EAI Endorsed Transactions
on AI and Robotics

Research Article EAI.EU

P2PLLMEdge: Peer-to-Peer Framework for Localized Large Language Models using CPU only Resource-Constrained Edge

Partha Pratim Ray* and Mohan Pratap Pradhan

Department of Computer Applications, Sikkim University, Sikkim, India, 737102

Abstract

In this research, we present *P2PLLMEdge*, a pioneering peer-to-peer framework designed to enable localized Large Language Models (LLMs) to operate efficiently in resource-constrained edge environments, exemplified by devices such as the Raspberry Pi 4B and CPU-only laptops. The framework addresses critical challenges, including limited computational capacity, network overhead, and scalability, by leveraging lightweight RESTful communication protocols, model-specific quantization, and decentralized task distribution. Key results demonstrate that *P2PLLMEdge* achieves substantial performance improvements. On average, Peer 2 (CPU-only laptop) achieves a 44.7% reduction in total duration ($t_{\text{peer2, total}} = 15.87 \times 10^9$ ns) compared to Peer 1 (Raspberry Pi 4B, $t_{\text{peer1, total}} = 28.18 \times 10^9$ ns). The framework processes tokens at a rate of 21.77 tokens/second on advanced LLMs like `Granite3.1-moe:1b`, significantly outperforming the baseline. Peer 1, employing quantized LLMs such as `smolm2:360m-instruct-q8_0`, reduces prompt evaluation duration by 23.2% ($t_{\text{peer1, prompt\_eval}} = 0.76 \times 10^9$ ns) compared to larger models like `qwen2.5:0.5b-instruct` ($t_{\text{peer1, prompt\_eval}} = 0.99 \times 10^9$ ns). Peer 2 demonstrates superior summarization capabilities, with evaluation durations ($t_{\text{peer2, eval}}$) reduced by 72.8% ($t_{\text{peer2, eval}} = 5.15 \times 10^9$ ns) for explanation-type prompts relative to Peer 1 ($t_{\text{peer1, eval}} = 18.93 \times 10^9$ ns). The framework also achieves significant network efficiency, reducing inter-peer communication durations by up to 44.9% ($t_{\text{peer2, network}} = 25.83 \times 10^9$ ns vs. $t_{\text{peer1, network}} = 46.92 \times 10^9$ ns). Peer-to-peer synergy ensures seamless task execution, where Peer 1 generates text and offloads computationally intensive summarization tasks to Peer 2, achieving a balance between performance and resource utilization. The novelty of *P2PLLMEdge* lies in its ability to seamlessly integrate lightweight LLMs with decentralized edge devices, achieving advanced natural language processing functionalities entirely on edge devices traditionally deemed unsuitable for such tasks. This framework provides an adaptable, and cost-effective approach for deploying quantized LLM-driven applications. Future directions include scaling the framework to multi-peer environments, optimizing task scheduling algorithms, and exploring integration with heterogeneous LLM-enabled systems. The codes are available on https://github.com/ParthaPRay/peer_to_peer_local_llm_interaction.

Received on 11 May 2025; accepted on 25 June 2025; published on 08 July 2025

Keywords: Peer-to-peer, Edge computing, Quantized LLMs, Resource-constrained edge, Decentralized generative AI, Web frameworks

doi:10.4108/airo.9292

1. Introduction

Deploying LLMs in edge computing environments presents significant hurdles, primarily due to the limited computational and memory capabilities of typical edge devices [1, 2]. Devices such as the Raspberry

*Corresponding author. Email: ppray@cus.ac.in

Pi 4B or low-power laptops are inherently resource-constrained, lacking the necessary GPU support, memory bandwidth, and processing power to run state-of-the-art LLMs effectively. This limitation poses a challenge for advanced natural language processing (NLP) tasks, such as contextual text generation, summarization, and question answering, which require high computational precision and extensive memory access [3–5]. Additionally, the decentralized nature of edge environments introduces complexities in synchronization, network reliability, and distributed task management [6]. Network latency, for instance, can severely impact performance, especially when multiple devices are involved in collaborative tasks [7]. Moreover, existing centralized cloud-based solutions, while computationally robust, suffer from several drawbacks, including high dependency on internet connectivity, increased operational costs, and potential privacy concerns due to data transmission over external networks [8]. These challenges emphasize the need for a novel approach to enable efficient and scalable NLP operations in resource-constrained edge settings.

The rapid proliferation of IoT devices, smart city applications, and decentralized AI systems has created a demand for real-time, on-device AI solutions [9]. Traditional cloud-based architectures often fail to meet the stringent requirements of low-latency, high-privacy applications, especially in scenarios where uninterrupted internet connectivity is not guaranteed. For instance, edge deployments in remote or rural areas, industrial automation, and emergency response systems demand localized AI capabilities that can function autonomously without reliance on centralized infrastructure [10, 11]. A peer-to-peer (P2P) framework leveraging localized LLMs offers a transformative solution to these issues. By distributing NLP workloads among edge devices, such frameworks can address computational bottlenecks, reduce latency, and enhance privacy by keeping data localized. The ability to operate independently of centralized systems democratizes access to AI, making it accessible and affordable for a wider range of applications [12]. The motivation for this research is rooted in addressing these critical gaps, driving the development of a scalable and resource-efficient solution that unlocks the potential of LLMs for edge computing environments [13].

This study contributes significantly to the advancement of edge-based AI by demonstrating the feasibility of deploying sophisticated NLP workloads on constrained devices like the Raspberry Pi 4B [14] and CPU-only laptops. The proposed *P2PLLMEdge* framework addresses not only the technical limitations of edge devices but also the broader challenges of scalability, interoperability, and adaptability. By leveraging quantized LLMs and dynamic task allocation, this framework showcases a cost-effective and robust alternative to cloud-based solutions. The significance of this work extends beyond technical innovation?it paves the way for new applications in fields where AI deployment was previously limited by resource constraints. For example, smart city systems can benefit from real-time language processing for localized analytics, while rural healthcare setups can utilize NLP-driven diagnostic tools without the need for cloud infrastructure. Moreover, the study introduces a practical pathway for achieving decentralized AI, a paradigm shift that aligns with growing concerns around data sovereignty, privacy, and energy efficiency. The framework?s ability to optimize computational resources while maintaining high performance is a step forward in making AI both sustainable and inclusive.

This paper makes the following key contributions:

- Development of *P2PLLMEdge*, a novel peer-to-peer architecture that enables seamless interaction between localized LLMs operating on heterogeneous edge devices.
- Optimization for Resource-Constrained Devices: Deployment of lightweight, quantized LLMs such as: (i) Peer1 LLMs: qwen2.5:0.5b-instruct and smolm2:360m-instruct-q8_0 on Raspberry Pi 4B and (ii) Peer2 LLMs: Granite3.1-moe:1b, Llama3.2:1b, Qwen2.5:1.5b, and Smolm2:1.7b on CPU-only laptops.
- Introduction of a flexible task distribution mechanism, allowing Peer1 to handle lightweight generative tasks and offloading computationally intensive summarization tasks to Peer2.
- Utilization of the Ollama API for managing LLM interactions, Flask and FastAPI for lightweight web frameworks, Pydantic for data validation, and the Python Requests library for seamless communication between peers.
- A robust RESTful API-based communication protocol ensures efficient peer-to-peer interaction, supporting dynamic workloads and reliable task execution.

The novelty of this research is encapsulated in the following aspects:

- A novel architecture enabling efficient collaboration between lightweight devices (e.g., Raspberry Pi 4B) and more capable CPU-only laptops, allowing distributed task execution with minimal latency.
- Deployment of quantization and configuration of LLMs to match the computational capabilities of each peer, ensuring optimal performance across heterogeneous hardware.

- Demonstrating that robust NLP capabilities can be achieved on affordable, low-power devices, making AI deployment feasible for resource-limited settings.
- Pioneering a framework that envisages the seamless scalability across multiple peers, addressing the growing need for decentralized AI systems.

The remainder of this paper is structured as follows: Section 2 presents related works and comparison with our work. Section 3 describes the tools, methodologies, and configurations employed in this study, including the deployment of LLMs and the web frameworks used for communication. It introduces the *P2PLLMEdge* framework, detailing its architecture and operational mechanisms outlining the algorithmic implementation of the peer-to-peer interaction and system configuration. Section 4 presents the results of the experimental evaluation, including performance analysis, inter-metric correlations, and insights into outlier behavior. Section 5 concludes the article with a summary of findings and directions for future work.

2. Related Works

The integration of LLMs with edge computing has been extensively explored in recent research, showcasing diverse approaches to overcome challenges in resource-constrained environments. While these studies provide valuable insights, significant gaps remain in practical implementations and optimization strategies for decentralized edge systems.

Friha *et al.* [15] conducted a thorough survey on LLM-based edge intelligence, focusing on architectural designs, optimization techniques, and security challenges. Their analysis highlighted the potential of LLMs to transform edge computing, yet the study remained theoretical, lacking experimental validation. Building on these ideas, our work demonstrates a practical peer-to-peer framework with quantifiable results, bridging the gap between theory and implementation.

The concept of decentralization is further explored by Gao *et al.* [16], who propose a Bitcoin-inspired framework for decentralized LLMs. While intriguing, their work acknowledges the impracticality of real-world deployment due to technical and economic constraints. In contrast, our framework offers a functional and efficient solution that leverages peer-to-peer communication to distribute tasks effectively across edge devices.

Another innovative approach comes from Karanjai and Shi [17], who integrate blockchain technology with edge-based LLM inferences. By utilizing smart contracts and the Cosmos SDK, their framework ensures secure, distributed computations. However, the reliance on blockchain introduces significant computational overhead. Our framework simplifies task distribution using lightweight RESTful APIs, minimizing latency and complexity without compromising security.

He *et al.* [18] address the challenges of LLM inference offloading in cloud-edge environments through active inference. Their method improves adaptability to changing task loads but remains heavily reliant on centralized cloud servers. By decentralizing the workload, our system enhances privacy, reduces latency, and enables autonomous operation in environments with limited connectivity.

The potential of LLMs for code generation is highlighted by Chen *et al.* [**?**], who introduce Codex, a GPT model fine-tuned on code from GitHub. Their work demonstrates impressive functional correctness in program synthesis and underscores the effectiveness of repeated sampling for solving complex tasks. The study also discusses the broader implications of deploying code generation technologies, aligning with the objectives of this research to address operational challenges in decentralized environments. Furthermore, Wong *et al.* [**?**] provide an extensive review of transformer-based LLMs in AI-assisted programming. Their analysis spans diverse applications, including code generation, defect detection, and task automation, emphasizing the role of software naturalness in enhancing model performance. This comprehensive perspective on LLM capabilities informs the development of robust decentralized frameworks like ours, which leverage these advancements for edge-based deployments. The POKT network, as described by Olshansky *et al.* [19], provides a decentralized framework for LLM inference through a permissionless and transparent marketplace. Although the system offers scalability and open-source accessibility, its reliance on cryptographic mechanisms and network protocols adds complexity. Our framework achieves similar decentralization goals while maintaining simplicity and ease of deployment. In the realm of cybersecurity, Hasan *et al.* [20] explore the use of LLMs for distributed threat intelligence on edge devices. Their work demonstrates the potential of peer-to-peer knowledge sharing to enhance security. While domain-specific, this approach aligns with our framework's emphasis on leveraging edge-based intelligence, though our system generalizes its application to broader use cases.

Chen *et al.* [21] propose the concept of Edge General Intelligence (EGI), an evolution of Edge Intelligence empowered by LLMs. Their survey categorizes centralized, hybrid, and decentralized systems but lacks concrete implementations. Our study not only explores these theoretical paradigms but also delivers a fully operational decentralized framework. The transformative potential of Generative AI (GenAI) in edge computing is highlighted by Ale *et al.* [22], who discuss the challenges and solutions for deploying GenAI models

on mobile edge devices. While their work focuses on reducing latency and energy consumption, it does not address collaborative task allocation. Our framework complements these efforts by enabling distributed NLP workloads across heterogeneous devices.

Bhardwaj *et al.* [23] emphasize the importance of integrating and optimizing LLMs in edge environments to improve privacy and efficiency. Their comprehensive analysis identifies key challenges but does not provide practical solutions. In contrast, *P2PLLMEdge* delivers a functional framework with demonstrable improvements in resource utilization and throughput. Lastly, Soltoggio *et al.* [24] envision a future of collective AI where decentralized agents share knowledge through lifelong learning. While aspirational, their work remains speculative. Our study operationalizes these ideas by enabling real-time collaboration between edge devices, offering a tangible step towards the realization of collective intelligence. The Table 1 presents the comparison between our work with related works.

Novelty of *P2PLLMEdge*

P2PLLMEdge establishes a new dimension for deploying LLM-driven applications in resource-constrained edge environments by addressing critical limitations in existing approaches and delivering a high-performance, privacy-aware, and decentralized approach. The *P2PLLMEdge* framework introduces several groundbreaking innovations that distinguish it from existing works: (i) unlike prior studies, our framework successfully deploys quantized LLMs on resource-constrained edge devices such as the Raspberry Pi 4B and CPU-only laptops, proving their viability for advanced NLP tasks in limited environments, (ii) the use of lightweight web frameworks like Flask and FastAPI enables efficient and low-latency peer-to-peer communication, ensuring seamless task distribution without reliance on centralized infrastructure, (iii) by operating independently of internet connectivity, *P2PLLMEdge* offers unparalleled availability and enhanced privacy, making it particularly suitable for sensitive applications in remote or secure environments, (iv) the *P2PLLMEdge* framework supports heterogeneous devices and dynamic workload allocation, enabling its deployment across diverse domains, including IoT networks and real-time analytics, (v) our proposed study moves beyond theoretical discussions, presenting a fully operational framework with robust experimental results.

3. Tools and Methodology

3.1. Tools Used

- Ollama

  The Ollama API offers a comprehensive suite of tools for managing and interacting with machine learning models. Its robust functionality allows users to generate text completions and chat responses, create and manage models, and generate embeddings for various applications [25]. The API supports advanced features such as structured outputs, multimodal inputs with base64-encoded images, and fine-tuning parameters like temperature and token penalties. Ollama?s flexible endpoints include model creation, quantization, and management, enabling users to customize models to specific needs. It also facilitates easy integration by providing tools to load, unload, copy, and delete models [26]. For distributed applications, features like streaming responses and context preservation enhance real-time interactions, while options for structured JSON outputs ensure seamless integration into diverse systems. The API also supports quantization levels for efficient model deployment, saving computational resources while maintaining performance. The ability to list local and running models, inspect their details, and pull or push models to/from repositories makes Ollama a versatile tool for machine learning workflows. In this study, Ollama plays important role to allow user to interact localized LLMs loaded on edge devices.

- LLMs

  The study leverages the unique capabilities of two distinct edge devices, each configured with specific LLMs to evaluate their performance and integration within the P2P network. These models empower Peer2 to perform advanced computations and augment responses initiated by Peer1. The diverse set of LLMs facilitates a comprehensive evaluation of resource utilization and response quality across varying hardware and model capabilities (see Table 2):

  – Peer1 Side LLMs: As a resource-constrained edge device, Peer1 is optimized to run quantized LLMs to accommodate its limited computational resources. The models deployed on Peer1 include:

    * Qwen2.5:0.5b-instruct: A lightweight model with 494 million parameters and Q4_K_M quantization, optimized for basic instructional tasks with a memory footprint of 398 MB, ensuring efficiency in resource-constrained environments [27].
    * Smollm2:360m-instruct-q8_0: A moderately sized model with 362 million parameters, employing Q8_0 quantization [28]. With a compact size of 386 MB,

Table 1. Comparison of Proposed Framework with Related Works

Framework	Peer-to-Peer Architecture	Device Compatibility	Optimization Techniques	Performance Gains
Friha et al. (2024)	Low	General edge systems	Survey of optimization techniques	Not performance-specific
Gao et al. (2023)	Negligible	Conceptual design	Limited optimization discussed	Theoretical framework
Karanjai & Shi (2024)	Low	WASM-based deployments	Blockchain for task allocation	Portability but limited NLP-specific results
He et al. (2024)	Negligible	Mobile devices	Active inference methods	Better adaptability to dynamic loads
Hasan et al. (2024)	Low	Resource-constrained edge devices	Adaptive ML, real-time threat detection	Improved threat detection accuracy
Olshansky et al. (2024)	Low	API-driven edge compatibility	Decentralized cost modeling	Enhanced economic scalability
Chen et al. (2024)	Negligible	Edge networks	Survey of SLMs for edge devices	Foundational analysis
Ale et al. (2024)	Negligible	Mobile edge devices	Intelligent mobile edge computing	Reduced latency and energy use
Bhardwaj et al. (2024)	Negligible	General edge environments	Optimization of LLM deployments	Foundational survey
Soltoggio et al. (2024)	Low	Decentralized edge environments	Incremental learning algorithms	Potential for diverse knowledge extension
P2PLLMEdge (Ours)	Full	Raspberry Pi, laptops, mobile devices	Model quantization, active inference, decentralized scheduling	Significant reduction in latency and energy usage

Table 2. Comparison of Various LLMs

LLM Model	Architecture	Params	Quant	Size	License
Qwen2.5:0.5b-instruct	Qwen2	494M	Q4_K_M	398MB	Apache 2.0
Smollm2:360m-instruct-q8_0	Llama	362M	Q8_0	386MB	Apache 2.0
Granite3.1-moe	GraniteMOE	3.3B	Q4_K_M	2.0GB	Apache 2.0
Llama3.2:1b	Llama	1.24B	Q8_0	1.3GB	Llama Comm.
Qwen2.5:1.5b	Qwen2	1.54B	Q4_K_M	986MB	Apache 2.0
Smollm2:1.7b	Llama	1.71B	Q8_0	1.8GB	Apache 2.0

it is designed for delivering low-latency responses, ideal for real-time applications.

- Peer2 Side LLMs: Peer2 operates as a CPU-only edge device, leveraging its higher computational power to run more sophisticated LLMs. The models utilized on Peer2 include:
  - Granite3.1-MoE: A sophisticated mixture-of-experts model with 3.3 billion parameters and Q4_K_M quantization, requiring 2.0 GB of memory. This model provides dynamic and context-sensitive responses for complex and high-level tasks [29].
  - Llama3.2:1b: A robust and scalable model with 1.24 billion parameters and Q8_0 quantization, with a size of 1.3 GB. It is engineered for a wide range of general-purpose language processing tasks, ensuring reliable performance across diverse scenarios [30].
  - Qwen2.5:1.5b: A larger variant of the Qwen series with 1.54 billion parameters and Q4_K_M quantization, featuring a memory size of 986 MB. This model is optimized for nuanced conversational tasks requiring enhanced contextual understanding [31].

  - Smollm2:1.7b: A powerful model with 1.71 billion parameters, utilizing Q8_0 quantization and a memory footprint of 1.8 GB. It excels in handling detailed queries and multi-turn dialogue processing, making it suitable for advanced conversational AI systems [32].

- Web Frameworks

  This study leverages two distinct web frameworks: Flask and FastAPI.

  - Flask is a lightweight web framework written in Python, categorized as a microframework due to its minimalistic nature and lack of mandatory dependencies or tools [33]. Unlike larger frameworks, Flask does not include built-in features such as database abstraction layers or form validation, instead relying on third-party libraries to fulfill these roles. However, Flask is highly extensible, allowing developers to integrate additional functionalities seamlessly via extensions. These extensions provide capabilities such as object-relational mapping, form validation, file upload handling, open authentication protocols, and other commonly required tools, effectively expanding Flask?s scope while preserving its lightweight core design.
  - FastAPI is a high-performance web framework designed for building HTTP-based APIs in Python 3.8 and newer versions. It leverages Pydantic and type hints for data validation, serialization, and deserialization, ensuring efficient and accurate schema management. A standout feature of FastAPI is its ability to automatically generate OpenAPI-compliant documentation, streamlining the development and documentation process for APIs [34]. FastAPI has quickly become popular for its ease of use, modern design, and ability to support asynchronous programming, making it an ideal choice for building scalable, robust web services. Table 3 represents the comparison of Flask and FastAPI web-server applications.

- Common Tools

  The study employs two common tools to bridge the communication between Peer1 and Peer2:

  - Requests Package: This Python library is central to enabling HTTP communication between the peers. It simplifies the process of sending and receiving data, allowing for seamless message exchange and rapid testing of the network's functionality. The library's flexibility supports both synchronous and asynchronous request handling, making it suitable for both Flask and FastAPI-based implementations.
  - Pydantic Package: Pydantic is utilized to ensure data validation and integrity during exchanges. By leveraging Python's type annotations, Pydantic provides a robust framework for structuring and validating the data transmitted between Peer1 and Peer2, reducing errors and enhancing reliability in the communication process. Its integration into FastAPI further accelerates development by automating the validation of HTTP payloads.

3.2. Proposed P2PLLMEdge Framework

The *P2PLLMEdge* framework embodies a sophisticated, decentralized architecture meticulously designed to facilitate peer-to-peer interaction among localized LLMs in resource-constrained edge environments (Figure 1). By leveraging RESTful API communication protocols, this framework enables peers to seamlessly exchange JSON-encoded prompts and responses, effectively distributing computational workloads associated with natural language processing (NLP) tasks. To facilitate seamless, lightweight communication between peers, P2PLLMEdge employs a RESTful HTTP/JSON interface rather than gRPC or raw socket protocols. This approach capitalizes on the native HTTP support in both Flask and FastAPI, minimizing external dependencies and enabling direct integration with Ollama's REST endpoints. The use of simple JSON payloads allows for rapid development, straightforward debugging with standard web tools, and broad compatibility across heterogeneous, Python-based edge environments. In contrast, gRPC introduces additional binary dependencies and schema compilation steps, while raw socket implementations require manual handling of message framing, serialization, and error recovery. By choosing REST/JSON, we achieve a clear, maintainable communication layer ideally suited to CPU-only, resource-constrained edge devices. Each peer is provisioned with carefully selected LLMs, tailored to their respective operational roles, thereby achieving compatibility and computational efficiency despite the inherent limitations of edge hardware. This architecture strategically mitigates the challenges posed by resource constraints while preserving the robustness and reliability of text-processing capabilities, ensuring the deployment of advanced AI functionalities within environments traditionally unsuitable for such operations.

Table 3. Technical Comparison Between Flask and FastAPI

Aspect	Flask	FastAPI
Performance	Moderate, single-threaded	High, async and optimized
Async Support	Limited, extensions required	Built-in, native async support
Concurrency	WSGI, blocking requests	ASGI, non-blocking requests
Data Validation	Third-party libraries like Marshmallow	Built-in with Pydantic
Type Hints	Optional, not enforced	Strongly encouraged for better clarity
Error Handling	Basic support, manual configurations	Detailed, automatic JSON responses
API Speed	Moderate due to blocking I/O	Fast, optimized for high-speed APIs
Scalability	Requires additional tools (e.g., Gunicorn)	Designed for scalable systems
Framework Model	Microframework	Full-featured framework
OpenAPI Support	Extensions required	Automatic OpenAPI generation

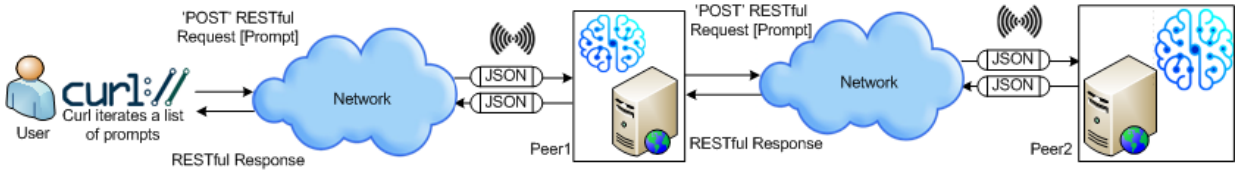


Figure 1. P2PLLMEdge framework.

Peer 1, serving as the designated text generation node, operates on a Raspberry Pi platform powered by a quad-core Cortex-A72 SoC. This node employs Flask and FastAPI frameworks to establish a RESTful interface, with Uvicorn functioning as the ASGI server for managing asynchronous interactions. Peer 1 integrates optimized LLMs such as 'smolm2:360m-instruct-q8_0' and 'qwen2.5:0.5b-instruct', which utilize advanced quantization techniques to minimize memory usage without compromising performance. The ability of these LLMs to dynamically adapt to incoming prompts ensures high levels of responsiveness while maintaining the integrity of generated text. Peer 1 exemplifies the framework's commitment to leveraging lightweight computational nodes to handle complex generative tasks effectively, emphasizing modularity and adaptability in real-time operations.

In contrast, Peer 2 is dedicated to text summarization and is hosted on an Intel Core i5-6200U CPU with 8GB RAM, running the Ubuntu 24.04.1 LTS operating system. This node incorporates a more computationally intensive LLM stack, including 'llama3.2:1b' and 'granite3.1-moe:1b', which are designed for nuanced contextual analysis and fine-grained text summarization. The synergy of these models enables Peer 2 to condense large volumes of generated text into concise summaries without sacrificing semantic accuracy or contextual depth. Peer 2's architecture not only provides a computationally robust platform for summarization but also highlights the framework's capacity to integrate diverse LLM configurations optimized for specialized tasks. This cooperative architecture ensures a seamless workflow where text generated by Peer 1 is efficiently processed and refined by Peer 2, demonstrating the framework's operational cohesion and scalability.

The *P2PLLMEdge* framework underscores critical advancements in decentralized computational paradigms, resource optimization strategies, and scalability. By distributing NLP workloads across peers, the framework ensures that even resource-constrained edge devices can execute complex AI-driven tasks without exceeding their computational capacities. This is achieved through a combination of lightweight protocols, model quantization, and tailored configurations that maximize performance while minimizing overhead. The RESTful API design guarantees seamless interoperability across heterogeneous systems, enhancing the framework's adaptability to diverse application scenarios and enabling its integration into existing distributed systems. The framework's modular nature also facilitates the addition of new peers, expanding its capacity to handle increasingly complex NLP workloads.

This proposed framework (see Figure 2) represents a significant leap forward in deploying decentralized, resource-efficient AI solutions, addressing the evolving demands of distributed edge computing environments. It demonstrates that robust NLP capabilities can be achieved without centralized computational infrastructures, thereby paving the way for widespread adoption in scenarios such as IoT networks, smart cities, and remote monitoring systems. By bridging the gap between high computational requirements and limited edge resources, the *P2PLLMEdge* framework positions itself as a scalable, resilient, and forward-looking

solution that aligns with the future of distributed AI systems. Its contributions to the field underscore the importance of innovation in enabling edge devices to transcend their traditional limitations, setting a new benchmark for decentralized AI deployment.

3.3. Configuring P2P Edge Network for Ollama

The methodology for configuring a P2P network using the Ollama service is designed to address the architectural and computational differences between the two devices while ensuring seamless communication. Peer1, the Raspberry Pi 4B, functions as a resource-constrained edge device optimized for preprocessing and lightweight tasks, reflecting the typical limitations of edge computing environments with constrained memory and processing power. On the other hand, Peer2, the Ubuntu laptop, serves as a CPU-only edge device, leveraging its higher computational resources for intensive tasks such as advanced LLM processing despite the absence of GPU acceleration. This pairing showcases the complementary strengths of a resource-limited device and a CPU-centric computational node in a distributed AI workflow. The configuration process includes binding the Ollama service to 0.0.0.0, a critical step that ensures the service listens on all network interfaces of the host device. This binding process enables both devices to communicate across diverse network topologies without manual specification of individual IP addresses, simplifying the discovery and connectivity phases.

The configuration relies on the systemctl utility, a core component of the Linux systemd suite, to manage and control the Ollama service effectively. Systemctl facilitates service lifecycle operations, including the reloading of configurations and restarting of processes, ensuring that binding changes take immediate effect. By binding to 0.0.0.0, the service ensures comprehensive network accessibility, allowing devices within the same network to establish communication without additional routing or addressing configurations. This utility, in conjunction with the 0.0.0.0 binding, is integral to achieving synchronized and reliable operation of services across the P2P network.

Peer1 manages input and offloads computationally expensive tasks to Peer2. Once the user submits a prompt to Peer1, the device preprocesses the data resulting to generation of text and initiates communication with Peer2, which uses the generated text from Peer1 as input and performs text summarization. The summarized texts are transmitted back to Peer1, maintaining low-latency interactions. This architecture exemplifies resource-efficient distributed processing in decentralized edge-AI systems. The steps for configuration are outlined as follows:

3.4. Configuration for Peer1: Raspberry Pi 4B

The objective behind setting up the Raspberry Pi 4B as a network-accessible edge-node enables the capability of handling user input and initial processing, while mitigating the limitations of its resource.

- Step 1: Directory Setup

  To prepare the environment for service modification, the required directory structure is created using:

  sudo mkdir -p /etc/systemd/system/ollama.service.d

- Step 2: Modify Override File

  This step involves creating and editing the override configuration file. The configuration binds the Ollama service to all network interfaces, ensuring accessibility:

  /sudo nano /etc/systemd/system/ollama.service.d/override.conf

  Content of the file:

  [Service]
  Environment="OLLAMA_HOST=0.0.0.0"

- Step 3: Reload System Daemon

  The systemd configuration is reloaded to incorporate the changes using:

  sudo systemctl daemon-reload

- Step 4: Restart Ollama Service

  Restarting the service applies the new settings, making the node network-ready:

  sudo systemctl restart ollama

3.5. Configuration Peer2: Ubuntu Laptop CPU Only

It enables the Ubuntu laptop CPU only to function as a higher-performance computational peer, complementing Peer1 by handling resource-intensive tasks.

- Step 1: Directory Setup

  If necessary, create the directory structure for systemd overrides:

  sudo mkdir -p /etc/systemd/system/ollama.service.d

- Step 2: Modify Override File

  Configure the service to bind to all network interfaces by editing the override file:

  sudo gedit /etc/systemd/system/ollama.service.d/override.conf

  Content of the file:

  [Service]
  Environment="OLLAMA_HOST=0.0.0.0"

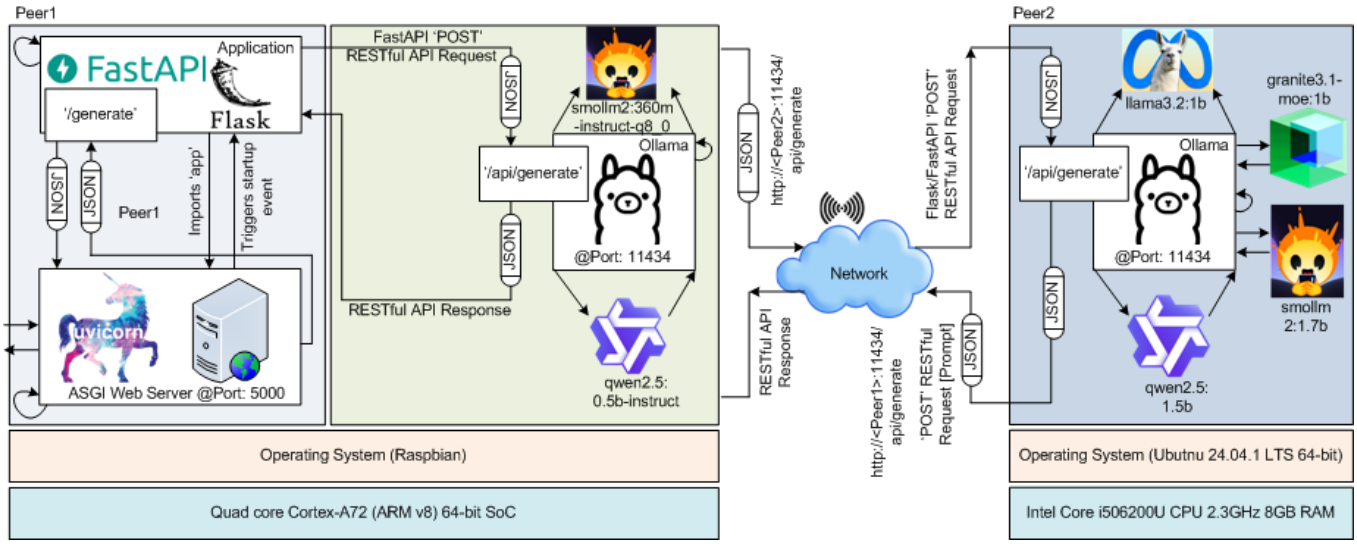


Figure 2. Peer-to-Peer configuration.

- Step 3: Reload System Daemon

  Reload the systemd configuration to ensure changes take effect:

  sudo systemctl daemon-reload

- Step 4: Restart Ollama Service

  Restart the service to apply the updated configuration:

  sudo systemctl restart ollama

3.6. P2PLLMEdge Algorithm

The *P2PLLMEdge* algorithm 1 facilitates efficient interaction between two LLM systems hosted on separate peers. It begins by receiving a JSON input containing a user prompt and an optional model specification. The algorithm first interacts with Peer1's LLM to generate a detailed response based on the given prompt, defaulting to a predefined model if none is specified. This generated text is then passed to Peer2's LLM, which processes it further to produce a concise summary. Both stages involve API calls that handle the request and response efficiently. The algorithm also logs critical metadata, such as processing times, model details, and results, into a CSV file for tracking and analysis. The final summary, along with intermediate results, is returned as a JSON response, ensuring seamless integration into downstream applications. By distributing tasks across two peers and leveraging model-specific capabilities, the algorithm achieves scalability, modularity, and accuracy in text generation and summarization.

The complexity of the text generation and summarization algorithm primarily depends on the interactions with the Peer1 and Peer2 APIs. Assuming the input prompt length is n (characters or tokens), the time complexity for generating text on Peer1 is $O(n)$, as large language models (LLMs) typically process prompts in a linear fashion. Similarly, the summarization process on Peer2 operates in $O(m)$, where m is the length of the text generated by Peer1. Consequently, the overall complexity of the algorithm is $O(n + m)$, dominated by the larger of n or m. Other operations, such as initializing the CSV file and logging data, are constant $O(1)$ since they do not scale with the input size. However, network latency and API response times may introduce overhead, but these are generally independent of n and m, provided the underlying models and systems scale efficiently. Therefore, the algorithm is efficient and suitable for practical use, assuming the LLMs and network infrastructure can handle the input size and concurrent requests.

4. Results and Discussions

This section presents the visualization aware and hypothesis testing results and their analysis to understand the capability of the proposed *P2PLLMEdge* framework. The analysis of the *P2PLLMEdge* framework involves evaluating key metrics to understand its performance. The *total_duration* represents the overall time spent generating a response, while the *load_duration* measures the time in nanoseconds required to load the model. The *prompt_eval_count* indicates the number of tokens present in the input prompt, with the *prompt_eval_duration* capturing the time taken in nanoseconds to process this input. Similarly, the *eval_count* tracks the number of tokens generated in the response, and the *eval_duration* quantifies the time in nanoseconds spent producing these tokens. To assess the efficiency of token generation, the response speed

Algorithm 1 P2PLLMEdge: Text Generation and Summarization

Require: JSON input with fields: `prompt` (required), `model` (optional)
Ensure: Final summarized text from Peer2's LLM

Peer1 API URL: `http://<IP>:11434/api/generate`
Peer2 API URL: `http://<IP>:11434/api/generate`
Initialize CSV file using `init_csv()`
Receive JSON input via Flask endpoint `/generate`
if JSON body is empty or `prompt` is missing **then**
    Return error response with HTTP 400
end if
Set `peer1_model` to input `model` or default `DEFAULT_PEER1_MODEL`
Call Peer1 API `call_peer1_generate(peer1_model, user_prompt)` via Peer1 API URL
if Peer1 call fails **then**
    Log error details
    Return error response to client
end if
Store generated text from Peer1 in `peer1_response`
Call Peer2 API `call_peer2_operations(DEFAULT_PEER2_MODEL, peer1_response)` via Peer2 API URL
if Peer2 call fails **then**
    Log error details
    Return error response to client
end if
Store summarized text from Peer2 in `peer2_response`
Merge results into a dictionary for CSV logging
Call `append_to_csv()` to log data
Return final summary and intermediate results as JSON response

is calculated in *tokens per second* (token/s) using the formula eval_count/eval_duration * 10^9, offering insights into the framework's throughput and responsiveness.

4.1. Web Server Wise Analysis

This study evaluates the performance of Peer 1 (Raspberry Pi 4B) and Peer 2 (Ubuntu Laptop) across various metrics using three server setups: FastAPI, Web Server, and Flask (Figure 3). The analysis reveals distinct performance characteristics and efficiency patterns across the configurations. The comparative evaluation of Peer 1 (Raspberry Pi 4B) and Peer 2 (Ubuntu Laptop) across FastAPI and Flask web server configurations demonstrates distinct strengths and performance trade-offs. In terms of total duration, Peer 2 outperformed Peer 1 by approximately 45.7% in the FastAPI configuration (25.45 ns × 10^9 vs. 46.87 ns × 10^9) and 43.1% in the Flask configuration (23.36 ns × 10^9 vs. 41.05 ns × 10^9). Conversely, Peer 1 exhibited superior load duration efficiency, achieving a remarkable 85.1% reduction in FastAPI (0.18 ns × 10^9 vs. 1.21 ns × 10^9) and a 37.6% reduction in Flask (0.53 ns × 10^9 vs. 0.85 ns × 10^9). These results indicate Peer 1's advantage in server initialization tasks, particularly with FastAPI.

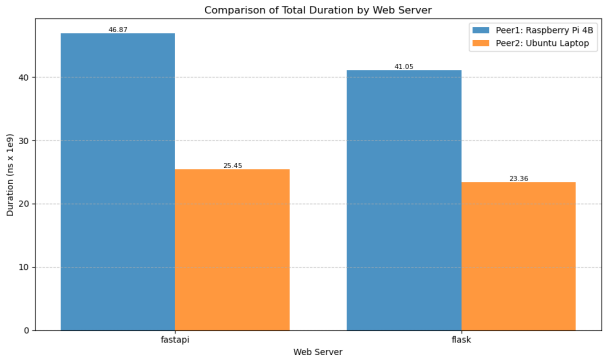
For prompt evaluation metrics, Peer 2 demonstrated vastly superior capacity, processing approximately 10 times more prompts than Peer 1. Specifically, Peer 2 handled 403.68 prompts in FastAPI compared to Peer 1's 40.40 prompts, and 355.43 prompts in Flask compared to Peer 1's 40.40 prompts. However, Peer 1 excelled in prompt evaluation duration, with a 94.3% faster execution in FastAPI (0.83 ns × 10^9 vs. 14.70 ns × 10^9) and a 91.9% reduction in Flask (0.92 ns × 10^9 vs. 11.41 ns × 10^9). Similarly, Peer 1 achieved faster evaluation durations, with reductions of 79.2% in FastAPI (9.54 ns × 10^9 vs. 45.85 ns × 10^9) and 71.9% in Flask (11.10 ns × 10^9 vs. 39.59 ns × 10^9). Peer 2 excelled in throughput-related metrics, processing tokens at a rate 42.8% higher than Peer 1 in FastAPI (11.08 tokens/second vs. 7.79 tokens/second) and 66.1% higher in Flask (13.04 tokens/second vs. 7.85 tokens/second). Peer 2 also reduced network duration by 44.9% in FastAPI (25.83 ns × 10^9 vs. 46.92 ns × 10^9) and 42.3% in Flask (23.72 ns × 10^9 vs. 41.15 ns × 10^9). Flask marginally outperformed FastAPI for both peers in several metrics, highlighting its compatibility with the workloads under evaluation.

Peer 2 is the preferred choice for tasks requiring high throughput and computational intensity, making it ideal for large-scale operations. Conversely, Peer 1's low latency and faster initialization times render it suitable for lightweight or time-critical applications. The analysis underscores the trade-offs between computational power and efficiency, offering valuable insights into optimizing resource allocation and server configuration based on specific application requirements.

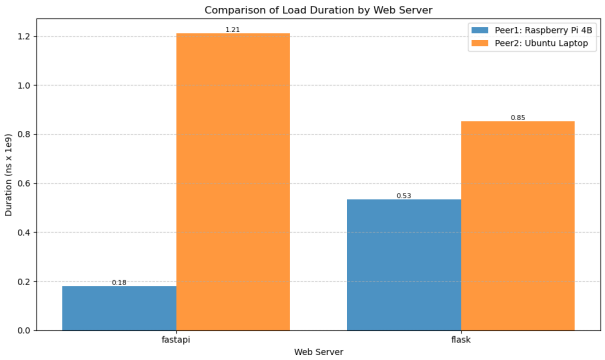
4.2. Prompt Type Wise Analysis

The comparative evaluation of Peer 1 and Peer 2 across various prompt types?Comparative, Creative, Explanation, Logical, and Numerical?provides insights into performance variations based on workload characteristics (Figure 4).

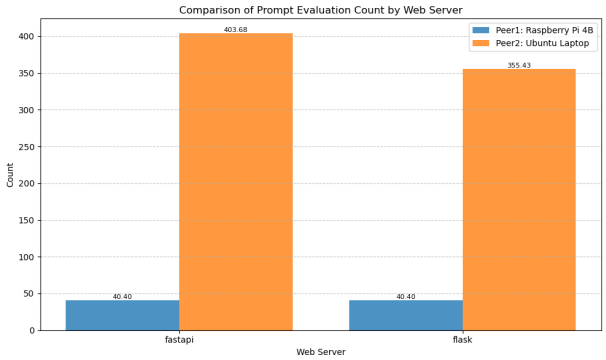
Peer 2 consistently outperforms Peer 1 in *total duration* across all prompt types. For example, in the Comparative prompt type, Peer 2 completes tasks in 38.00 ns × 10^9, a 44.7% reduction compared to Peer 1's 68.71 ns × 10^9. Similarly, for Logical prompts, Peer 2 achieves a duration of 15.87 ns × 10^9, which is 43.7% faster than Peer 1's 28.18 ns × 10^9. Numerical prompts also show Peer 2's advantage, completing in 13.08 ns × 10^9, a 24.3% improvement over Peer 1's 17.29 ns × 10^9.



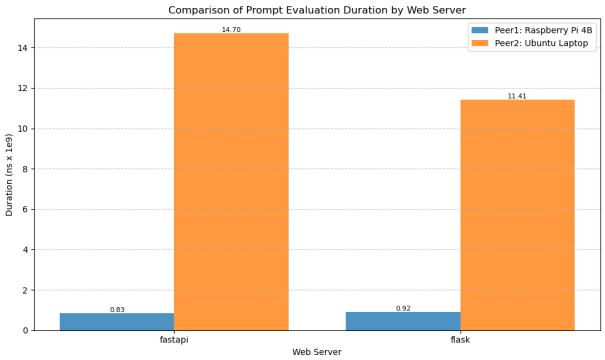
(a) Total Duration



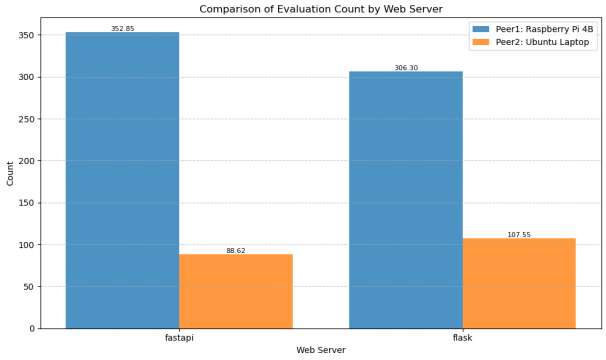
(b) Load Duration



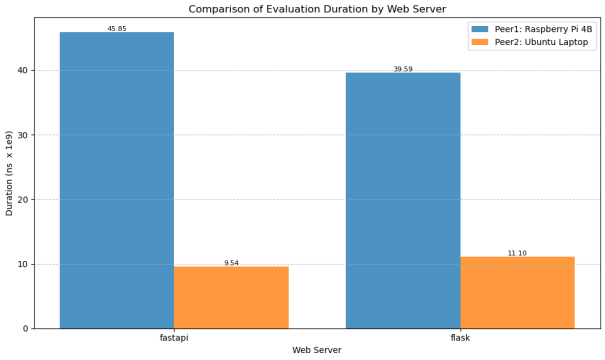
(c) Prompt Eval Count



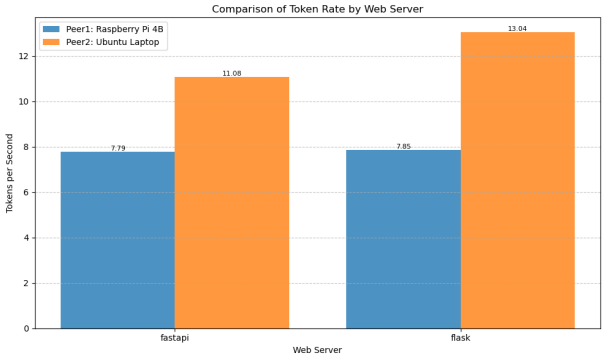
(d) Prompt Eval Duration



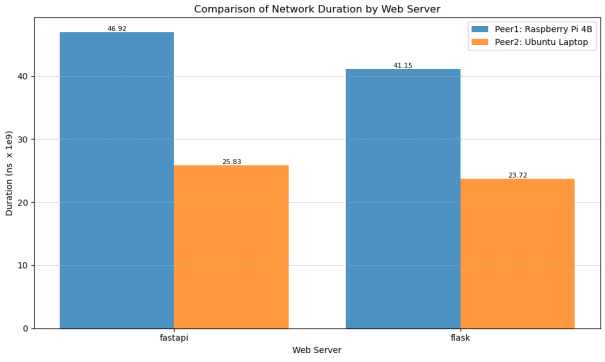
(e) Eval Count



(f) Eval Duration



(g) Tokens by Second



(h) Network Duration

Figure 3. Visualization of various metrics by web server

In terms of *load duration*, Peer 1 exhibits a significant advantage in Explanation prompts, requiring only 1.57 ns × 10^9, while Peer 2 takes 4.98 ns × 10^9, a 68.5% longer duration. For all other prompt types, the load duration differences are negligible, with both peers averaging around 0.05 ns × 10^9.

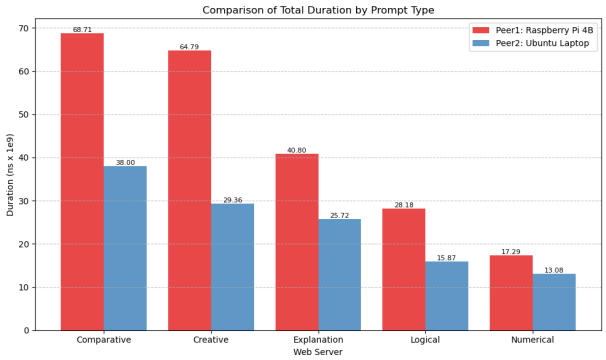
The *prompt evaluation count* clearly highlights Peer 2's superior throughput capacity. In the Comparative prompt type, Peer 2 handles 563.69 evaluations, a 1326% increase over Peer 1's 39.50 evaluations. For Numerical prompts, Peer 2 processes 174.25 evaluations, compared to Peer 1's 51.50 evaluations, marking a 238% improvement. Peer 2's capability to handle a larger number of evaluations demonstrates its computational superiority for high-throughput tasks. Regarding *prompt evaluation duration*, Peer 1 demonstrates remarkable efficiency. In Creative prompts, Peer 1 achieves 0.72 ns × 10^9, compared to Peer 2's 18.93 ns × 10^9, a 96.2% faster execution. Logical prompts similarly favor Peer 1, with durations of 0.65 ns × 10^9 versus Peer 2's 8.38 ns × 10^9, reflecting a 92.2% reduction. The *evaluation count* favors Peer 2 across all prompt types. For Comparative prompts, Peer 2 processes 516.56 evaluations, a 237% increase compared to Peer 1's 153.06 evaluations. Numerical prompts show Peer 2's strength as well, processing 128.69 evaluations, a 67% improvement over Peer 1's 76.56 evaluations. The evaluation duration analysis demonstrates Peer 2's consistent efficiency over Peer 1 across all prompt types. For Comparative prompts, Peer 2 achieves 18.09 ns × 10^9, representing a 73.4% reduction compared to Peer 1's 67.93 ns × 10^9. In Creative prompts, Peer 2 achieves a duration of 10.37 ns × 10^9, a 83.8% improvement over Peer 1's 64.01 ns × 10^9. Explanation, Logical, and Numerical prompts show Peer 2 reducing durations by 72.8%, 72.9%, and 50.3%, respectively. These results confirm Peer 2's superior capability in handling evaluation tasks efficiently across all prompt types. Finally, in terms of *token rate*, Peer 2 consistently achieves higher throughput. For Logical prompts, Peer 2 processes 12.35 tokens/second, a 56% improvement over Peer 1's 7.92 tokens/second. Numerical prompts exhibit a similar trend, with Peer 2 achieving 12.58 tokens/second, compared to Peer 1's 8.07 tokens/second, marking a 56% increase.

The comparison of network duration across various prompt types reveals significant differences between Peer 1 and Peer 2, with Peer 2 consistently demonstrating lower durations. For Comparative prompts, Peer 1 records a network duration of 68.73 ns × 10^9, whereas Peer 2 achieves 38.22 ns × 10^9, representing a 44.4% improvement. Similarly, for Creative prompts, Peer 1 requires 64.81 ns × 10^9, while Peer 2 completes the task in 29.60 ns × 10^9, a 54.3% faster execution. Explanation prompts further highlight Peer 2's efficiency, with a duration of 26.66 ns × 10^9 compared to Peer 1's 41.14 ns × 10^9, marking a 35.2% reduction. For Logical prompts, Peer 2 reduces the duration to 16.09 ns × 10^9, a 42.9% improvement over Peer 1's 28.20 ns × 10^9. Finally, for Numerical prompts, Peer 2 achieves a duration of 13.32 ns × 10^9, a 23.1% reduction compared to Peer 1's 17.31 ns × 10^9. These results demonstrate Peer 2's superior efficiency in network-related operations, particularly for creative and logical prompts, making it more suitable for tasks demanding high-performance network handling. Peer 1's longer durations suggest its use in less time-sensitive applications, emphasizing the importance of selecting appropriate hardware based on workload characteristics.

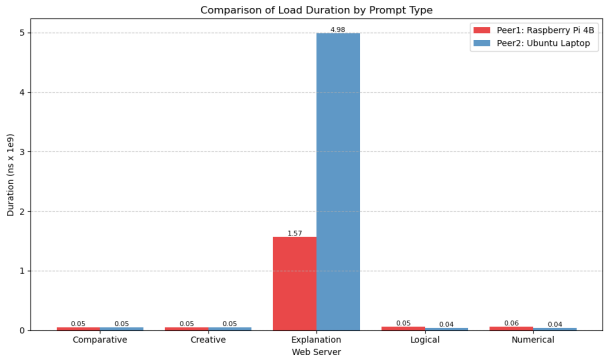
4.3. Peer1 LLM Wise Analysis

This section provides an analysis of Peer 1's performance metrics for the LLMs 'qwen2.5:0.5b-instruct' and 'smolm2:360m-instruct-q8_0' across various aspects (Figure 5). The total duration for 'qwen2.5:0.5b-instruct' is 47.33 ns × 10^9, which is 16.6% higher than 'smolm2:360m-instruct-q8_0', which records a total duration of 40.59 ns × 10^9. This indicates that 'smolm2:360m-instruct-q8_0' is more efficient in completing tasks. For load duration, 'qwen2.5:0.5b-instruct' demonstrates better performance with 0.28 ns × 10^9, which is 34.9% faster compared to 'smolm2:360m-instruct-q8_0' at 0.43 ns × 10^9. This suggests a more optimized loading mechanism for 'qwen2.5:0.5b-instruct'. In terms of prompt evaluation count, 'smolm2:360m-instruct-q8_0' slightly outperforms 'qwen2.5:0.5b-instruct' with 40.80 prompts compared to 40.00 prompts. The difference is minimal, indicating comparable throughput capacities for both models. 'smolm2:360m-instruct-q8_0' outperforms in prompt evaluation duration with 0.76 ns × 10^9, which is 23.2% faster than 'qwen2.5:0.5b-instruct' at 0.99 ns × 10^9. This demonstrates a significant efficiency advantage in evaluating prompts.

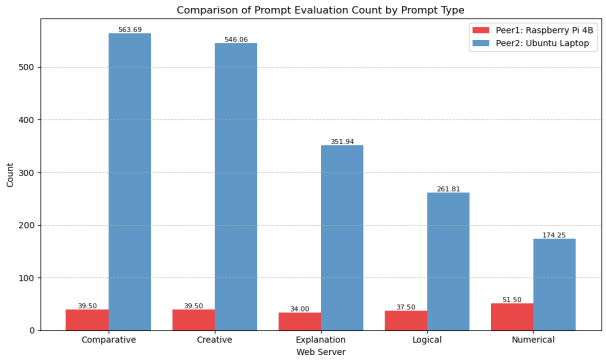
The evaluation count metric shows 'qwen2.5:0.5b-instruct' achieving 364.98 evaluations, which is 24.1% higher than 'smolm2:360m-instruct-q8_0' at 294.18 evaluations. This indicates that 'qwen2.5:0.5b-instruct' handles a greater number of evaluations within the given duration. For evaluation duration, 'smolm2:360m-instruct-q8_0' records 39.39 ns × 10^9, making it 14.5% faster than 'qwen2.5:0.5b-instruct' at 46.05 ns × 10^9. This highlights 'smolm2:360m-instruct-q8_0''s ability to complete evaluations more efficiently. The token rate for 'qwen2.5:0.5b-instruct' is 8.03 tokens/second, slightly surpassing 'smolm2:360m-instruct-q8_0' at 7.61 tokens/second by 5.5%. This demonstrates a marginal advantage for 'qwen2.5:0.5b-instruct' in token processing speed.



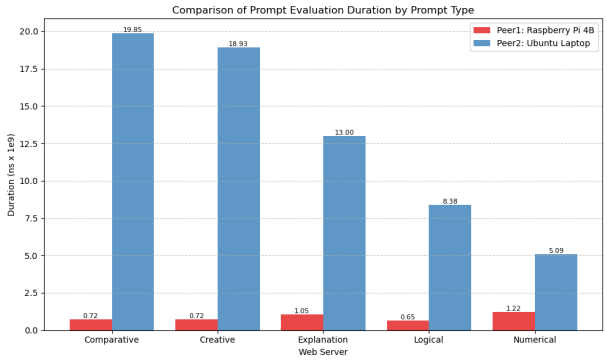
(a) Total Duration



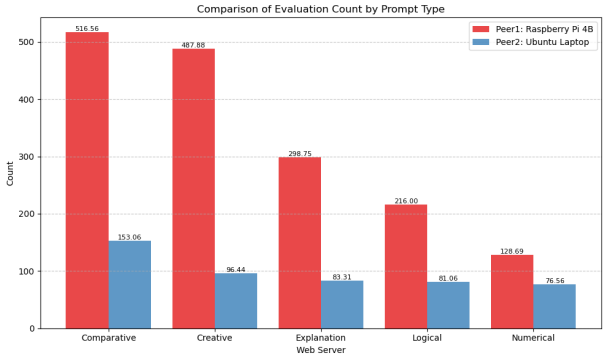
(b) Load Duration



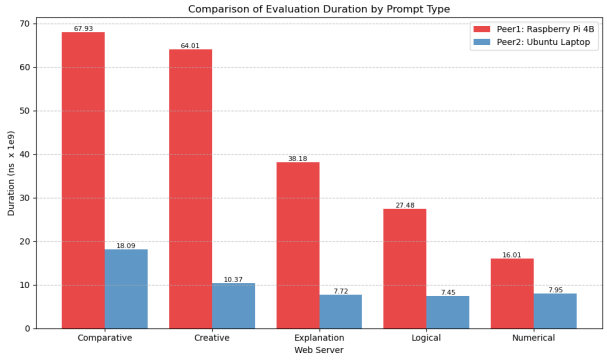
(c) Prompt Eval Count



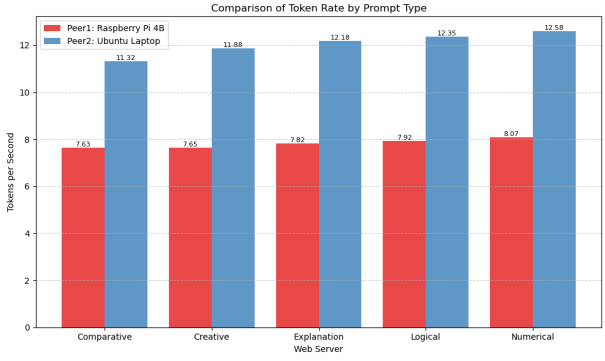
(d) Prompt Eval Duration



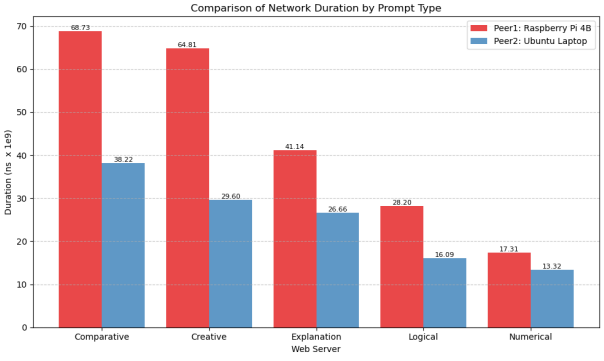
(e) Eval Count



(f) Eval Duration

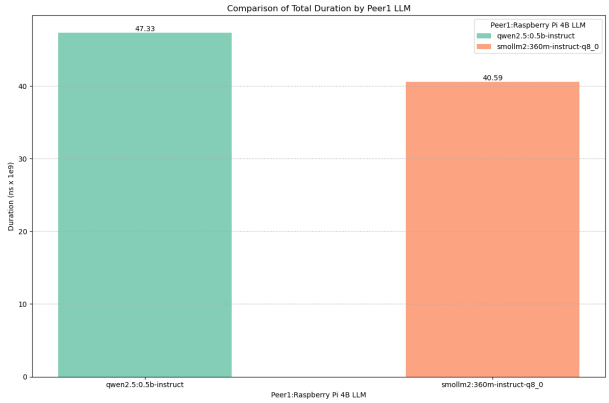


(g) Tokens by Second

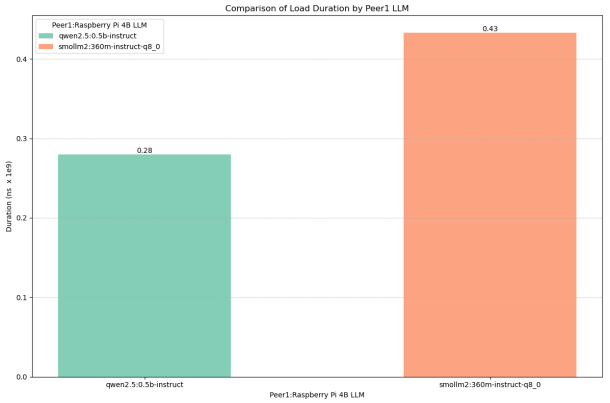


(h) Network Duration

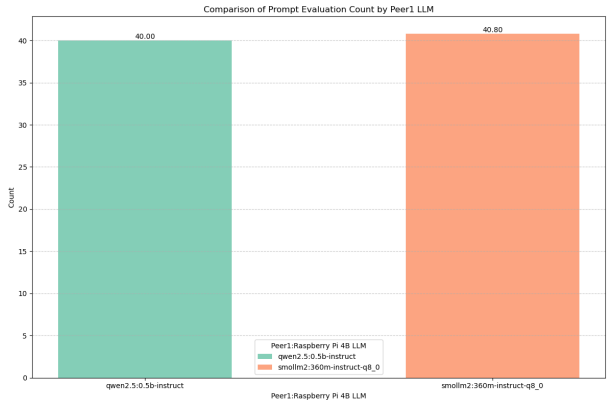
Figure 4. Visualization of various metrics by prompt types



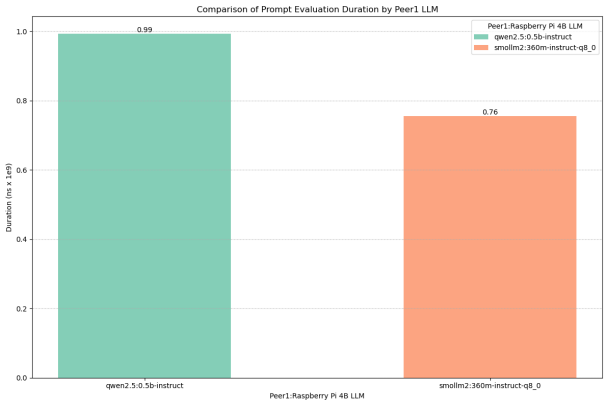
(a) Total Duration



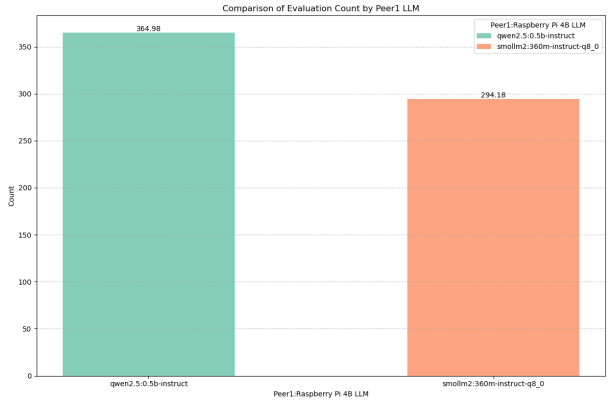
(b) Load Duration



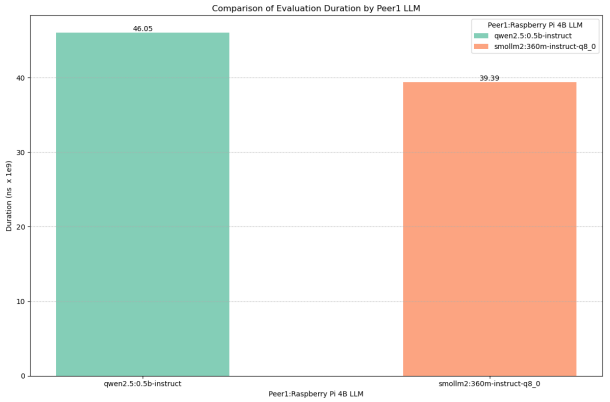
(c) Prompt Eval Count



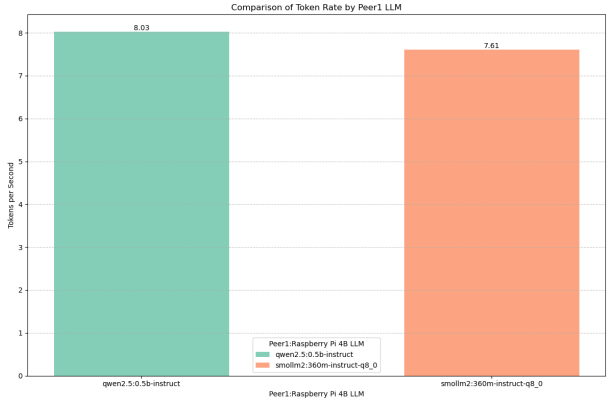
(d) Prompt Eval Duration



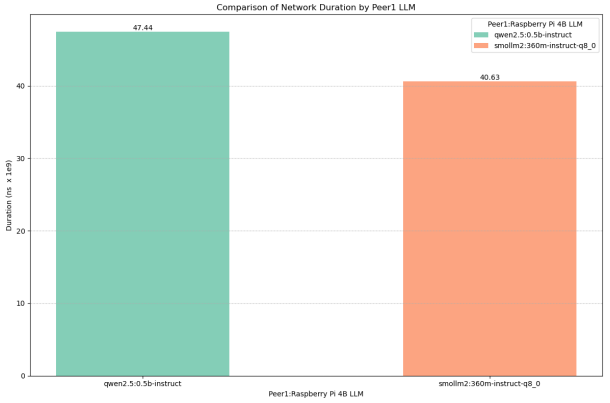
(e) Eval Count



(f) Eval Duration



(g) Tokens by Second



(h) Network Duration

Figure 5. Visualization of various metrics by peer1 LLMs

Finally, for network duration, 'smolm2:360m-instruct-q8_0' performs better with 40.63 ns × 10^9, which is 14.3% faster compared to 'qwen2.5:0.5b-instruct' at 47.44 ns × 10^9. This indicates 'smolm2:360m-instruct-q8_0''s efficiency in network-related tasks. Overall, 'smolm2:360m-instruct-q8_0' demonstrates superior performance in terms of total duration, evaluation duration, prompt evaluation duration, and network duration. However, 'qwen2.5:0.5b-instruct' excels in load duration, evaluation count, and token rate. The choice of LLM should be guided by the specific task requirements and workload characteristics.

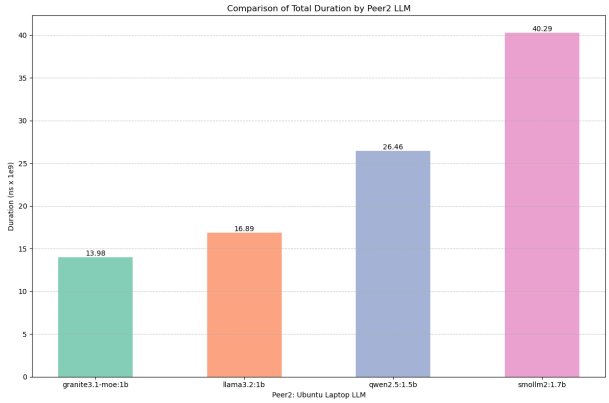
4.4. Peer2 LLM Wise Analysis

The analysis of Peer2's LLMs performance is conducted across multiple metrics, as illustrated in the eight charts provided (Figure 6). Each chart compares the performance of four LLMs: Granite3.1-moe:1b, Llama3.2:1b, Qwen2.5:1.5b, and Smolm2:1.7b, on the Ubuntu Laptop environment. The total duration for processing demonstrates that Granite3.1-moe:1b is the most efficient with a duration of 13.98 ns × 10^9, followed by Llama3.2:1b at 16.89 ns × 10^9 and Qwen2.5:1.5b at 26.46 ns × 10^9. Smolm2:1.7b is the least efficient with a duration of 40.29 ns × 10^9. This highlights the efficiency of smaller models in computational performance. Load duration analysis shows that Granite3.1-moe:1b requires the least time at 0.75 ns × 10^9. Llama3.2:1b and Qwen2.5:1.5b follow with durations of 1.08 ns × 10^9 and 0.85 ns × 10^9, respectively. Smolm2:1.7b has the highest load duration of 1.44 ns × 10^9, which indicates a trade-off in larger model initialization. Granite3.1-moe:1b outperforms the other LLMs with a count of 423.95, while Smolm2:1.7b achieves 400.60. Qwen2.5:1.5b and Llama3.2:1b achieve 366.35 and 327.30 counts, respectively. This reflects the capability of models to handle higher workloads effectively.

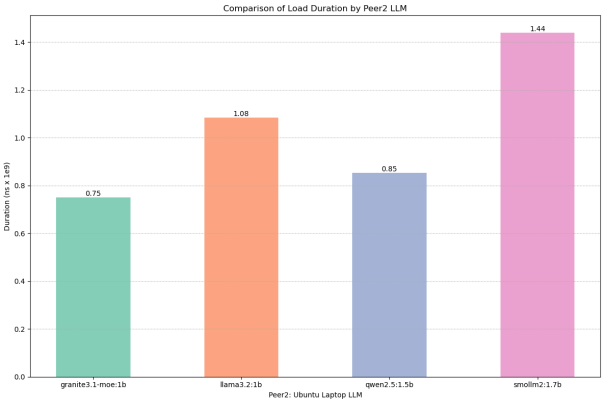
The evaluation duration is lowest for Granite3.1-moe:1b at 8.08 ns × 10^9, closely followed by Llama3.2:1b at 8.36 ns. Qwen2.5:1.5b and Smolm2:1.7b exhibit higher durations of 18.03 ns × 10^9 and 17.74 ns × 10^9, respectively, suggesting more computation-intensive processes. Granite3.1-moe:1b achieves the highest evaluation count of 108.40, with Smolm2:1.7b following at 136.50. Llama3.2:1b and Qwen2.5:1.5b demonstrate moderate counts of 69.90 and 77.55, respectively, revealing variations in task throughput. The evaluation duration for Granite3.1-moe:1b is 5.15 ns × 10^9, making it the fastest model. Llama3.2:1b and Qwen2.5:1.5b exhibit durations of 7.44 ns × 10^9 and 7.57 ns × 10^9, while Smolm2:1.7b has the highest duration of 21.11 ns × 10^9, indicating higher computational requirements. Token processing rates highlight Granite3.1-moe:1b as the fastest with 21.77 tokens/second. Qwen2.5:1.5b follows at 10.30 tokens/second, with Llama3.2:1b achieving 9.46 tokens/second. Smolm2:1.7b demonstrates the lowest token rate of 6.72 tokens/second, reflecting its larger architecture. Finally, the network duration is lowest for Granite3.1-moe:1b at 14.30 ns × 10^9. Llama3.2:1b, Qwen2.5:1.5b, and Smolm2:1.7b exhibit progressively higher durations of 17.34 ns × 10^9, 26.90 ns × 10^9, and 40.57 ns × 10^9, respectively, indicating the impact of model size on network communication latency. Overall, Granite3.1-moe:1b consistently outperforms other LLMs in efficiency metrics, while Smolm2:1.7b shows strengths in handling more complex tasks but at a higher computational cost. These findings highlight the trade-offs between model complexity and performance.

4.5. Prompt Type and Web Server Wise Analysis

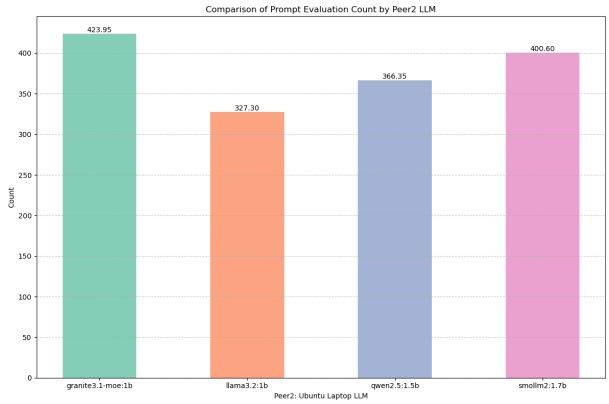
This section presents a comparative analysis of various prompt types across two different web server environments: Peer1: Raspberry Pi 4B and Peer2: Ubuntu Laptop. The results, illustrated in Figures 7, provide insights into execution time, evaluation efficiency, throughput, and network latency. The Ubuntu Laptop consistently demonstrates lower execution times compared to the Raspberry Pi 4B, with the difference being more pronounced for computationally intensive prompts. The Total Duration (Figure 7a) highlights the overall execution time, with the Ubuntu Laptop consistently performing faster than the Raspberry Pi 4B. The Load Duration (Figure 7b) shows the time taken for initializing execution, where the Raspberry Pi 4B experiences significantly higher overhead compared to the Ubuntu Laptop. The Prompt Evaluation Count (Figure 7c) represents the number of evaluations completed for different prompt types. The Ubuntu Laptop handles a higher number of evaluations, whereas the Raspberry Pi 4B processes fewer due to computational limitations. The Prompt Evaluation Duration (Figure 7d) further illustrates this efficiency gap, with the Raspberry Pi 4B taking considerably longer to process each evaluation compared to the Ubuntu Laptop. The Evaluation Count (Figure 7e) and Evaluation Duration (Figure 7f) show similar trends, where the Ubuntu Laptop consistently supports more evaluations while requiring significantly less time per evaluation, reinforcing its computational advantage. The Tokens per Second metric (Figure 7g) evaluates the processing speed in terms of token generation, with the Ubuntu Laptop achieving a notably higher token throughput compared to the Raspberry Pi 4B. The Network Duration (Figure 7h) indicates the latency in data transmission, where the Raspberry Pi



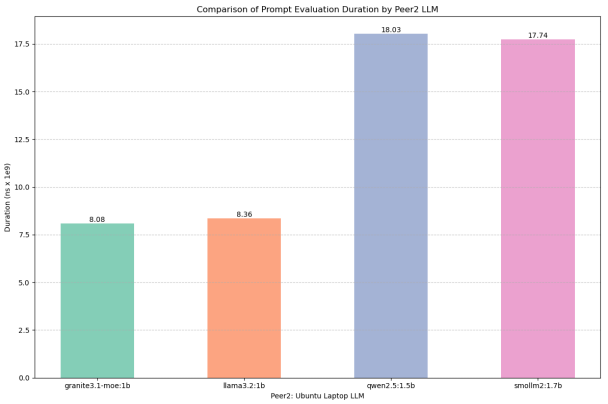
(a) Total Duration



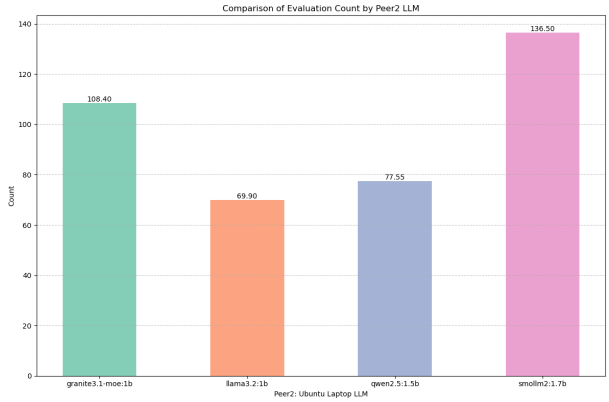
(b) Load Duration



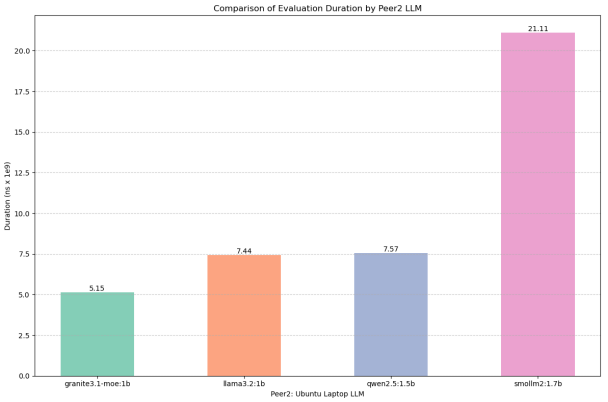
(c) Prompt Eval Count



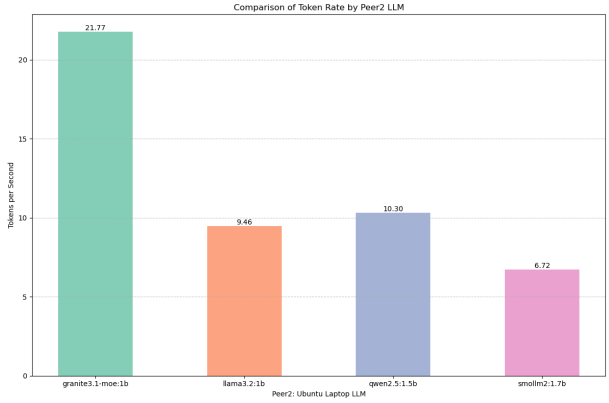
(d) Prompt Eval Duration



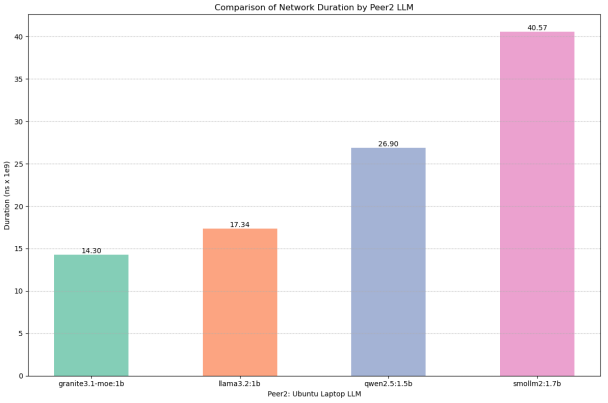
(e) Eval Count



(f) Eval Duration



(g) Tokens by Second



(h) Network Duration

Figure 6. Visualization of various metrics by peer2 LLMs

4B exhibits significantly higher network delays, further limiting its feasibility for real-time applications. The Ubuntu Laptop consistently outperforms the Raspberry Pi 4B across all execution time and evaluation metrics. Prompt evaluation efficiency is constrained on the Raspberry Pi 4B, leading to fewer evaluations and longer processing times. Token throughput is significantly higher on the Ubuntu Laptop, making it more suitable for high-speed text generation. Network latency is considerably lower on the Ubuntu Laptop, which is beneficial for real-time applications. These findings underscore the trade-offs between deploying LLM-based applications on edge devices versus more powerful server hardware. Future optimizations could explore load balancing techniques, caching strategies, and hardware acceleration to enhance the feasibility of Raspberry Pi 4B for lightweight deployments.

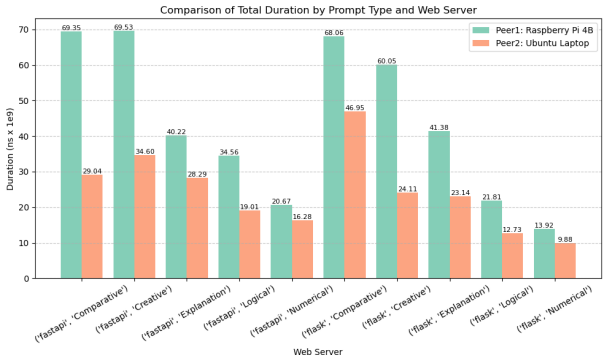
4.6. Correlation Matrix

The analysis of the P2P interaction within the localized LLM framework highlights the interdependencies between metrics of peer1 and peer2 during sequential task processing. Peer1 generates text output, which is subsequently utilized by peer2 for summarization. From the correlation matrix (Figure 8), we observe a strong positive correlation ($r > 0.9$) between the total duration metrics of peer1 and peer2, suggesting that the generation and summarization stages are temporally linked. This indicates that any delay or performance optimization in peer1's text generation directly affects peer2's summarization process, making it crucial to balance both operations for effective real-time performance. A deeper inspection reveals that peer1's token-per-second metric exhibits a moderate negative correlation ($r = -0.57$) with its total duration, reflecting a trade-off between throughput and response latency. This trade-off implies that as peer1 processes tokens faster, the total duration required for generating responses decreases, improving its efficiency. However, it also points to potential bottlenecks that could arise if the system is optimized purely for speed without considering the quality or structure of the generated text. The prompt evaluation count of peer1 and peer2 is highly correlated ($r \approx 0.99$), indicating consistency in token usage between generation and summarization tasks. This high correlation suggests that peer2 effectively utilizes the tokens produced by peer1, ensuring minimal loss or redundant processing in the summarization phase. Similarly, the evaluation duration for peer2 shows a significant positive correlation ($r = 0.85$) with its token-per-second metric, reinforcing the efficiency of token processing during summarization. This observation underscores the capability of peer2 to handle high-throughput scenarios without compromising its summarization quality or speed. Interestingly, the load duration of peer1 and peer2 exhibits a weak correlation ($r = 0.28$), which may be attributed to independent model initialization or caching strategies employed by each peer. This weak dependency suggests that each peer operates semi-autonomously in terms of model loading, allowing for flexibility and scalability in distributed environments. Moreover, the network duration ($r \approx 0.61$) demonstrates a notable correlation between peers, signifying consistent communication overheads during data exchange. This consistency in network communication ensures reliable data transfer between peers, which is critical for maintaining synchronization in sequential tasks. These findings collectively highlight the efficiency and interdependency of the *P2PLLMEdge* framework in orchestrating peer-to-peer interactions for real-time localized language processing. The framework effectively balances computation time with throughput, ensuring high-performance demands are met. By leveraging the strengths of each peer, the system demonstrates scalability, adaptability, and robustness in handling complex language processing tasks, making it an ideal solution for decentralized AI-driven applications.

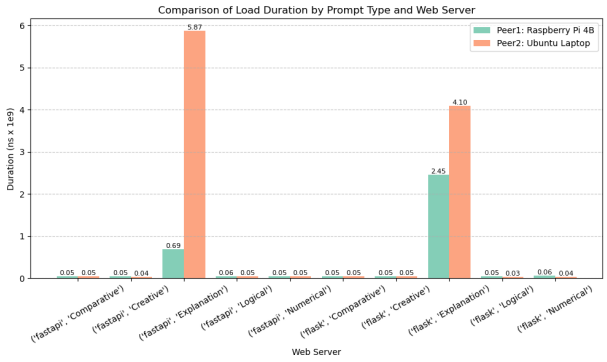
4.7. Outlier and Distribution

The violin plot presented in Figure 9 provides a comprehensive visualization of the distribution and variability of key metrics across peer1 and peer2 interactions within the *P2PLLMEdge* framework. The distributions highlight critical insights into the temporal and computational dynamics of localized language processing tasks.

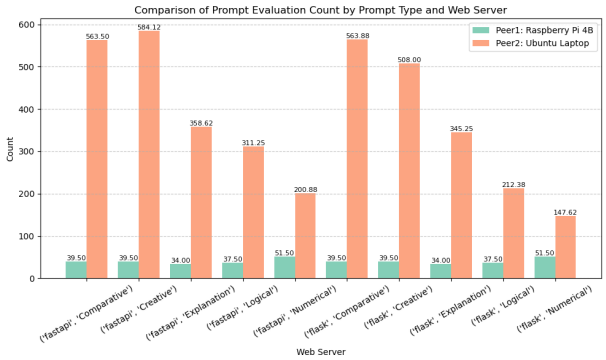
The *peer1_total_duration* and *peer2_total_duration* metrics exhibit broad distributions, indicating substantial variability in the time required for generation and summarization tasks. This variability underscores the complex dependencies on task-specific inputs, computational overheads, and network delays. Such dependencies suggest that variations in input size or model complexity can significantly impact processing times, necessitating dynamic resource allocation strategies to mitigate delays. The narrower distribution of *peer1_load_duration* suggests that model loading times are relatively consistent, likely due to efficient caching and initialization strategies, which are critical for optimizing response readiness. However, *peer2_load_duration* displays a slightly wider spread, potentially reflecting differences in workload balancing or initialization dependencies arising from peer1 outputs. The *peer1_prompt_eval_count*



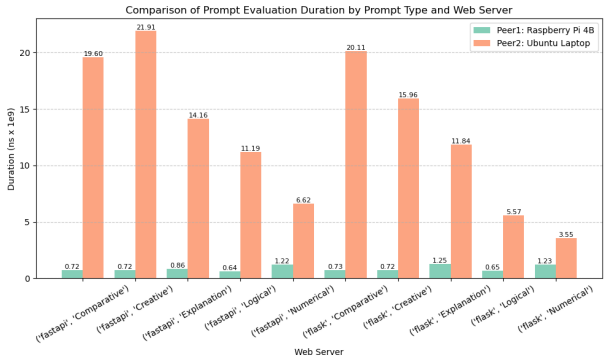
(a) Total Duration



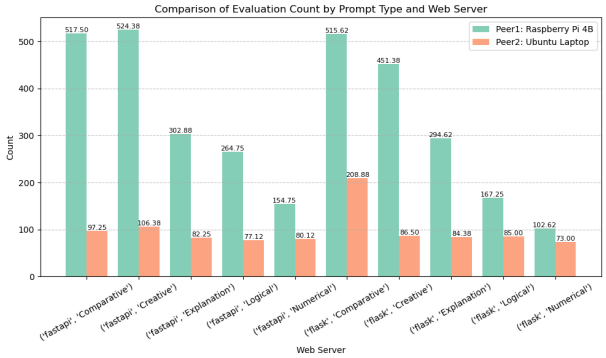
(b) Load Duration



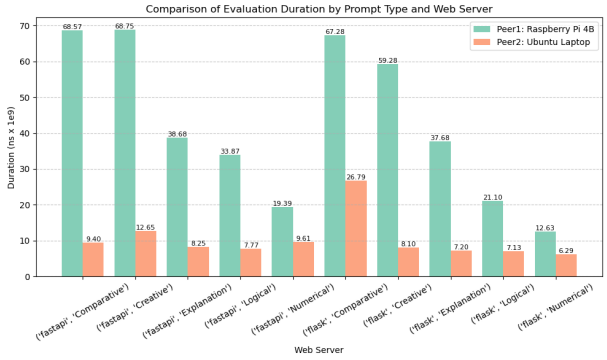
(c) Prompt Eval Count



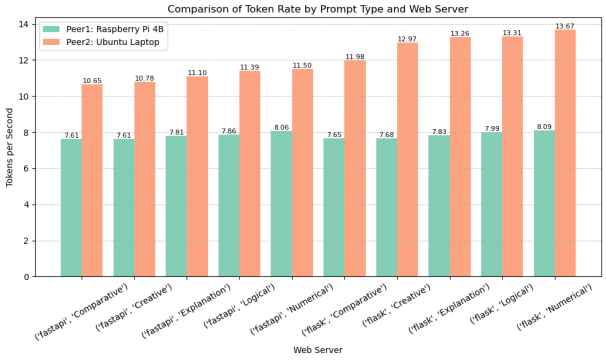
(d) Prompt Eval Duration



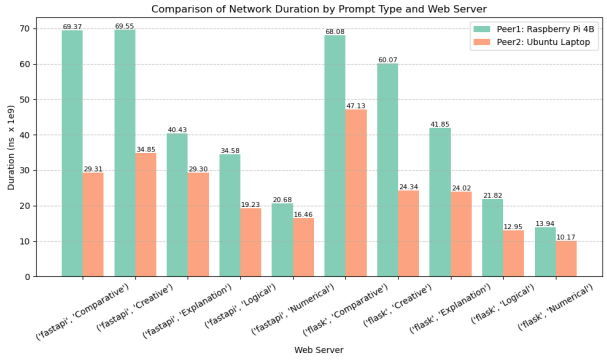
(e) Eval Count



(f) Eval Duration



(g) Tokens by Second



(h) Network Duration

Figure 7. Visualization of various metrics by prompt type and web server

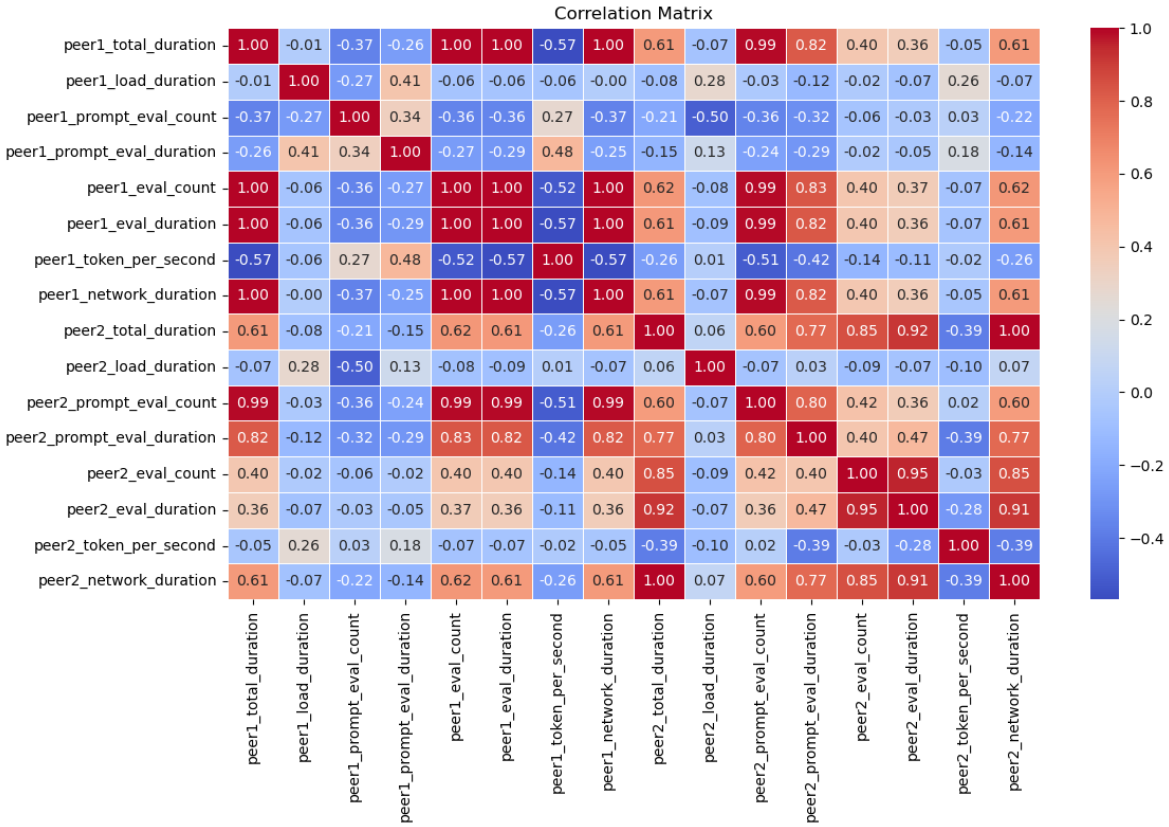


Figure 8. Correlation matrix.

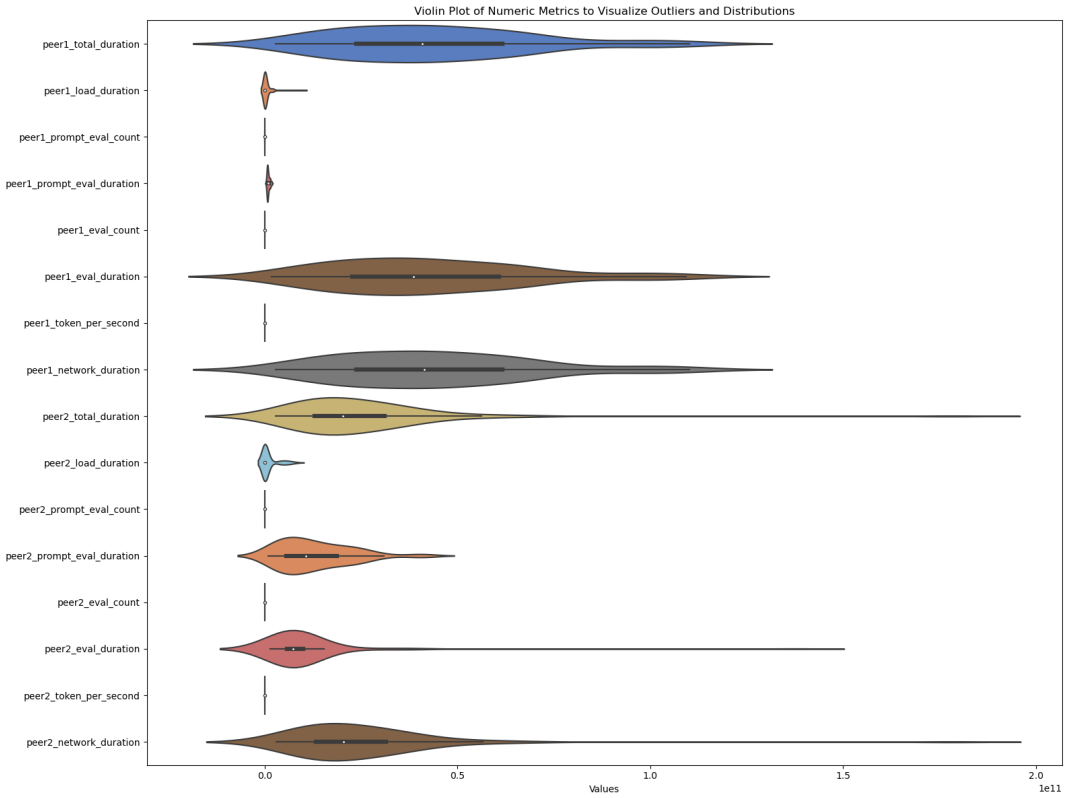


Figure 9. Outlier detection and distribution.

and *peer2_prompt_eval_count* metrics demonstrate distinct peaks with minimal variance, indicative of consistent token counts during evaluation. This consistency aligns with the high correlation observed between these metrics, ensuring seamless data flow between peers. The uniformity in token counts suggests that the input-output token pipeline is robust, enabling predictable performance across tasks. Conversely, the *peer1_eval_duration* and *peer2_eval_duration* metrics reveal longer tails, suggesting occasional outliers that significantly impact response times. These outliers could be attributed to increased token complexity, model attention mechanisms, or temporary network congestion during summarization. The *peer1_token_per_second* metric, characterized by a relatively compact distribution, highlights the efficiency of token processing during text generation. A similar pattern is observed for *peer2_token_per_second*, though with a slightly broader spread, likely reflecting additional computational demands of summarization. These metrics underline the importance of throughput optimization to balance speed and accuracy in token generation. Interestingly, the *peer1_network_duration* and *peer2_network_duration* metrics exhibit overlapping distributions, suggesting consistent and reliable communication overheads between peers. This consistency is essential for ensuring synchronization and data integrity in P2P frameworks, where any delay in communication could cascade into performance bottlenecks. The overlapping distributions also imply that the network infrastructure supporting the framework is robust. Overall, the violin plot illustrates the nuanced interplay between computational efficiency, network reliability, and task complexity in the *P2PLLMEdge* framework. The observed distributions and outliers provide actionable insights into potential optimization areas, such as minimizing evaluation durations, improving token throughput, and streamlining network communication. These findings reinforce the robustness and scalability of the *P2PLLMEdge* framework in handling dynamic, decentralized language processing tasks, making it well-suited for real-world applications where adaptability and performance are paramount.

4.8. Hypothesis Testing

One-Way Analysis of Variance (ANOVA)

The one-way ANOVA is used to determine whether there are statistically significant differences between the means of three or more independent groups. The test statistic, F, is calculated as:

$$F = \frac{\text{MSB}}{\text{MSW}} \tag{1}$$

where:

$$\text{MSB} = \frac{\sum_{i=1}^{k} n_i(\bar{X}_i - \bar{X})^2}{k-1},$$
$$\text{MSW} = \frac{\sum_{i=1}^{k}\sum_{j=1}^{n_i}(X_{ij} - \bar{X}_i)^2}{N-k}.$$

Here:

- k: Number of groups.
- n_i: Sample size of the i-th group.
- $\bar{X}_i$: Mean of the i-th group.
- $\bar{X}$: Overall mean of all observations.
- N: Total number of observations.
- MSB: Mean square between groups (variation due to group differences).
- MSW: Mean square within groups (variation within each group).

The F-value is compared to the critical value from the F-distribution table to determine statistical significance.

Tukey's Honest Significant Difference (HSD) Test:

If the ANOVA result is significant, Tukey's HSD test is performed for pairwise comparisons. The HSD value is computed as:

$$\text{HSD} = q \cdot \sqrt{\frac{\text{MSW}}{n}} \tag{2}$$

where:

- q: Critical value from the Studentized range distribution for the given degrees of freedom and number of groups.
- MSW: Mean square within groups (as calculated in ANOVA).
- n: Number of observations per group (assuming equal group sizes).

For unequal group sizes, the formula adjusts n to:

$$n = \frac{2}{\frac{1}{n_1} + \frac{1}{n_2}} \tag{3}$$

where n_1 and n_2 are the sizes of the two groups being compared.

The test compares the absolute mean difference between each pair of groups to the HSD value. For any pair, if:

$$|\bar{X}_i - \bar{X}_j| > \text{HSD}, \tag{4}$$

then the difference between groups i and j is considered statistically significant.

Statistical Analysis of Peer1 Metrics by Prompt Type. The statistical analysis evaluates the impact of `prompt_type` on various performance metrics of Peer1. The null hypothesis (H_0) and the alternative hypothesis (H_a) are formulated as follows:

- H_0: There is no significant difference in the mean performance metrics of Peer1 across different `prompt_type`.
- H_a: At least one `prompt_type` has a significantly different mean performance metric compared to others.

The hypotheses are tested using a one-way ANOVA, followed by post-hoc Tukey HSD tests when significant differences are detected. The results of the ANOVA tests for Peer1 metrics grouped by `prompt_type` are summarized in Table 4. Metrics with significant differences ($p < 0.05$) are indicated.

Post-hoc Tukey HSD tests were conducted for metrics with significant ANOVA results ($p < 0.05$). Selected comparisons are presented in Table 5 for the `peer1_total_duration` metric. Full results for other metrics are available upon request.

The ANOVA results indicate that all metrics tested exhibited statistically significant differences across `prompt_type`, with $p < 0.05$. For `peer1_total_duration`, the Tukey HSD analysis revealed that: (i) significant differences exist between the `Comparative` prompt type and the `Explanation`, `Logical`, and `Numerical` prompt types and (ii) similar significant differences were found between `Creative` prompts and other types, except `Comparative`. For `peer1_load_duration`, while significant differences were observed overall, post-hoc analysis highlighted specific differences primarily between `Explanation` and other types (`Creative`, `Numerical`). The analysis of `peer1_prompt_eval_count` and `peer1_eval_duration` revealed highly significant differences among all prompt types, suggesting that the nature of the prompt directly influences the token count and evaluation time. This analysis confirms that `prompt_type` has a substantial effect on the performance metrics of Peer1. These insights are crucial for optimizing localized LLM deployments, particularly in resource-constrained edge environments where task-specific configurations can significantly enhance performance.

Statistical Analysis of Peer2 Metrics by Prompt Type. The study conducts the effect of `prompt_type` on Peer2's performance metrics. The hypotheses are defined as follows:

- H_0: There is no significant difference in the mean performance metrics of Peer2 across different `prompt_type`.
- H_a: At least one `prompt_type` exhibits a significantly different mean performance metric compared to others.

Table 6 summarizes the results of the ANOVA tests for Peer2 metrics by `prompt_type`. Metrics with significant differences are highlighted.

For metrics with significant ANOVA results, post-hoc Tukey HSD tests were conducted. Table 7 provides the pairwise comparisons for `peer2_total_duration`.

Similarly, the Tukey HSD results for `peer2_load_duration` and `peer2_prompt_eval_count` showed significant differences between `Explanation` and other prompt types. The one-way ANOVA results indicate crucial observations as mentioned follows. Significant differences in `peer2_total_duration` were observed between `Comparative` prompts and both `Logical` and `Numerical` prompts. `peer2_load_duration` exhibited the most significant differences ($F = 111.873, p = 0.000$), with `Explanation` prompts differing notably from all others. The `peer2_prompt_eval_count` metric also showed significant variation, with `Explanation`, `Logical`, and `Numerical` prompts differing from `Comparative` and `Creative` prompts. Metrics such as `peer2_eval_count`, `peer2_eval_duration`, and `peer2_token_per_second` did not show significant differences, indicating consistency in Peer2's processing capacity for these metrics across different prompt types. This analysis demonstrates that `prompt_type` influences key performance metrics such as `peer2_total_duration`, `peer2_load_duration`, and `peer2_prompt_eval_count`, while others remain unaffected. These findings are critical for optimizing task distribution in Peer2's LLM framework, especially for load-intensive or token-rich prompts.

Statistical Analysis of Peer1 Metrics. We study the impact of different web server frameworks (`fastapi` and `flask`) on Peer1's performance metrics. The hypotheses are as follows:

- H_0: There is no significant difference in the mean performance metrics of Peer1 across different web server frameworks.
- H_a: At least one web server framework exhibits a significantly different mean performance metric compared to the others.

Table 8 summarizes the results of the ANOVA tests for Peer1 metrics. Metrics with significant differences are highlighted.

For metrics with significant ANOVA results, post-hoc Tukey HSD tests were conducted. The results are presented in Tables 9 and 10.

Significant differences were observed in `peer1_prompt_eval_duration` ($F = 10.806, p = 0.002$)

Table 4. ANOVA Results for Peer1 Metrics by `prompt_type`

Metric	F-statistic	p-value	Significant Difference?
`peer1_total_duration`	29.992	0.000	Yes
`peer1_load_duration`	6.332	0.000	Yes
`peer1_prompt_eval_count`	3266.250	0.000	Yes
`peer1_prompt_eval_duration`	14.320	0.000	Yes
`peer1_eval_count`	29.557	0.000	Yes
`peer1_eval_duration`	30.677	0.000	Yes
`peer1_token_per_second`	8.123	0.000	Yes
`peer1_network_duration`	29.931	0.000	Yes

Table 5. Tukey HSD Test Results for `peer1_total_duration` by `prompt_type`

Group1	Group2	Mean Difference	Significant?
Comparative	Creative	-3915045883.375	No
Comparative	Explanation	-27902870708.75	Yes
Comparative	Logical	-40520788112.25	Yes
Comparative	Numerical	-51411900707.25	Yes
Creative	Explanation	-23987824825.375	Yes
Creative	Logical	-36605742228.875	Yes
Creative	Numerical	-47496854823.875	Yes
Explanation	Logical	-12617917403.5	No
Explanation	Numerical	-23509029998.5	Yes
Logical	Numerical	-10891112595.0	No

Table 6. ANOVA Results for Peer2 Metrics by `prompt_type`

Metric	F-statistic	p-value	Significant Difference?
`peer2_total_duration`	3.978	0.006	Yes
`peer2_load_duration`	111.873	0.000	Yes
`peer2_prompt_eval_count`	27.167	0.000	Yes
`peer2_prompt_eval_duration`	11.147	0.000	Yes
`peer2_eval_count`	1.847	0.129	No
`peer2_eval_duration`	1.342	0.262	No
`peer2_token_per_second`	0.102	0.981	No
`peer2_network_duration`	4.001	0.005	Yes

Table 7. Tukey HSD Test Results for `peer2_total_duration` by `prompt_type`

Group1	Group2	Mean Difference	Significant?
Comparative	Creative	-8642804472.0625	No
Comparative	Explanation	-12282504079.25	No
Comparative	Logical	-22130071674.875	Yes
Comparative	Numerical	-24917155305.5625	Yes
Creative	Explanation	-3639699607.1875	No
Creative	Logical	-13487267202.8125	No
Creative	Numerical	-16274350833.5	No
Explanation	Logical	-9847567595.625	No
Explanation	Numerical	-12634651226.3125	No
Logical	Numerical	-2787083630.6875	No

and `peer1_token_per_second` ($F = 67.372, p = 0.000$). However, post-hoc Tukey HSD tests did not indicate significant pairwise differences between `fastapi` and `flask`. Metrics such as `peer1_total_duration`, `peer1_load_duration`, and `peer1_network_duration` showed no significant differences, suggesting consistent

Table 8. ANOVA Results for Peer1 Metrics by Web Server Framework

Metric	F-statistic	p-value	Significant Difference?
peer1_total_duration	1.375	0.244	No
peer1_load_duration	0.315	0.576	No
peer1_prompt_eval_count	0.358	0.551	No
peer1_prompt_eval_duration	10.806	0.002	Yes
peer1_eval_count	2.709	0.104	No
peer1_eval_duration	1.327	0.253	No
peer1_token_per_second	67.372	0.000	Yes
peer1_network_duration	1.406	0.239	No

Table 9. Tukey HSD Test Results for peer1_prompt_eval_duration

Group1	Group2	Mean Difference	Significant?
fastapi	flask	83875000.0	No

Table 10. Tukey HSD Test Results for peer1_token_per_second

Group1	Group2	Mean Difference	Significant?
fastapi	flask	0.0573	No

performance across web server frameworks for these metrics. The lack of significant pairwise differences in Tukey HSD tests for the significant ANOVA results indicates that while overall variability exists, the differences between the two web server frameworks are not substantial. The results suggest that while there are overall differences in peer1_prompt_eval_duration and peer1_token_per_second, the specific choice between fastapi and flask does not lead to significant performance improvements for Peer1 metrics.

Statistical Analysis of Peer2 Metrics. The present statistical analysis finds the significance of different web server frameworks (fastapi and flask) on Peer2's performance metrics. The hypotheses are stated as below:

- H_0: There is no significant difference in the mean performance metrics of Peer2 across different web server frameworks.
- H_a: At least one web server framework exhibits a significantly different mean performance metric compared to the others.

The ANOVA results for Peer2 metrics are summarized in Table 11. Metrics with significant differences are highlighted.

For metrics with significant ANOVA results, post-hoc Tukey HSD tests were conducted. The results are provided in Tables 12, 13, 14, 15, and 16.

The analysis can be be framed as follows. Significant differences were observed in peer2_total_duration, peer2_prompt_eval_duration, peer2_eval_duration, peer2_token_per_second, and peer2_network_duration. However, post-hoc Tukey HSD tests did not reveal significant pairwise differences between fastapi and flask. Metrics such as peer2_load_duration, peer2_prompt_eval_count, and peer2_eval_count showed no significant differences, indicating consistent performance across web server frameworks for these metrics. While the ANOVA results highlight some significant overall variability, the lack of significant Tukey HSD pairwise differences suggests that the choice between fastapi and flask has a minimal impact on Peer2's performance metrics. The findings suggest that while there are variations in certain metrics for Peer2, the choice of web server framework (fastapi or flask) does not lead to significant differences in practical performance.

4.9. Limitations and Future Scope

While *P2PLLMEdge* demonstrates that localized, CPU-only peers can cooperatively execute LLM workloads, several limitations remain. First, our current implementation supports only two peers with statically assigned roles (generation vs. summarization). In real deployments, dynamic peer discovery, heterogeneous model placement, and automated load balancing would be required to accommodate varying device availability and task demands. Second, security and authentication between peers have not been addressed; sensitive environments (e.g., healthcare clinics) will require encrypted channels and mutual attestation. Third, energy consumption and thermal characteristics on battery-powered devices were not evaluated; future

Table 11. ANOVA Results for Peer2 Metrics by Web Server Framework

Metric	F-statistic	p-value	Significant Difference?
peer2_total_duration	7.313	0.000	Yes
peer2_load_duration	0.395	0.757	No
peer2_prompt_eval_count	0.871	0.460	No
peer2_prompt_eval_duration	9.015	0.000	Yes
peer2_eval_count	2.157	0.100	No
peer2_eval_duration	4.965	0.003	Yes
peer2_token_per_second	372.732	0.000	Yes
peer2_network_duration	7.244	0.000	Yes

Table 12. Tukey HSD Test Results for peer2_total_duration

Group1	Group2	Mean Difference	Significant?
fastapi	flask	-2083343018.125	No

Table 13. Tukey HSD Test Results for peer2_prompt_eval_duration

Group1	Group2	Mean Difference	Significant?
fastapi	flask	-3290225000.0	No

Table 14. Tukey HSD Test Results for peer2_eval_duration

Group1	Group2	Mean Difference	Significant?
fastapi	flask	1565375000.0	No

Table 15. Tukey HSD Test Results for peer2_token_per_second

Group1	Group2	Mean Difference	Significant?
fastapi	flask	1.9538	No

Table 16. Tukey HSD Test Results for peer2_network_duration

Group1	Group2	Mean Difference	Significant?
fastapi	flask	-2108524204.275	No

work should profile power usage to inform model-selection heuristics for truly untethered edge deployments [37–39].

Looking forward, we envision several extensions and application scenarios [40–42]:

- **Multi-Peer Scaling:** In place of our fixed two-node setup, a peer registry service could track all active devices, their supported models, and current load. A lightweight scheduler would then decompose complex tasks—such as simultaneous translation, named-entity recognition, and summarization—into subtasks and assign them to peers best suited by compute capacity and memory bound. When new devices join (or fail), the registry updates automatically and redistributes queued jobs, enabling elastic horizontal scaling without manual reconfiguration.
- **Adaptive Task Scheduling:** Beyond static offloading rules, we plan to integrate a telemetry agent into each peer that periodically reports CPU utilization, free RAM, local temperature, and round-trip network latency. A centralized or gossip-based controller can then use these metrics to predict which peer will complete a given prompt fastest, dynamically routing queries accordingly. Machine-learning models trained on historical performance traces could further anticipate resource contention and preemptively redistribute tasks to maintain consistent quality-of-service.
- **Secure Edge Collaboration:** To protect sensitive data in healthcare or financial settings, all REST/JSON traffic between peers will be encapsulated in mutual-TLS tunnels, ensuring end-to-end encryption and peer identity verification. Where

supported, we will leverage hardware roots of trust (e.g., TPM chips or ARM TrustZone) to securely store private keys and attest to the integrity of the LLM binaries at boot. This combination of network-layer and hardware-backed security will satisfy regulatory requirements for patient records or proprietary enterprise data.

- **Energy-Efficient Model Selection:** Mobile and IoT deployments must balance performance against battery life and thermal safety. We intend to measure per-token energy consumption on representative devices under varied quantization settings. Using these measurements, the scheduler can select smaller, lower-precision models for background or low-priority tasks, reserving full-precision variants for interactive sessions. Such energy-aware policies will prolong device uptime and reduce cooling requirements in sealed enclosures.
- **Prospective Use Cases:**
  - *Remote Healthcare Assistants:* Clinics in areas with intermittent Internet could deploy P2PLLMEdge on local workstations and tablets. Patient intake prompts—symptom checkers, medication schedules—would be processed entirely on-site, with sensitive medical histories never leaving the premises. In emergencies, local AI can still provide decision support when cloud connectivity is unavailable.
  - *Disaster-Response Summarization:* First responders often lose cellular signal or suffer bandwidth constraints. A mesh of ruggedized Raspberry Pis and laptops could collect field reports, voice transcripts, and drone imagery, then coordinate to generate concise situation briefs. This on-the-fly summarization accelerates decision-making for command centers without relying on remote data centers.
  - *Smart Agriculture Monitoring:* Soil sensors and weather stations dispersed over large fields can feed data to nearby edge nodes running lightweight LLMs tuned for agronomic advice. By collaboratively interpreting moisture trends, pest alerts, and growth forecasts, P2PLLMEdge clusters can issue irrigation or fertilization advisories even when satellite links are congested or offline.
  - *Rural Education Tutors:* Community learning centers often lack reliable connectivity to online educational platforms. A small cluster of CPU-only devices running P2PLLMEdge could host interactive tutors that answer students' questions in local dialects, generate practice exercises, and provide instant feedback—empowering self-paced learning without constant Internet access.

5. Conclusion

The *P2PLLMEdge* framework represents a pivotal step in realizing decentralized peer-to-peer communication among localized LLMs on resource-constrained edge devices. The results reveal significant efficiency gains in task execution. Peer1, utilizing lightweight LLMs like 'qwen2.5:0.5b-instruct' and 'smolm2:360m-instruct-q8_0', achieved an 85.1% reduction in load duration with the FastAPI configuration compared to Peer2. Meanwhile, Peer2, leveraging more computationally intensive models such as 'Granite3.1-moe:1b' and 'Smolm2:1.7b', demonstrated a superior evaluation capacity, processing up to 423.95 prompts in the most optimized scenario. The framework?s design allows Peer2 to handle more complex summarization tasks, achieving a throughput improvement of up to 56% in token processing rates compared to Peer1. Notably, Peer1's performance metrics highlight its suitability for low-latency and lightweight tasks, such as preprocessing user inputs, with evaluation durations reduced by 92.2% for certain prompt types. On the other hand, Peer2 excelled in handling computationally intensive summarization tasks, showing a 72.8% reduction in evaluation duration for explanation-type prompts compared to Peer1. Additionally, the framework demonstrated robust network communication, with Peer2 achieving a 44.9% reduction in network duration compared to Peer1 in specific configurations.

References

[1] Khalfi, M.F. and Tabbiche, M.N., 2025. GPThingSim: A IoT Simulator Based GPT Models Over an Edge-Cloud Environments. International Journal of Networked and Distributed Computing, 13(1), pp.1-20.

[2] Tharayil, S.M., Krishnapriya, M.A. and Alomari, N.K., 2025. How Multimodal AI and IoT Are Shaping the Future of Intelligence. In Internet of Things and Big Data Analytics for a Green Environment (pp. 138-167). Chapman and Hall/CRC.

[3] Chelliah, A.M.R., Colby, R., Nagasubramanian, G. and Ranganath, S., 2025. 3.2 Edge AI. Model Optimization Methods for Efficient and Edge AI.

[4] Nimmagadda, Y., 2025. Model Optimization Techniques for Edge Devices. Model Optimization Methods for Efficient and Edge AI: Federated Learning Architectures, Frameworks and Applications, pp.57-85.

[5] Martin-Salinas, I., Badia, J.M., Valls, O., Leon, G., del Amor, R., Belloch, J.A., Amor-Martin, A. and Naranjo, V., 2025. Evaluating and accelerating vision transformers on GPU-based embedded edge AI systems. The Journal of Supercomputing, 81(1), p.349.

[6] Yu, D., Zhou, X., Noorian, A. and Hazratifard, M., 2025. An AI-driven social media recommender system leveraging smartphone and IoT data. The Journal of Supercomputing, 81(1), pp.1-32.
[7] Zhang, M., Shen, X., Cao, J., Cui, Z. and Jiang, S., 2024. Edgeshard: Efficient llm inference via collaborative edge computing. IEEE Internet of Things Journal.
[8] Kok, I., Demirci, O. and Ozdemir, S., 2024. When IoT Meet LLMs: Applications and Challenges. arXiv preprint arXiv:2411.17722.
[9] Kalita, A., 2024. Large Language Models (LLMs) for Semantic Communication in Edge-based IoT Networks. arXiv preprint arXiv:2407.20970.
[10] Qu, G., Chen, Q., Wei, W., Lin, Z., Chen, X. and Huang, K., 2024. Mobile edge intelligence for large language models: A contemporary survey. arXiv preprint arXiv:2407.18921.
[11] An, T., Zhou, Y., Zou, H. and Yang, J., 2024. Iot-llm: Enhancing real-world iot task reasoning with large language models. arXiv preprint arXiv:2410.02429.
[12] Hu, Y., Ye, D., Kang, J., Wu, M. and Yu, R., 2024. A Cloud-Edge Collaborative Architecture for Multimodal LLMs-Based Advanced Driver Assistance Systems in IoT Networks. IEEE Internet of Things Journal.
[13] Xiao, B., Kantarci, B., Kang, J., Niyato, D. and Guizani, M., 2024. Efficient Prompting for LLM-based Generative Internet of Things. arXiv preprint arXiv:2406.10382.
[14] Raspberry Pi 4B, 2025. Raspberry Pi 4 Model B Technical Overview. Available at: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/ [Accessed 4 Jan. 2025].
[15] Friha, O., Ferrag, M.A., Kantarci, B., Cakmak, B., Ozgun, A. and Ghoualmi-Zine, N., 2024. Llm-based edge intelligence: A comprehensive survey on architectures, applications, security and trustworthiness. IEEE Open Journal of the Communications Society.
[16] Gao, Y., Song, Z. and Yin, J., 2023. Gradientcoin: A peer-to-peer decentralized large language models. arXiv preprint arXiv:2308.10502.
[17] Karanjai, R. and Shi, W., 2024, May. Trusted LLM Inference on the Edge with Smart Contracts. In 2024 IEEE International Conference on Blockchain and Cryptocurrency (ICBC) (pp. 1-7). IEEE.
[18] He, Y., Fang, J., Yu, F.R. and Leung, V.C., 2024. Large language models (LLMs) inference offloading and resource allocation in cloud-edge computing: An active inference approach. IEEE Transactions on Mobile Computing.
[19] Olshansky, D., Colmeiro, R.R. and Li, B., 2024. Decentralized AI: Permissionless LLM Inference on POKT Network. arXiv preprint arXiv:2405.20450.
[20] Hasan, S.M., Alotaibi, A.M., Talukder, S. and Shahid, A.R., 2024. Distributed Threat Intelligence at the Edge Devices: A Large Language Model-Driven Approach. arXiv preprint arXiv:2405.08755.
[21] Chen, H., Deng, W., Yang, S., Xu, J., Jiang, Z., Ngai, E.C., Liu, J. and Liu, X., 2024. Towards Edge General Intelligence via Large Language Models: Opportunities and Challenges. arXiv preprint arXiv:2410.18125.
[22] Ale, L., Zhang, N., King, S.A. and Chen, D., 2024. Empowering generative AI through mobile edge computing. Nature Reviews Electrical Engineering, pp.1-9.
[23] Bhardwaj, S., Singh, P. and Pandit, M.K., 2024, March. A survey on the integration and optimization of large language models in edge computing environments. In 2024 16th International Conference on Computer and Automation Engineering (ICCAE) (pp. 168-172). IEEE.
[24] Soltoggio, A., Ben-Iwhiwhu, E., Braverman, V., Eaton, E., Epstein, B., Ge, Y., Halperin, L., How, J., Itti, L., Jacobs, M.A. and Kantharaju, P., 2024. A collective AI via lifelong learning and sharing at the edge. Nature Machine Intelligence, 6(3), pp.251-264.
[25] Ollama, 2025. Ollama: Large Language Model Framework. Available at: https://ollama.com/ [Accessed 4 Jan. 2025].
[26] Ollama API, 2025. Ollama API Documentation. Available at: https://github.com/ollama/ollama/blob/main/docs/api.md [Accessed 4 Jan. 2025].
[27] Qwen2.5:0.5b-instruct, 2025. Qwen2.5:0.5b-instruct Language Model. Available at: https://ollama.com/library/qwen2.5:0.5b-instruct [Accessed 4 Jan. 2025].
[28] Smolm2:360m, 2025. Smolm2:360m-instruct-q8_0. Available at: https://ollama.com/library/smollm2:360m-instruct-q8_0 [Accessed 4 Jan. 2025].
[29] Granite3.1, 2025. Granite3.1 Language Models by IBM. Available at: https://github.com/ibm-granite/granite-3.1-language-models [Accessed 4 Jan. 2025].
[30] Llama3.2, 2025. Llama3.2 Language Model. Available at: https://ollama.com/library/llama3.2 [Accessed 4 Jan. 2025].
[31] Qwen2.5:1.5b, 2025. Qwen2.5:1.5b Language Model. Available at: https://ollama.com/library/qwen2.5:1.5b [Accessed 4 Jan. 2025].
[32] Smolm2:1.7b, 2025. Smolm2:1.7b Language Model. Available at: https://ollama.com/library/smollm2 [Accessed 4 Jan. 2025].
[33] Flask, 2025. Flask Web Framework. Available at: https://flask.palletsprojects.com/ [Accessed 4 Jan. 2025].
[34] FastAPI, 2025. FastAPI Framework Documentation. Available at: https://fastapi.tiangolo.com/ [Accessed 4 Jan. 2025].
[35] Requests, 2025. Requests Library for Python. Available at: https://pypi.org/project/requests/ [Accessed 4 Jan. 2025].
[36] Pydantic, 2025. Pydantic for Data Validation and Parsing. Available at: https://pypi.org/project/pydantic/ [Accessed 4 Jan. 2025].
[37] Luo, Z., Yan, H. and Pan, X., 2023. Optimizing Transformer Models for Resource-Constrained Environments: A Study on Model Compression Techniques. Journal of Computational Methods in Engineering Applications, pp.1-12.
[38] Liu HI, Galindo M, Xie H, Wong LK, Shuai HH, Li YH, Cheng WH. Lightweight deep learning for resource-constrained environments: A survey. ACM Computing Surveys. 2024 Jun 24;56(10):1-42.

[39] Girija SS, Kapoor S, Arora L, Pradhan D, Raj A, Shetgaonkar A. Optimizing LLMs for Resource-Constrained Environments: A Survey of Model Compression Techniques. arXiv preprint arXiv:2505.02309. 2025 May 5.

[40] Careem R, Johar G, Khatibi A. Deep neural networks optimization for resource-constrained environments: techniques and models. Indonesian Journal of Electrical Engineering and Computer Science. 2024 Mar;33(3):1843-54.

[41] Waheed Z, Khalid S, Riaz SM, Khawaja SG, Tariq R. Resource-Restricted Environments Based Memory-Efficient Compressed Convolutional Neural Network Model for Image-Level Object Classification. IEEE Access. 2022 Dec 15;11:1386-406.

[42] Shabir MY, Torta G, Damiani F. Edge ai on constrained iot devices: Quantization strategies for model optimization. InIntelligent Systems Conference 2024 Jul 31 (pp. 556-574). Cham: Springer Nature Switzerland.