deployments and how they interact. We present them separately to clarify responsibilities while acknowledging areas of overlap; detailed numbers remain in §6.8 (comparison matrix) with explicit hardware (GPU/VRAM/CPU), quantization, context, and decoding parameters.

Backends (inference engines). Implement token execution and memory policy: attention kernels, batching/scheduling within a process, KV-cache and prefix sharing, quantization/format handling, and (often) a lightweight HTTP/OpenAI-style server. Examples in §6.1: vLLM, TensorRT-LLM, SGLang, llama.cpp, and Ray/Ray Serve (as a runtime substrate that often embeds engines).

Orchestration platforms. Provide the network/API surface and multi-node control plane: request routing, autoscaling, rollouts/canaries, resource isolation, and observability/policy. They select and host one or more backends. Examples in §6.2: Ollama (single-node), KServe (Kubernetes-native), and NVIDIA Dynamo (aggregated/disaggregated).

Where the lines blur.

- Some backends ship servers and limited scaling features (e.g., vLLM/SGLang expose APIs; llama.cpp includes sample servers).
- Some platforms influence inference behavior (e.g., KServe concurrency targets; Dynamo's prefill/decode split), so tuning spans both layers.
- Ray Serve can act as a backend runtime for multi-stage pipelines while also offering orchestration-like routing and autoscaling.
- Ollama wraps llama.cpp and provides minimal orchestration, making it a pragmatic single-node bridge between the two roles.

Use this framing as you read §6.1 and §6.2, and consult §6.8 for quantitative, apples-to-apples comparisons.

When offering inference and hosting as a service, providers can choose both the orchestration layer and the inference backends-often selecting stacks optimized for their hardware and network topology. This maximizes flexibility to manage demand and, as a higher-level service, reduces customer onboarding friction and can command a premium. As this section describes, some workloads benefit from customizing both the backend and the orchestrator to a specific model and SLO. For these cases, exposing **GPU/NPU-as-a-Service** with direct device access from containers can be sufficient and operationally simpler, but it is best suited to more sophisticated users.

7.1. Backend and Orchestration Comparison

We compared the performance of each of the backend configurations using a test set containing 119 questions related to SCTE 280 that capture essential knowledge for understanding, diagnosing, and resolving downstream cable impairments; these questions were developed and reviewed by numerous veteran field technicians. Each of the model hosting environments was tested using latency as the evaluation metric; here latency considers the elapsed time in seconds from when the edge-hosted LLM receives its input and when it finishes generating its output.

In order to compare the latency results across model hosting strategies, we generated two plots, which are shown below. The first plot, shown in xref:fig-latency-dist, shows the distribution of response latency values by hosting method, while the second plot, shown in xref:fig-avg-latency, shows the average latency across all questions by hosting method. That is, xref:fig-latency-dist plots the latency value for each 119 questions in the question set for each of the hosting methods, allowing us to compare the distribution of latency values across the responses for each of the 119 questions in the question set. While knowing the latency statistics is useful, analyzing the distribution of latency values across all the questions in the question set allows us to determine whether there is any skew present in the response latencies and whether there are response latency values that are outliers relative to the rest of the distribution; these details are not captured by summary statistics such as the standard deviation, mean, or median.

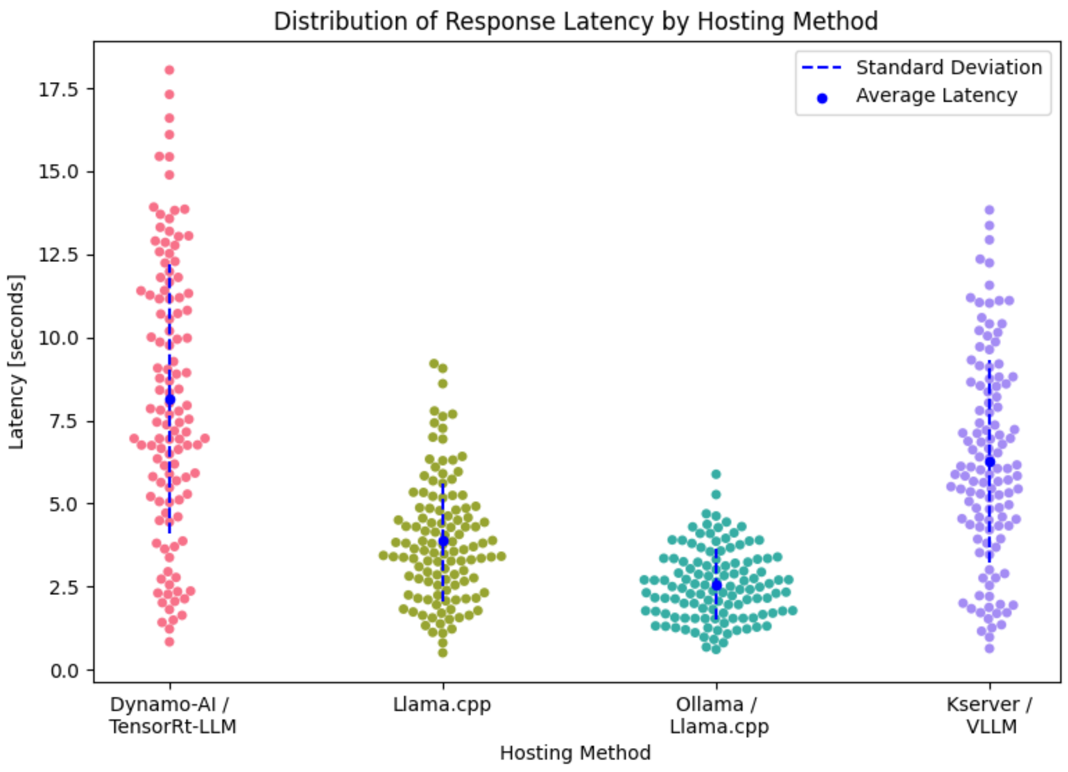

Figure 3- Distribution of Response Latency Values by LLM Hosting Method

Notice that in xref:fig-latency-dist we see that the distribution of response latency values for the Dynamo-AI/TensorRT-LLM hosting method range between 1 second and 18.5 seconds. Further notice that Dynamo-AI/TensorRT-LLM has the largest range in response latency values out of all the hosting methods. Observe that Dynamo-AI/TensorRT-LLM has a very similar distribution of response latency values where the density remains very consistent between 1 second and 14 seconds. That said, there is slightly more density in response latency values clustered between 7 and 7.5 seconds; overall, though, it is a very heavy-tailed distribution in both the left and right tails. Even given the heavy-tailed nature of the distribution, however, the long tail of low density values between 14 and 18.5 seconds gives the distribution a very slight right skew. While there is a higher density of response latency values between 7 and 7.5 seconds, the mean response latency is around 8 seconds as denoted by the location of the blue dot on the distribution. The blue bars that extend from the mean response latency indicate the standard deviation of the distribution and how far it extends both above and below the mean response latency. Notice that one standard deviation above the mean is 12 seconds and one standard deviation below the mean is 4 seconds, which indicates that the standard deviation of the distribution is 4 seconds. Further notice that the Dynamo-AI/TensorRT-LLM hosting method has the highest mean response latency and largest standard deviation out of all the hosting methods. Overall, this means that the Dynamo-AI/TensorRT-LLM hosting method takes the longest amount of time to generate responses and has the highest variability in response time.

For the llama.cpp hosting method, we see that the response latency values vary between 0.5 seconds and 9 seconds with the majority of latency values being between 3 seconds and 4.5 seconds. Notice that the right tail of the distribution of response latency values for the llama.cpp hosting method is longer than the left tail of the distribution and that the right tail has a higher density of response latency values than the left tail of the distribution. This indicates that the distribution of response latency values for the llama.cpp hosting method is right-skewed. Observe that the mean response latency is around 4 seconds as evidenced by the location of the blue dot. Further observe that one standard deviation below the mean is 2 seconds and one standard deviation above the mean is 6 seconds, so the standard deviation for this distribution is 2 seconds. Notice that the llama.cpp hosting method has the second lowest mean response value, the second lowest standard deviation, and the second smallest range behind only the Ollama/Llama.cpp hosting method. Hence, the llama.cpp hosting method takes the second shortest amount of time to generate responses and has the second lowest variability in response time out of the hosting methods.

Observe that the Ollama/Llama.cpp hosting method has the smallest range of response latency values out of the hosting methods, with the response latency values being between 1 second and 6 seconds. Further observe that the majority of the response latency values are clustered between 1.5 seconds and 3 seconds. While the majority of the response latency values are between 1.5 seconds and 3 seconds, there is a higher density of response latency values above 3 seconds than below 1.5 seconds. This means that the right tail of the distribution is heavier than the left tail of the distribution; additionally, we notice that the right tail of the distribution is longer than the left tail of the distribution. Hence, the distribution of response latency values for the Ollama/Llama.cpp hosting method is right-skewed. Notice that the mean response latency for the Ollama/Llama.cpp hosting method is around 2.5 seconds based on the location of the blue dot on the distribution. Furthermore, we see that one standard deviation below the mean is 1.5 seconds and one standard deviation above the mean is 3.5 seconds, so the standard deviation for this distribution is 1 second. Comparing this distribution to the other distributions, we see that the Ollama/Llama.cpp hosting method has the smallest range of response latency values, the lowest mean response latency, and the lowest standard deviation of all the hosting methods. Therefore, the Ollama/Llama.cpp hosting method takes the shortest amount of time to generate responses and has the least variability in response time out of the hosting methods.

Finally, the distribution of Kserver/VLLM response latency values has latency values that vary between 1 second and 13.5 seconds, with the majority of latency values between 5 seconds and 7 seconds. Notice that the Kserver/VLLM hosting method has the second largest range in response latency values out of the hosting methods, second only to the Dynamo-AI/TensorRT-LLM hosting method. Observe that while the majority of the density in response latency values is clustered between 5 and 7 seconds, there is a relatively high density of values in both tails. Further observe that the right tail is slightly longer and more dense than the left tail, meaning that the distribution of response latency values for the Kserver/VLLM hosting method is slightly right-skewed. Additionally, notice that the mean latency value is around 6 seconds which is in the middle of the cluster of latency values, indicating that the skew in the distribution is very minor. The blue bars indicating the standard deviation for this distribution show that one standard deviation below the mean is 3 seconds and one standard deviation above the mean is 9 seconds. Hence, the standard deviation for this distribution is 3 seconds. When compared to the other distributions, we see that the distribution of response latency values for the Kserver/VLLM hosting method has the second largest range, the second highest mean latency, and the second largest standard deviation. Thus, the Kserver/VLLM hosting method takes the second longest amount of time to generate responses and has the second largest variability in response time of the hosting methods.

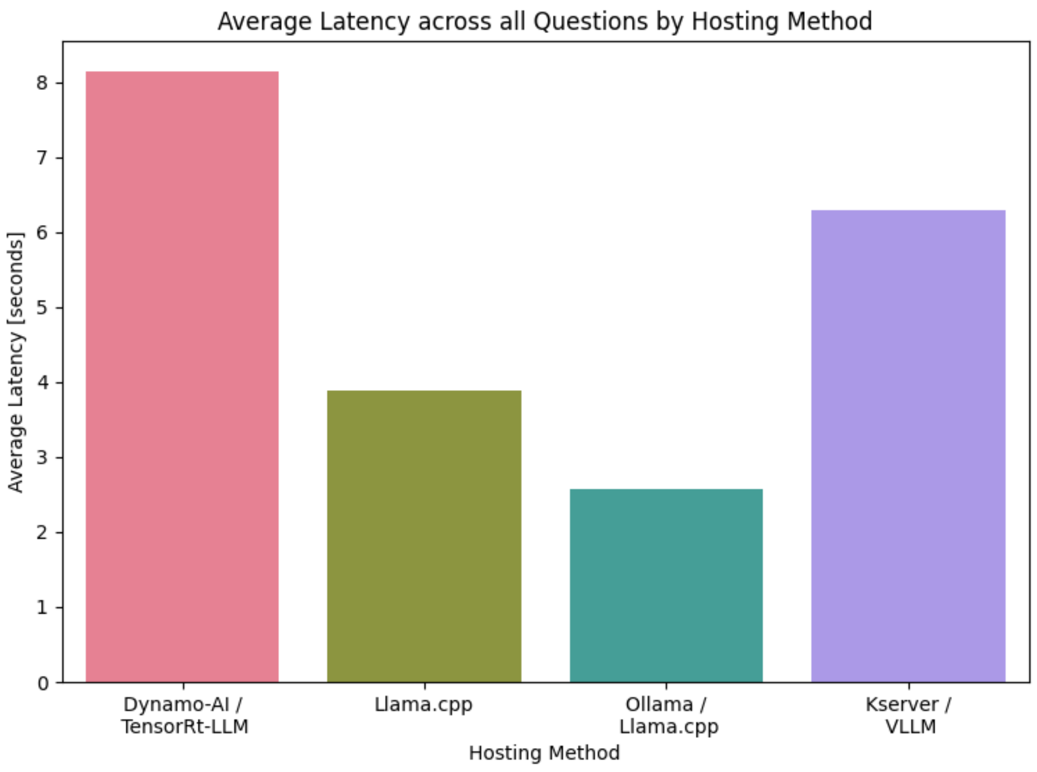

Figure 4- Average Response Latency Comparison Across LLM Hosting Methods

For ease in comparing the average latency value across the hosting methods, xref:fig-avg-latency shows a comparison of the average response latency value for the different hosting methods. In xref:fig-avg-latency, we see that the Dynamo-AI/TensorRT-LLM hosting method has an average response latency of 8.1 seconds, the llama.cpp hosting method has an average response latency of 3.9 seconds, the Ollama/llama.cpp hosting method has an average response latency of 2.7 seconds, and the Kserver/VLLM hosting method has an average response latency of 6.2 seconds. Hence, we see that Ollama/Llama.cpp has the lowest average response latency, followed by llama.cpp, then Kserver/VLLM, and finally Dynamo-AI/TensorRT-LLM. Therefore, Ollama/Llama.cpp has the fastest LLM responses out of the hosting methods, Dynamo-AI/TensorRT-LLM has the slowest LLM responses out of the hosting methods, llama.cpp has the second fastest LLM responses but is significantly slower on average than Ollama/Llama.cpp, and Kserver/VLLM has the second slowest LLM responses and is significantly slower than both Ollama/Llama.cpp and llama.cpp.

7.2. Backends

Backends sit closest to the silicon and determine how effectively **hardware acceleration** (GPU, NPU) is used. Their kernels, schedulers, and memory policies dominate **time-to-first-token (TTFT)**, **tokens/sec**, and **tail latency** often more than the orchestration layer. They can be tuned for specific **models** and **use cases**:

- **Quantization & weight format**: FP16/BF16/FP8 vs. INT8/INT4; GGUF/AWQ/GPTQ trade accuracy for lower memory/latency.

- **Attention & cache policy**: Paged/Radix attention, prefix sharing, KV-cache size/eviction, and prefill/decode behavior.
- **Batching & scheduling**: continuous batching, max batch tokens, speculative decoding (draft models), and sampling knobs.
- **Parallelism strategy**: tensor/pipeline/expert parallelism; multi-LoRA batching; CPU↔GPU layer offload on constrained nodes.
- **I/O & memory**: pinned/pooled memory, CUDA graphs/engines, prefetch/warmers, NUMA placement.

Single node vs. many nodes. On a **single node**, an aggregated backend minimizes hops and usually yields the best TTFT, but is bounded by local VRAM and PCIe bandwidth—ideal for small/medium models and predictable traffic. Across **many nodes**, scale comes from data/tensor/pipeline parallelism or **disaggregated prefill/decode**; this favors large models, long contexts, or bursty loads but introduces interconnect sensitivity (NVLink/PCIe/IB), KV-cache movement costs, and routing/topology tuning. Choose the topology based on latency SLOs, context length mix, and how much prefix reuse you expect.

Table 3- Model Hosting Backend Comparisons

Engine	Key Capabilities	Performance Highlights	Strengths	Limitations
vLLM	• Paged KV cache • Continuous batching • Tensor / pipeline parallel; OpenAI-style API	• Stable p95 under mixed prompt loads • Cold-start cost mostly in weight load	• Mature scheduler • Broad model support, active ecosystem	• Prefill / decode disaggregation handled externally
TensorRT-LLM	• Graph-level optimizations • FP8 / INT8 / INT4 kernels • Multi-GPU parallelism	• Best per-GPU latency and throughput when tuned	• Hardware-tailored for NVIDIA; ideal for metro/core edge	• Build complexity • Limited portability across non-NVIDIA stacks
SGLang	• RadixAttention, paged attention • Speculative decoding • Built-in prefill / decode split	• Large gains with prefix reuse and branchy dialogs	• High cache reuse efficiency • Features for assistants & tools	• Rapidly evolving; requires careful version pinning
llama.cpp	• Minimal deps C/C++ engine • Aggressive quantization	• Competitive for small models at low concurrency	• Runs on CPU-only edge nodes; tiny footprint	• Limited production features; needs external wrapper

Engine	Key Capabilities	Performance Highlights	Strengths	Limitations
	(GGUF) • CPU-friendly			
Ray / Ray Serve	• Actor-based scaling • HTTP ingress, autoscaling • DAG orchestration	• Good for multi-stage pipelines; acceptable overhead	• Flexible graphs; horizontal scaling	• Requires extra ops layers for tenancy & strict SLOs

7.3. Orchestration Platforms

This subsection explains how orchestration layers expose and operate inference services across one or many nodes, and the criteria we use to compare them.

Evaluation criteria

- **API & compatibility**: REST/gRPC, OpenAI-style, streaming, and support for tool/function calling.
- **Scaling & scheduling**: autoscaling (HPA/KPA), concurrency targets, batching influence, and scale-to-zero behavior.
- **Topology & placement**: single node vs. many nodes; GPU binding/affinity; aggregated vs. disaggregated prefill/decode; interconnect awareness (PCIe/NVLink/IB).
- **Rollouts & reliability**: revisions, canary/blue-green, warmers, cold-start mitigation, and failover policies.
- **Multitenancy & isolation**: namespaces, quotas, resource limits, and per-tenant policies.
- **Observability**: token/request metrics, traces, logs; queue depth; SLO-driven alerts.
- **Security & networking**: ingress, mTLS, authN/Z, network policies, and egress controls.
- **Model lifecycle & storage**: image build pipelines, weight distribution, local NVMe/object-store caching.
- **Edge readiness**: intermittent links, offline tolerance, and remote operations.
- **Cost efficiency**: GPU utilization, bin-packing, and energy/headroom.

Single node vs. many nodes

On a **single node**, minimal hops reduce TTFT; place weights locally and prefer aggregated engines. Across **many nodes**, routing and autoscaling enable burst handling and larger models but add network overhead and cache movement-tune placement, warmers, and interconnect to protect p96.

Table 4- Model Hosting Frontend Comparisons

Platform	**Role**	**Why Pick It**	**Common Back-Ends**
Ollama	Lightweight single-node wrapper around **llama.cpp** with simple REST model management. Ideal for low-volume edge sites and lab work.	Fast time-to-value; easy model pull/tag flow; can run CPU-only.	**llama.cpp** (native) for small GGUF models
KServe	Kubernetes-native serving layer built on Knative autoscaling, supports canary rollouts and rich observability. Runs any backend packaged as a container.	Strong SRE tooling (HPA/KPA, scale-to-zero, revision control); integrates with cluster policies, gateways, and GitOps pipelines.	**vLLM** (default) and other containerized engines
NVIDIA Dynamo	Distributed inference orchestrator that supports aggregated or disaggregated prefill/ decode and optional routing. Engine-agnostic.	Delivers low latency at scale; clean separation of routing, prefill, and decode; exploits high-bandwidth interconnects.	**SGLang** (prefix sharing, speculative), **TensorRT-LLM** (raw perf), **vLLM** (balanced)

7.4. Composition Guide

Table 5- Recommended LLM Platform Configurations by Edge Deployment Context

	Recommended pairing	Rationale
Edge context		
Single-GPU PoP, light traffic	**Ollama + llama.cpp**	Minimal footprint, quick setup; GGUF quants; no need for multi-node.
Metro-edge cluster, mixed workloads	**KServe + vLLM**	Mature autoscaling/rollouts; strong concurrency and long-context support.
Performance-critical NVIDIA GPUs	**KServe or Dynamo + TensorRT-LLM**	Lowest latency when engines are prebuilt; leverage GPU features.
Chatty assistants with prefix reuse	**Dynamo (disaggregated) + SGLang**	RadixAttention + PD split improves throughput/p95 under bursts.

Edge context	Recommended pairing	Rationale
Pipelines (RAG/CV/ASR + LLM) on shared nodes	**Ray Serve + (vLLM/SGLang/TensorRT-LLM)**	Compose multi-stage graphs; scale non-LLM tasks alongside inference.

7.5. Edge LLM Inference Operations Checklist

Use this checklist to harden and operate LLM inference at edge sites. Prioritize minimizing cold starts, ensuring predictable p95 latency, securing model assets, and maintaining service continuity—even with intermittent connectivity. Select controls that align with your Service Level Objectives (SLOs) and site-specific constraints.

7.5.1. Warm Path Optimization

- Pre-pull model containers and weights.
- Pre-load attention kernels and build inference engines (e.g., TensorRT/CUDA).
- Pin model versions and enable auto-warmers.

7.5.2. Concurrency and Throughput

- Tune pod concurrency settings to match latency targets and model size.
- Limit max tokens per request to control resource spikes.
- Fine-tune batching and prefill behavior for smoother performance.

7.5.3. Model and Cache Storage

- Keep active weights and KV caches on fast local NVMe.
- Use node-local or cluster-level S3-compatible object storage for durability across restarts.
- Support range GETs and prefetching for large models and long-context use cases.
- Encrypt at rest.
- Sign artifacts and validate during deployment.

7.5.4. Network & Egress Management

- Restrict outbound egress from inference nodes.
- Mirror registries and model storage for disconnected or air-gapped edge sites.
- Version and pin container images and model files to ensure deployment reproducibility.

7.5.5. Security Controls

- Front inference endpoints with gateways that enforce authentication, authorization, and rate limits.
- Apply strict network policies and namespace isolation.
- Scrub prompts and model outputs from logs to preserve user privacy.

7.5.6. Observability & Monitoring

- Export metrics: request rate, tokens/sec, cache hit rate, and queue depth.
- Set alerts for VRAM headroom, KV-cache evictions, and model/weight loading latency.

8. Recommendations and Best Practices

This section provides practical recommendations for service providers looking to deploy and manage large language models (LLMs) at the edge. These guidelines are designed to help operators navigate the complexities of edge AI, optimize performance, and deliver reliable, low-latency services to end users. Not all recommendations will apply to every operator, but they provide a comprehensive framework for evaluating and implementing edge AI solutions.

Operators are increasingly starting their edge AI journeys by deploying AI to support internal operations. This approach provides immediate benefits, including improved network reliability, reduced manual interventions, and greater operational insight. Hosting and managing these workloads locally allows operators to maintain end-to-end control over data privacy and performance while minimizing dependency on external services.

A phased deployment model is often preferred. Many operators begin with a single metro cluster to establish core infrastructure and validate performance. Key metrics to track in early deployments include time-to-first-token (TTFT), throughput per watt, and reductions in truck rolls. Operators can then expand incrementally to edge locations such as hub sites and street-level cabinets, prioritizing sites with strong return on investment. A common threshold is a projected ROI of 1.3 times or greater within three years.

To manage risk during rollout, practices such as blue-green and canary deployment models are recommended. These allow for gradual traffic shifts between model versions and fast rollback if latency, accuracy, or behavior issues are detected. Technologies like KServe support such rollout patterns, providing robust tooling for managing inference lifecycle at scale [36].

For monetization, multiple service models are possible, depending on the operator's maturity and strategy. Infrastructure-as-a-Service (IaaS) can be a starting point, targeting technically advanced partners who want direct access to edge resources. Platform-as-a-Service (PaaS) extends value by offering managed environments for third-party application developers to deploy and scale AI models. Software-as-a-Service (SaaS) unlocks the highest margins by providing tailored, turnkey AI services such as personalized assistants, predictive maintenance, or intelligent monitoring. Each model comes with different operational complexity and margin potential.

Operators should also consider the opportunity to provide fine-tuning-as-a-service and Retrieval Augmented Generation (RAG) support, especially in industries requiring data locality and privacy. Edge platforms can enable enterprises to adapt models close to where data is generated while maintaining control over compliance-sensitive workloads.

Edge AI deployment strategies should balance internal optimization, phased infrastructure expansion, and incremental monetization through flexible service models. Each operator's path will differ, but starting with high-value internal use cases and building a foundation for external services provides a strong, adaptable roadmap.

9. Future Directions and Emerging Technologies

These future directions highlight the ongoing evolution of edge AI and its potential to transform service provider operations, enhance customer experiences, and drive innovation across industries. By embracing these advancements, operators can position themselves at the forefront of the AI revolution.

9.1. Multi-Model Hosting and Large-Model Sharding

As edge AI adoption accelerates, future architectures must prioritize scalability, adaptability, and efficient resource utilization. One key direction involves supporting **multi-model hosting** on shared infrastructure. This allows multiple LLMs, VLMs, and task-specific models to be served simultaneously from a single node or cluster. Techniques such as model multiplexing, dynamic batching, and on-demand loading enable high resource utilization while minimizing latency impact. Some model-serving frameworks have early support for concurrent inference with memory-optimized scheduling and token-level interleaving.

In parallel, **large-model hosting across multiple systems** is gaining traction. With some LLMs requiring hundreds of gigabytes of memory, distributed inference is often necessary. Techniques like model parallelism and pipeline parallelism, as seen in frameworks such as DeepSpeed [32] and Hugging Face Accelerate [33], allow operators to shard large models across several GPUs or nodes. When deployed at the edge, this enables inference from high-parameter models without requiring hyperscale data center infrastructure. Integrating these solutions with orchestration platforms like KServe [34] and Kubernetes supports elasticity and service availability.

9.2. Reinforcement Learning at the Edge

Reinforcement learning (RL) provides a powerful way to make AI systems adaptive and self-improving in dynamic environments. For service providers, RL can significantly enhance automation, resilience, and performance across edge-hosted applications. By enabling AI agents to learn from real-time feedback, RL supports intelligent decision-making in scenarios like network optimization, anomaly detection, and field operations.

A key technique is reinforcement learning from human feedback (RLHF), originally used to align LLMs like ChatGPT with human preferences [27]. Adapted for edge use, RLHF can continuously refine models based on input from local users or technicians. For example, a troubleshooting assistant running on a headend server could learn from technician feedback to improve its responses over time. While heavy training might occur in centralized infrastructure, deploying feedback loops at the edge ensures alignment with localized workflows and conditions.

RL also enables autonomous agents to optimize real-world systems. In cable operations, an RL-based agent could adjust node tilt or amplifier settings to maintain optimal signal-to-noise ratio (SNR), using network KPIs as a reward signal. Over time, such models learn to proactively tune plant configurations, potentially reducing outages and improving QoE. Hosting these models at the edge enables real-time control loops, minimizing reaction time and dependency on cloud backhaul.

Reinforcement learning transforms edge AI from static inference engines into adaptive, autonomous systems. For service providers, this enables more intelligent networks and operations that improve continuously over time.

9.3. Federated Learning and Decentralized AI

Federated learning (FL) is an emerging paradigm that allows AI models to be trained across distributed devices while keeping data local. This approach is particularly relevant for edge AI, where data privacy and bandwidth constraints limit centralized training. By aggregating model updates from multiple edge nodes, federated learning enables collaborative model improvement without exposing sensitive data.

Another evolving area is **distributed reinforcement learning (RL)** applied to edge AI. This includes scenarios where multiple edge agents learn policies concurrently and share updates periodically. For operators, distributed RL can be used to manage dynamic resources (e.g., spectrum, power), optimize plant configurations, or automate anomaly mitigation. Continued evaluation of decentralized learning algorithms and federated policy sharing is crucial to unlocking adaptive and self-optimizing network behaviors [42].

This domain is evolving rapidly. Significant breakthroughs in hardware acceleration, model optimization, and inference orchestration are occurring multiple times per year. Examples include the rise of sparse expert models, transformer optimizations such as FlashAttention, and hardware like AMD Instinct accelerators or Habana Gaudi, which enable better performance-per-watt at the edge. Operators must adopt agile development and evaluation strategies to keep pace with these changes. Iterative testing, modular deployment architectures, and model-agnostic APIs (e.g., OpenAI API compatibility) are key to maintaining long-term flexibility.

10. Conclusions

As edge AI continues to mature, it represents a strategic inflection point for service providers. By hosting AI models closer to end-users and within their own network infrastructure, operators can deliver low-latency, privacy-preserving services that were previously only feasible through centralized cloud deployments. This shift enables a range of benefits, from enhanced operational efficiency through AI-assisted network management, to the development of innovative, real-time customer applications.

The paper outlined how operators can leverage edge AI across various deployment models, including customer premises equipment (CPE), headends, regional data centers, and towers. Each location offers unique trade-offs in terms of latency, bandwidth, and resource availability. Operators must align deployment strategies with specific business needs, infrastructure capabilities, and service goals. Hosting models, including Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS), provide flexible paths to monetize edge compute infrastructure and foster third-party innovation.

The use of large language models (LLMs), vision-language models (VLMs), and task-specific machine learning models was examined in detail. Fine-tuning, retrieval-augmented generation (RAG), and reinforcement learning were identified as essential methods to enhance model performance and adapt models to local environments. Techniques for multi-model hosting, distributed inference, and orchestration were also discussed, highlighting the importance of scalable and flexible edge compute infrastructure.

Edge AI is evolving rapidly, driven by advances in hardware, model optimization, and orchestration tools. Operators are encouraged to adopt agile development and evaluation methodologies to remain competitive and responsive to technological change. By investing in flexible, scalable edge platforms, service providers can future-proof their AI capabilities and unlock new opportunities across both operational and customer-facing domains.

Edge AI offers a powerful mechanism for service providers to differentiate themselves, improve service quality, and enable new business models. Those who embrace and lead in edge AI adoption will be well positioned to shape the next generation of intelligent, real-time digital services.

Abbreviations

5G	5th Generation
AI	Artificial Intelligence
AMD	Advanced Micro Devices
API	Application Programming Interface
ASR	Automatic Speech Recognition
AWQ	Activation-aware Weight Quantization
BF16	Bfloat16 floating-point format
BLIP	Bootstrapping Language-Image Pre-training
CAMARA	Common Access to Mobile Network Resources and APIs
CDN	Content Delivery Network
CLI	Command-Line Interface
CMTS	Cable Modem Termination System
CPE	Customer Premises Equipment
CV	Computer Vision
DAA	Distributed Access Architecture
DAGs	Directed Acyclic Graphs
DGX	Deep Learning and AI Platform from NVIDIA
DSLAMs	Digital Subscriber Line Access Multiplexers
ETSI	European Telecommunications Standards Institute
FAISS	Facebook AI Similarity Search
FP16	Half-precision floating-point format
FTTP	Fiber to the Premises
FWA	Fixed Wireless Access
GDPR	General Data Protection Regulation
GGUF	GPT-Generated Unified Format
GPON	Gigabit Passive Optical Network
GPTQ	Generative Pre-trained Transformer Quantization
GPU	Graphics Processing Unit
gRPC	Google Remote Procedure Call
HIPAA	Health Insurance Portability and Accountability Act
HPA	Horizontal Pod Autoscaler
IaaS	Infrastructure as a Service
IAM	Identity and Access Management
IB	InfiniBand
INT8	8-bit integer
IPU	Intelligence Processing Unit

KPA	Knative Pod Autoscaler
KV-cache	Key-Value cache
LLD	Low Latency DOCSIS
LLM	Large Language Model
LoRA	Low-Rank Adaptation
MEC	Multi-access Edge Computing
MLPerf	Machine Learning Performance benchmark suite
MSO	Multiple-System Operator
mTLS	mutual Transport Layer Security
NLP	Natural Language Processing
NPU	Neural Processing Unit
NVMe	Non-Volatile Memory Express
OLTs	Optical Line Terminals
ONTs	Optical Network Terminals
OTA	Over-the-Air
PaaS	Platform as a Service
PON	Passive Optical Network
POP	Point of Presence
QLoRA	Quantized Low-Rank Adaptation
QoS	Quality of Service
RAG	Retrieval-Augmented Generation
RAN	Radio Access Network
RDK	Reference Design Kit
RF	Radio Frequency
RL	Reinforcement Learning
RLHF	Reinforcement Learning from Human Feedback
ROCm	Radeon Open Compute platform
ROI	Return on Investment
RPDs	Remote PHY Devices
RTT	Round-Trip Time
S3	Simple Storage Service
SaaS	Software as a Service
SCTE	Society of Cable Telecommunications Engineers
SIMD	Single Instruction, Multiple Data
SLAs	Service Level Agreements
SNR	Signal-to-Noise Ratio
SoCs	Systems on a Chip
TCO	Total Cost of Ownership
TTFT	Time-To-First-Token
URLLC	Ultra-Reliable Low-Latency Communications
VLM	Vision-Language Model
VRAM	Video RAM
XGS-PON	10 Gigabit Symmetrical Passive Optical Network

Bibliography & References

[1] ETSI MEC, "Multi-access Edge Computing (MEC); Framework and Reference Architecture," ETSI, 2023. [Online]. Available: https://www.etsi.org/technologies/multi-access-edge-computing

[2] CAMARA Project, "Edge Cloud APIs," Linux Foundation, 2024. [Online]. Available: https://lf-camaraproject.atlassian.net/wiki/spaces/CAM/pages/86573347/Edge+Cloud

[3] Diana Linton, Esteban Sandino, Nader Foroughi, Keith Auzenne, Premton Bogaj, "Exploring the Benefits of Network Intelligence Applications to Optimize HFC Networks Using a Data-Driven Design Approach," 2023. [Online]. Available: https://account.scte.org/documents/6370/3585_Linton_5314_paper.pdf

[4] Sharshar et al., "Vision-Language Models for Edge Networks: A Comprehensive Survey," 2025. [Online]. Available: https://arxiv.org/pdf/2502.07855

[5] ARK Invest, "PaaS Goldilocks: IaaS vs SaaS," [Online]. Available: https://www.ark-invest.com/articles/analyst-research/paas-goldilocks-iaas-saas

[6] Tech and Science Post, "How Amazon Web Services Makes Money: Estimated Margins by Service," 2021. [Online]. Available: https://techandsciencepost.com/news/companies/how-amazon-web-services-makes-money-estimated-margins-by-service/

[7] Omkar Dharmadhikari, "Moving Beyond Cloud Computing to Edge Computing," 2019. [Online]. Available: https://www.cablelabs.com/blog/moving-beyond-cloud-computing-to-edge-computing

[8] CableLabs, "Distributed Access Architecture (DAA) Overview," 2022. [Online]. Available: https://www.cablelabs.com/blog/distributed-access-architecture

[9] Broadband Library, "Creating Smarter HFC Networks with AI/ML Technologies," [Online]. Available: https://broadbandlibrary.com/creating-smarter-hfc-networks-with-ai-ml-technologies

[10] W. Kwon, S. Y. Feng, H. Zhang, et al., "Efficient Memory Management for Large Language Model Serving with PagedAttention," **SOSP**, 2023. [Online]. Available: https://arxiv.org/abs/2309.06180

[11] vLLM Team, "Welcome to vLLM (docs)," v0.10.0, 2025. [Online]. Available: https://docs.vllm.ai/

[12] NVIDIA, "TensorRT-LLM Documentation," 2026. [Online]. Available: https://docs.nvidia.com/tensorrt-llm/

[13] SGLang Project, "SGLang Documentation: Features and Backend Runtime (RadixAttention, Speculative Decoding, PD Disaggregation)," 2026. [Online]. Available: https://docs.sglang.ai/

[14] ggml-org, "llama.cpp - LLM inference in C/C++," GitHub repository, 2026. [Online]. Available: https://github.com/ggml-org/llama.cpp

[15] KServe, [Online]. Available: https://kserve.github.io/website/master/

[16] Knative, "About Autoscaling (KPA)," 2025. [Online]. Available: https://knative.dev/docs/serving/autoscaling/

[17] Ollama, [Online]. Available: https://ollama.com/

[18] Canoga Perkins, "The Importance of Low Latency in 5G Networks: A Deep Dive into URLLC," [Online]. Available: https://www.canogaperkins.net/post/the-importance-of-low-latency-in-5g-networks-a-deep-dive-into-urllc

[19] CableLabs, "Low Latency DOCSIS," [Online]. Available: https://www.cablelabs.com/technologies/low-latency-docsis

[20] Broadband Forum, "Network Quality: Over Bandwidth, Speed is Not Enough," [Online]. Available: https://www.broadband-forum.org/news/2022-10-28-network-quality-over-bandwidth-speed-is-not-enough/

[21] Touvron, H., et al. "LLaMA: Open and Efficient Foundation Language Models," Meta AI, 2023. [Online]. Available: https://arxiv.org/abs/2302.13971

[22] Dettmers, T., et al. "QLoRA: Efficient Finetuning of Quantized LLMs," 2023. [Online]. Available: https://arxiv.org/abs/2305.14314

[23] Li, J., et al. "BLIP-2: Bootstrapping Language-Image Pre-training with Frozen Image Encoders and Large Language Models," 2023. [Online]. Available: https://arxiv.org/abs/2301.12597

[24] Training Machine Learning models at the Edge: A Survey https://arxiv.org/html/2403.02619v1

[25] Hu, E. J., et al. "LoRA: Low-Rank Adaptation of Large Language Models," 2021. [Online]. Available: https://arxiv.org/abs/2106.09685

[26] Benaim, S., et al. "Video Diffusion Models," 2023. [Online]. Available: https://arxiv.org/abs/2304.08818

[27] Christiano, P. et al., "Deep reinforcement learning from human preferences," **Advances in Neural Information Processing Systems**, vol. 30, 2017. [Online]. Available: https://arxiv.org/abs/1706.03741

[28] P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in **Advances in Neural Information Processing Systems**, 2020. [Online]. Available: https://arxiv.org/abs/2005.11401

[29] Chi Sun, et al., "Challenging Common Assumptions in the Unsupervised Learning of Disentangled Representations," 2020. [Online]. Available: https://arxiv.org/abs/1905.05583

[30] FAISS: Facebook AI Similarity Search. [Online]. Available: https://github.com/facebookresearch/faiss

[31] Society of Cable Telecommunications Engineers (SCTE), "SCTE 280 2022 Understanding and Troubleshooting Cable RF Spectrum", 2022. [Online]. Available: /https://wagtail-prod-storage.s3.amazonaws.com/documents/SCTE_280_2022.pdf

[32] Microsoft, "DeepSpeed: A Deep Learning Optimization Library," 2023. [Online]. Available: https://www.microsoft.com/en-us/research/project/deepspeed

[33] Hugging Face, "Accelerate: Training and Inference at Scale," 2024. [Online]. Available: https://huggingface.co/docs/accelerate

[34] KServe, "Model Serving on Kubernetes," 2024. [Online]. Available: https://kserve.github.io/website/

[35] Distributed Deep Reinforcement Learning: An Overview. [Online]. Available: https://arxiv.org/abs/2011.11012

[36] KServe, "Canary and Blue-Green Rollouts," [Online]. Available: https://kserve.github.io/website/master/modelserving/v1beta1/rollout/canary-example/

[37] Sundaresan, "Edge Intelligence: Enabling Distributed ML Applications in Cable Networks", SCTE TechExpo 2024, https://account.scte.org/documents/7622/AI01_Sundaresan_6686_paper.pdf

[38] Volpe, "Embracing Proactive Network Maintenance with AI," Broadband Library, [Online]. Available: https://broadbandlibrary.com/embracing-proactive-network-maintenance-with-ai

[39] ABDELLI, et al., "Machine Learning-based Anomaly Detection in Optical Fiber Monitoring," [Online]. Available: https://arxiv.org/pdf/2204.07059

[40] Wang, et al., "Optimizing Edge AI: A Comprehensive Survey on Data, Model, and System Strategies," [Online]. Available: https://arxiv.org/html/2501.03265v1

[41] Digital Realty, "Data Gravity Index (DGx)TM 2.0," [Online]. Available: https://go2.digitalrealty.com/rs/087-YZJ-646/images/Report_Digital_Realty070523__Data_Gravity_Index_DGx_2.0.pdf

[42] Comcast Business, "Key Network Considerations for Supporting Enterprise AI Applications," [Online]. Available: https://business.comcast.com/community/browse-all/details/key-network-considerations-for-supporting-enterprise-ai-applications

[43] IBM, "What are small language models?," [Online]. Available: https://www.ibm.com/think/topics/small-language-models

[44] Microsoft, "What Are Small Language Models (SLMs)?," [Online]. Available: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-are-small-language-models

[45] Comcast, "Comcast Edge Compute to Enhance Streaming Experience with Standardized Platform and Content Delivery Network," [Online]. Available: https://corporate.comcast.com/press/releases/comcast-edge-compute-enhance-streaming-experience-standardized-platform-content-delivery-network

Appendix A: Platforms

11. Edge LLM Hosting Platforms

11.1. Backends

11.1.1. vLLM

vLLM is a high-throughput LLM engine with **PagedAttention** and **continuous batching**. Suited for multi-request concurrency, long contexts, and general-purpose chat/completion at the edge. [10], [11]

Key Capabilities

- Paged KV cache; request interleaving across prefill/decode.
- Tensor/pipeline parallel options; OpenAI-style API compatibility via server.

Performance Considerations (what we observed)

- Strong throughput at moderate/high concurrency; stable p95 with mixed prompt lengths.
- Cold starts dominated by weight load; benefits from warmers and pre-pull.

Strengths

- Mature scheduler; broad model support; thriving ecosystem.

Limitations / Gaps

- Disaggregated prefill/decode is typically handled **outside** vLLM (via platform like Dynamo) if required.

11.1.2. TensorRT-LLM

NVIDIA-optimized inference stack generating highly tuned kernels and execution graphs. Best where **lowest latency** and **GPU efficiency** on NVIDIA hardware are paramount. [12]

Key Capabilities

- Graph-level optimizations, FP8/INT8/INT4 paths, plugin kernels; multi-GPU parallelism.

Performance Considerations (what we observed)

- Top-tier per-GPU latency/throughput when calibrated; payoff increases with large models.

Strengths

- Hardware-tailored performance; good fit for **core/metro edge** with consistent GPUs.

Limitations / Gaps

- Build/compile complexity; model coverage and portability lag OSS engines for some use cases.

11.1.3. SGLang

Fast serving engine emphasizing **RadixAttention** (prefix sharing), **continuous batching**, **paged attention**, and **speculative decoding**. Excels for **multi-turn** and **template-heavy** workloads with shared prefixes. [13]

Key Capabilities

- Built-in prefill/decode disaggregation; structured outputs; multi-LoRA batching; parallelism strategies.

Performance Considerations (what we observed)

- Gains grow with prefix reuse and branchy dialogs; speculative decoding reduces tail latency when tuned.

Strengths

- Efficient cache reuse; flexible features for assistants and tools.

Limitations / Gaps

- Rapidly evolving surface area; pin releases and validate attention backends by GPU gen.

11.1.4. llama.cpp

Lightweight C/C++ engine for **CPU/GPU-constrained** nodes with aggressive quantization (GGUF). Ideal for **single-host** edge, offline, or embedded scenarios. [14]

Key Capabilities

- Layer offload control, wide quantization set, minimal deps; sample server/CLI.

Performance Considerations (what we observed)

- Competitive for small/medium models at low concurrency; falls behind at high concurrency without continuous batching.

Strengths

- Portability, tiny footprint, runs without GPUs.

Limitations / Gaps

- Lacks production features (scheduler/tenancy/metrics); rely on a wrapper.

11.1.5. Ray / Ray Serve

Runtime substrate for Python Directed Acyclic Graphs (DAGs) and services; **Ray Serve** adds model-serving primitives (routing, replicas, autoscaling). We list it as a backend runtime because teams often embed engines (vLLM/TensorRT-LLM/SGLang) **inside** Ray workers to scale horizontally. [32], [33]

Key Capabilities

- Actor-based scaling across nodes; HTTP ingress; can co-locate GPU workers.

Performance Considerations (what we observed)

- Good for composing multi-stage pipelines (retrieval, scoring, generation); overhead acceptable when replicas are sized well.

Strengths

- Flexible graphs; built-in autoscaling at the worker/replica level.

Limitations / Gaps

- Overlaps with orchestration features; requires additional ops layers for tenancy, rollouts, and strict SLOs.

11.2. Orchestration

11.2.1. Ollama

Role

Lightweight **single-node** serving platform wrapping llama.cpp with simple model management and REST. Best for **low-volume** edge and labs. [17]

Why pick it

- Fast time-to-first-deployment; easy model pulls/tags; CPU-only possible.

Mind the gaps

- No dynamic batching, multi-GPU, or multi-node scaling. Put behind a gateway for auth/rate-limit/metrics.

Backends commonly paired

- **llama.cpp** (native). Can front small GGUF models effectively.

11.2.2. KServe

Role

Kubernetes-native serving with **Knative** autoscaling, canary rollouts, and observability. Runs arbitrary backends via containers (vLLM). [15], [16]

Why pick it

- Strong **SRE posture** (HPA/KPA, scale-to-zero, revisions); integrates with cluster policies, gateways, and GitOps.

Mind the gaps

- Requires Kubernetes maturity; per-pod GPU binding; startup time for large images/weights.

Backends commonly paired

- **vLLM** (default choice)

11.2.3. NVIDIA Dynamo

Role

Distributed inference **orchestration** with **aggregated** or **disaggregated** prefill/decode modes and optional routing layer. Engine-agnostic: vLLM, TensorRT-LLM, **SGLang**. [13]-[15]

Why pick it

- Low-latency at scale; separates concerns (routing, prefill, decode); leverages high-bandwidth interconnects.

Mind the gaps

- Younger project; pair with gateway/observability for auth/tenancy/quotas.

Backends commonly paired

- **SGLang** (great for prefix sharing + speculative), **TensorRT-LLM** (raw perf), **vLLM** (balanced throughput) - aggregated or disaggregated per need.