Comparative Performance Evaluation of GVMP and Edge-Ward Module Placement in Fog Computing

B.Tech. Project (CS 300) Report
submitted in partial fulfilment of the requirements for the award of the degree of

Bachelor of Technology

by

Rudra Garg

(Roll No: 2201170)

Under the supervision of

Dr. Shubha Brata Nath

Department of Computer Science and Engineering
Indian Institute of Information Technology Guwahati
Guwahati, India
April 2025

DECLARATION

I hereby *declare* this project entitled **Comparative Performance Evaluation of GVMP and Edge-Ward Module Placement in Fog Computing** that is being submitted to **Indian Institute of Information Technology Guwahati, Assam**, in partial fulfilment for the requirements of the award of degree of **Bachelor of Technology** in **Computer Science and Engineering** in the department of **Computer Science and Engineering**, is a *genuine report of the work carried out by me*. The material contained in this report has not been submitted at any other University or Institution for the award of any degree.

Rudra Garg, 2201170

....
(Signature of the Student)
Computer Science and Engineering

Place: IIIT Guwahati, Assam
Date: April 9, 2025

Abstract

The proliferation of Internet of Things (IoT) devices challenges traditional cloud computing with latency and network congestion, hindering real-time applications. Fog computing, extending computation to the network edge, offers a solution, but efficient application module placement on heterogeneous fog devices remains an NP-Hard problem. Existing strategies like Edge-Ward Module Placement (EWMP) can underutilize local resources by prioritizing upward shifts to the cloud when edge devices are busy. This project introduces and evaluates Gateway Validation Module Placement (GVMP), a novel, sibling-aware strategy that validates resources on neighboring fog nodes via the gateway before considering cloud offload, aiming to minimize latency and network usage by maximizing local resource utilization.

GVMP's performance was compared against EWMP and Cloud-Only strategies using the iFogSim toolkit for an EEG gaming application simulation, and validated practically using a Docker-based virtual testbed emulating the fog hierarchy and network conditions with Prometheus/Grafana monitoring. Simulation results show GVMP drastically reduces network usage (up to 95% vs. EWMP) while maintaining competitive low latency. virtual testbed results confirm significant E2E latency and CPU load differences based on placement, validating GVMP's effectiveness. The findings demonstrate that GVMP's sibling-aware approach efficiently utilizes local fog resources, offering a more scalable and network-efficient solution for latency-sensitive IoT applications compared to EWMP.

Contents

1 Introduction

1.1 Overview

The Internet of Things (IoT) paradigm is rapidly transforming various aspects of modern life, connecting billions of devices – from wearable sensors and smart home appliances to industrial machinery and vehicles – to the internet [5]. This interconnectivity enables innovative applications in domains like smart cities, healthcare, transportation, and manufacturing. However, the sheer volume of data generated by these devices presents a significant challenge. McKinsey estimates a potential economic impact of $11 trillion annually by 2025, with projections reaching one trillion connected devices [8].

Traditionally, cloud computing has provided the scalable infrastructure required for storing and processing this massive influx of IoT data [3]. Cloud platforms offer virtually unlimited computational and storage resources on demand. However, the centralized nature of the cloud introduces inherent limitations, primarily network latency caused by the physical distance between IoT data sources and remote cloud datacenters. Furthermore, aggregating data streams from millions of devices towards a central point can lead to significant network congestion [4].

For many emerging IoT applications, particularly those requiring real-time analytics and control (e.g., autonomous vehicles, remote patient monitoring, industrial automation, augmented reality games like the one studied in [11]), the latency induced by the cloud round-trip is unacceptable [2]. This has led to the development of the Fog computing paradigm. Fog computing, often used interchangeably with Edge computing, extends cloud services and computation closer to the edge of the network, near the data sources and end-users [6, 1]. By utilizing computational resources available in intermediate devices like gateways, routers, and edge servers (collectively termed fog devices), Fog computing aims to reduce latency, decrease network bandwidth consumption, and improve responsiveness, security, and resilience [4].

1.2 Challenges in Fog Computing

While Fog computing offers compelling advantages, its implementation presents several challenges:

(1) **Resource Heterogeneity:** Fog environments typically consist of a diverse range of devices with varying computational capabilities (CPU, RAM), storage, network bandwidth, and energy constraints [6]. This heterogeneity complicates resource management.

(2) **Module Placement Complexity:** IoT applications are often decomposed into multiple processing modules forming a Directed Acyclic Graph (DAG). Determining the optimal placement of these modules across the available heterogeneous fog devices is a critical and computationally complex problem, identified as NP-Hard [9]. An inefficient placement can negate the benefits of fog computing, leading to high latency or resource bottlenecks.

(3) **Distributed System Management:** Managing a large-scale, geographically distributed infrastructure involving numerous fog devices, sensors, and actuators requires sophisticated monitoring, scheduling, and orchestration mechanisms.

(4) **Scalability and Mobility:** Fog architectures need to scale efficiently as the number of IoT devices grows and handle device mobility, which can change network topology and resource availability dynamically.

1.3 Motivation

The primary motivation for this project stems from the limitations of existing module placement strategies in fully leveraging the potential of Fog computing. While the concept of processing data closer to the edge is established, strategies like the baseline Edge-Ward Module Placement (EWMP), as described in [6], exhibit a key drawback. EWMP prioritizes placing a module on the closest available device; if that device is overloaded, it tends to shift the module directly upwards towards the cloud, potentially bypassing other capable fog devices within the same local network region. This can lead to underutilization of regional fog resources and unnecessary reliance on the distant, higher-latency cloud.

There is a need for a more intelligent placement strategy that considers the broader resource landscape within a local fog domain before resorting to cloud offloading. Such a strategy could further reduce latency and network traffic, enhancing the performance of latency-sensitive IoT applications and improving overall system efficiency. This project proposes and evaluates such a strategy, named Gateway Validation Module Placement (GVMP), initially presented in [9].

1.4 Problem Statement

The objective of this project is to design, implement, and conduct a comparative performance evaluation of a novel IoT application module placement strategy, Gateway Validation Module Placement (GVMP), against the established Edge-Ward Module Placement (EWMP) strategy within heterogeneous Fog computing environments. The primary goals are to assess the effectiveness of GVMP in minimizing end-to-end application latency and reducing overall network usage compared to EWMP.

Specifically, this project aims to:

1. Clearly define and contrast the algorithmic logic of EWMP and GVMP, emphasizing GVMP's sibling fog device resource validation mechanism.
2. Implement both the EWMP and GVMP strategies within the iFogSim simulation toolkit [6].
3. Conduct comparative performance evaluations using iFogSim for a representative latency-sensitive IoT application (EEG Tractor Beam Game [11]) under varying network configurations, focusing on average application latency and network usage metrics.
4. Develop and utilize a virtual testbed environment using Docker containers and network emulation tools to practically validate and compare the performance characteristics (end-to-end latency, CPU utilization) of applications deployed according to EWMP and GVMP-inspired placement logic.
5. Analyze and interpret the results from both simulation and virtual emulation to provide a comprehensive assessment of the relative strengths and weaknesses of the GVMP strategy compared to EWMP.

2 Related Works

The efficient management of resources and placement of application components in Fog and Edge computing environments is a critical area enabling the successful deployment of latency-sensitive and data-intensive IoT applications. This section reviews prior work relevant to this project, focusing on simulation tools for Fog environments and established module placement strategies.

2.1 Fog Computing Simulators: iFogSim

Developing and testing Fog computing systems in real-world settings can be prohibitively expensive and complex due to the scale and heterogeneity involved. Simulation toolkits provide a crucial alternative for evaluating architectures, applications, and resource management policies in a repeatable and controlled manner. Several simulators exist, but **iFogSim** [6] stands out as a widely adopted, open-source toolkit specifically designed for this domain.

Built as an extension of the popular CloudSim framework [3], iFogSim offers specialized abstractions for modeling key Fog/Edge components:

- **Physical Entities:** Allows defining heterogeneous 'FogDevice' instances with specific characteristics (MIPS, RAM, uplink/downlink bandwidth, power consumption models, hierarchical level) along with associated 'Sensor' and 'Actuator' entities connected at the edge. Sensors can have distinct tuple emission rates and patterns.
- **Logical Entities (Applications):** Models applications using a Distributed Data Flow (DDF) approach, represented as a Directed Acyclic Graph (DAG). Vertices are 'AppModule' instances (representing processing units, akin to VMs or containers), and edges ('AppEdge') define data dependencies, tuple types, CPU/network load associated with data transfer, and communication direction (UP/DOWN). iFogSim supports both periodic and event-driven tuple generation along edges.
- **Management Components:** Includes a 'Controller' to manage the simulation lifecycle and gather results, and 'ModuleMapping' / 'ModulePlacement' abstractions to define how application modules are assigned to physical fog devices.
- **Evaluation Metrics:** Facilitates the measurement of critical performance indicators such as end-to-end latency (via 'AppLoop' definitions), network usage, energy consumption, and resource utilization.

The iFogSim toolkit, as detailed in [6] and its tutorial chapter [7], provides the necessary foundation and baseline components (like the EWMP strategy) upon which this project builds and compares the novel GVMP strategy. Its ability to model heterogeneous devices and complex application flows makes it well-suited for evaluating placement algorithms like the ones discussed here.

2.2 Module Placement Strategies

Given a Fog infrastructure and an IoT application decomposed into modules, the module placement strategy determines which fog device will host and execute each module instance. This decision significantly impacts application performance, resource utilization, and network traffic. Several strategies have been explored:

2.2.1 Cloud-Only Placement

This represents the traditional model where computational tasks are offloaded entirely to a centralized cloud. In the context of iFogSim simulations and our evaluation, this typically involves placing sensor/actuator interface modules (like the 'Client' module in the EEG game) on the edge devices (mobiles), while all core processing modules ('$concentration\_calculator$', 'connector') are mapped exclusively to the 'cloud' device instance [6]. This serves as a performance baseline, highlighting the high latency and network usage inherent in centralized processing for edge-generated data.

2.2.2 Edge-Ward Module Placement (EWMP)

EWMP is a heuristic placement strategy provided as a default option within iFogSim [6]. It embodies the core principle of Fog computing: process data as close to the source as possible.

- **Logic:** The algorithm considers devices along the path from the leaf node (e.g., mobile) upwards to the cloud. For an application module ready to be placed, it first attempts placement on the current device being considered.
- **Resource Check:** If the current device possesses sufficient CPU resources (MIPS) to handle the estimated load of the module, the module is placed there.

- **Upward Shift:** If the current device is overloaded, EWMP immediately abandons placement on that device and considers the **parent** device in the hierarchy for placement. It does not explore other devices at the same level.
- **Goal:** Primarily aims to minimize the latency component associated with the initial data transfer from the source/previous module to the current module by keeping placement low in the hierarchy.

While effective in reducing latency compared to Cloud-only, EWMP's limitation is its strictly vertical search for resources. It fails to utilize potentially available resources on sibling devices within the same local region, potentially leading to inefficient resource utilization and unnecessary upward data traffic if the first-choice edge device is busy [9]. Other works also propose resource-aware placements, sometimes focusing on different objectives like energy [10] or using different algorithmic approaches, but EWMP serves as the direct baseline against which GVMP's sibling-awareness is compared.

2.3 Gateway Validation Module Placement (GVMP) Introduction

Addressing the limitation of EWMP's purely vertical placement search, the Gateway Validation Module Placement (GVMP) strategy was proposed [9]. GVMP introduces the concept of 'sibling awareness'. Before shifting a module upwards due to resource constraints on the current device, GVMP initiates a check, typically coordinated by the parent gateway, to determine if any sibling devices (other fog nodes connected to the same parent) have sufficient capacity to host the module. This strategy aims to maximize resource utilization within the local fog region, further reducing the need for cloud offloading and minimizing associated network traffic and latency compared to EWMP. The detailed algorithm and its implementation are discussed in Chapter 4.

3 System Architecture

This chapter describes the architectural components of the system under study, encompassing the conceptual model of the Fog environment, the specific application used for evaluation, and the architectures of both the simulation and virtual testbed environments employed for performance analysis.

3.1 Conceptual Fog Architecture

The project considers a hierarchical Fog computing architecture, commonly deployed in IoT scenarios, extending from edge devices to the central cloud. This multi-tier structure, illustrated conceptually in Fig. 1, facilitates processing closer to data sources.

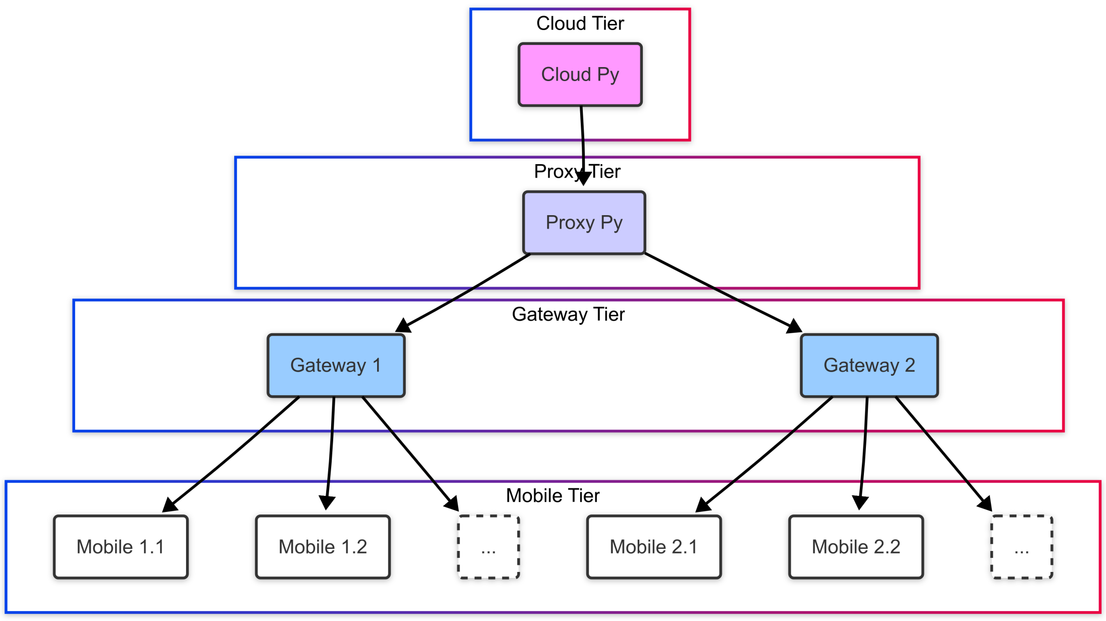

Figure 1: Conceptual Hierarchical Fog Computing Architecture showing tiers from IoT devices to the Cloud.

The architecture consists of the following logical tiers:

- **IoT Device Tier:** Contains sensors generating data (e.g., EEG headsets in our case study) and actuators performing actions (e.g., displays). These are typically resource-constrained.
- **End Device/Mobile Tier:** Devices directly interfacing with sensors/actuators (e.g., smartphones). They perform initial data reception, quality checks, and may host the first level of application modules. These act as the 'leaf' nodes in the placement hierarchy.
- **Gateway/Edge Tier:** Intermediate fog devices aggregating data from multiple end devices (e.g., WiFi routers/gateways). They possess more computational resources than end devices and are primary candidates for hosting computationally intensive application modules within the fog layer. Gateways connect end devices to the upper tiers and play a crucial role in the GVMP strategy by coordinating sibling checks.
- **Proxy Tier (Optional):** A higher-level aggregation point, potentially representing a regional edge server or micro-datacenter, bridging multiple gateways to the cloud. This tier was included in our simulation and testbed configurations.
- **Cloud Tier:** Centralized datacenter offering extensive computational and storage resources. It serves as the final point for data processing if modules cannot be placed lower in the hierarchy or for executing modules requiring global state or large-scale analytics.

Communication primarily occurs vertically between adjacent tiers. GVMP introduces the capability for gateways to probe horizontally among their children (siblings) for resource availability.

3.2 Application Model: EEG Tractor Beam Game

To provide a concrete basis for evaluating placement strategies, we utilize the EEG Tractor Beam Game application scenario, previously modeled in iFogSim [6, 11]. This application simulates a latency-sensitive human-vs-human game involving brain-computer interaction via EEG headsets connected to smartphones.

The application logic involves interactions between several modules, forming a data flow graph as depicted in Fig. 2.

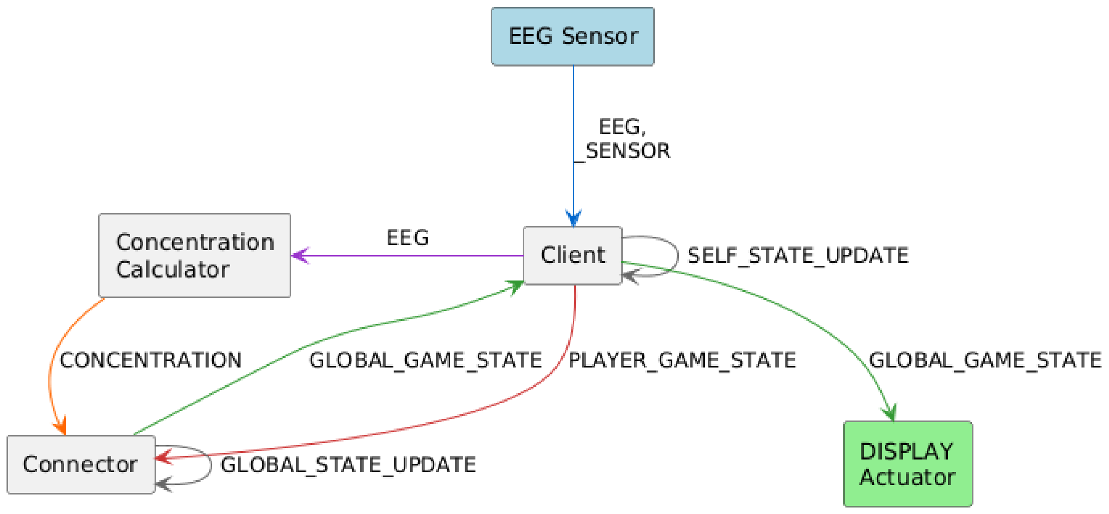

Figure 2: EEG Tractor Beam Game Application Data Flow Graph showing module dependencies and primary data flows (Tuple types: EEG, _SENSOR, CONCENTRATION, PLAYER_GAME_STATE, GLOBAL_GAME_STATE, SELF_STATE_UPDATE, GLOBAL_STATE_UPDATE). Note the cycle between Client and Concentration Calculator.

The application consists of three main software modules:

1. **Client Module (L1):** Typically resides on the end device (smartphone). It receives raw `EEG` data tuples from the sensor, performs initial validation or filtering (outputting an internal `_SENSOR` tuple towards the calculator), processes feedback tuples (`CONCENTRATION` from the calculator, `GLOBAL_GAME_STATE` from the connector) coming down the hierarchy, and sends display update tuples (`SELF_STATE_UPDATE`, `GLOBAL_STATE_UPDATE`) to the local `DISPLAY` actuator.

2. **Concentration Calculator Module (L2):** Processes the filtered sensor data (`_SENSOR`) to compute the player's concentration level based on EEG features (e.g., alpha band power). It outputs `CONCENTRATION` tuples directed back down to the Client module and periodically sends `PLAYER_GAME_STATE` tuples upwards for aggregation.

3. **Connector Module (L3):** Usually placed higher in the hierarchy (Gateway, Proxy, or Cloud). It aggregates `PLAYER_GAME_STATE` tuples from multiple players, maintains the global game state, and periodically sends `GLOBAL_GAME_STATE` tuples downwards to all connected Client modules.

The placement of the `concentration_calculator` (L2) and `connector` (L3) modules is the primary focus of the placement strategies, as their location significantly impacts the latency of the critical user feedback loop (EEG $\rightarrow$ Client $\rightarrow$ Calculator $\rightarrow$ Client $\rightarrow$ Display) and overall network traffic.

3.3 Simulation Environment (iFogSim)

The initial evaluation comparing GVMP and EWMP utilized the iFogSim toolkit [6]. The conceptual architecture and application model described above were mapped onto iFogSim components:

- **Devices:** Instances of the 'FogDevice' class represented Cloud, Proxy, Gateway, and Mobile tiers, configured with specific MIPS, RAM, bandwidth, and power characteristics based on values adapted from [6, 9]. 'Sensor' and 'Actuator' instances were attached to mobile 'FogDevice' instances.

- **Application:** The EEG game was defined using 'Application', 'AppModule' (for Client, Calculator, Connector), and 'AppEdge' classes, specifying tuple types, CPU lengths, network lengths, directionality (UP/DOWN), and selectivity models as per the application logic. 'AppLoop' was defined to measure the critical loop latency.
- **Placement:** Custom placement classes 'ModulePlacementEdgewards.java', 'ModulePlacementGVMP.java' extended iFogSim's 'ModulePlacement' abstract class to implement the respective algorithms. The 'Controller' class managed simulation execution and result aggregation.
- **Scenarios:** Different network scales were simulated by varying the number of gateways 'numOfDepts' parameter in 'VRGameFog.java', allowing evaluation under different system sizes Config-1 to Config-4.

This simulation setup provided a controlled environment for quantitatively comparing the latency and network usage performance of the different placement algorithms based on iFogSim's internal models.

3.4 Virtual Testbed Environment (Docker Emulation)

To complement the simulation study and assess performance under conditions closer to a real deployment (incorporating OS overhead, network stack behavior, resource contention), a virtual testbed was constructed using Docker containers to emulate the Fog hierarchy. Figure 3 shows the high-level structure.

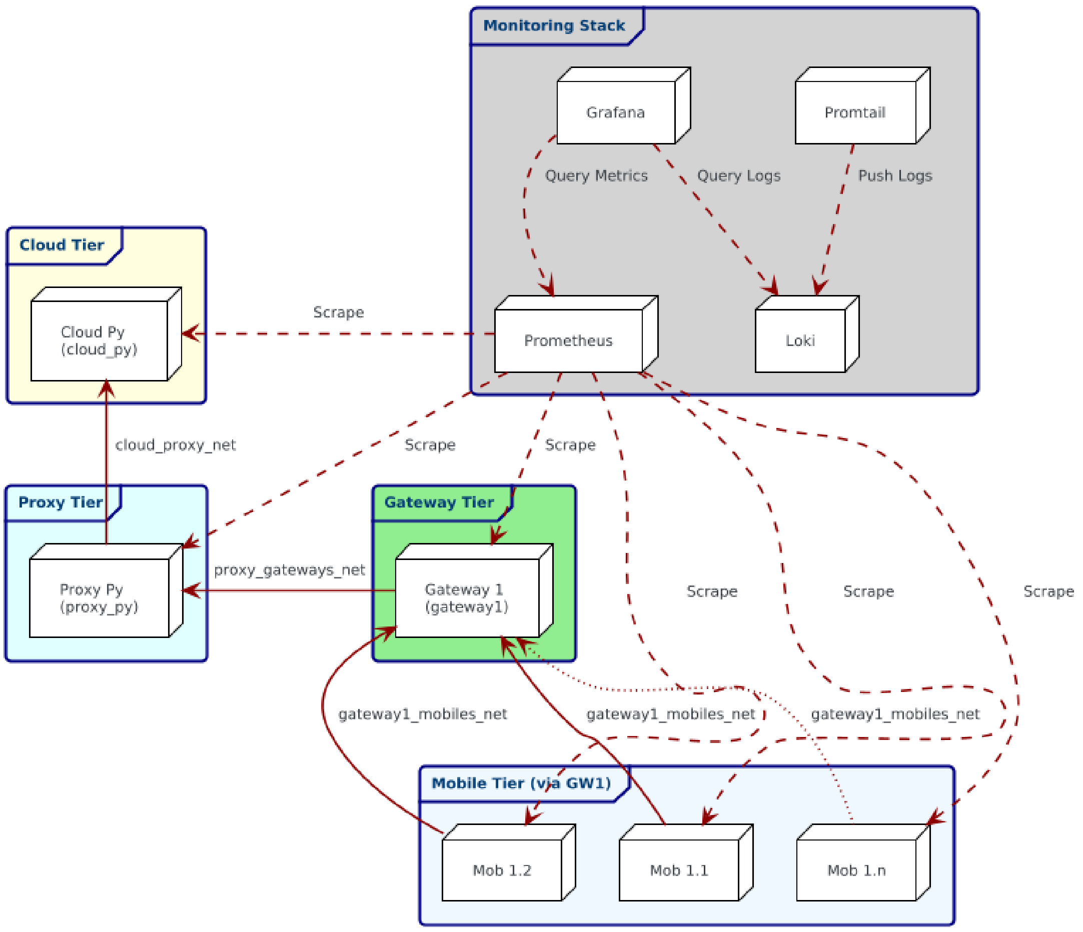

Figure 3: Virtual Testbed Architecture using Docker Containers and Networks.

Key components of the testbed include:

3.4.1 Container Roles and Networking

Docker containers running Python Flask applications represented each entity: multiple 'mobile' instances, 'gateway' instances, a 'proxy_py' instance, and a 'cloud_py' instance. Dedicated Docker networks 'gatewayX_mobiles_net', 'proxy_gateways_net', 'cloud_proxy_net' were configured in the 'docker-compose.yaml' file to restrict communication strictly between adjacent tiers, mirroring the hierarchical topology. Resource limits (CPU, memory) were applied to containers using Docker Compose's 'deploy.resources' section to simulate device heterogeneity.

3.4.2 Application Logic and Shared Modules

The core logic of the EEG application modules (Client, Calculator, Connector) was implemented as reusable Python classes within the 'shared_modules' directory. Each tier's Flask application ('mobile.py', 'gateway.py', 'proxy_app.py', 'cloud_app.py') dynamically loaded and utilized these shared modules based on its configured processing level. Mobile containers used a 'SensorSimulator' class to generate synthetic EEG data. Communication between tiers was handled via HTTP POST requests.

3.4.3 Network Condition Emulation

To simulate realistic network characteristics between tiers, Linux Traffic Control ('tc') commands were employed. Entrypoint scripts ('entrypoint.sh') within the mobile, gateway, and proxy containers read latency, jitter, and packet loss values from environment variables (defined in '.env' and passed via 'docker-compose.yaml'). These scripts then configured 'netem' queueing disciplines on the container's primary network interface to artificially introduce the specified network impairments, allowing for testing under various network quality scenarios.

3.4.4 Configuration and Placement Control

The testbed's behavior, including network conditions and module placement, was controlled via environment variables defined in the '.env' file. The '*_PROCESSING_LEVEL' variables were particularly important, dictating the highest application module level (0=Passthrough, 1=Client, 2=Calculator, 3=Connector) each tier was permitted to execute. The Flask applications read these levels at startup, initializing only the necessary modules. During runtime, they compared the 'last_processed_level' of incoming data with their own effective level to decide whether to process the data locally or forward it upwards. This mechanism allowed for deterministic emulation of different placement strategy outcomes (e.g., forcing L3 completion on Gateway vs. Cloud).

3.4.5 Monitoring Infrastructure

A comprehensive monitoring stack based on Prometheus, Grafana, Loki, and Promtail was deployed alongside the application containers. The Flask applications exposed performance metrics (module execution counts/latency/errors, passthrough counts, E2E latency, gateway RTT, CPU utilization via cgroup scraping) using the 'prometheus-client' library. Prometheus scraped these metrics, Grafana provided dashboard visualizations (see Chapter 5), and Loki/Promtail aggregated container logs, enabling detailed real-time analysis of the system's behavior under different placement configurations.

This testbed architecture provided a practical, controllable, and observable environment for validating the performance implications of module placement strategies beyond pure simulation.

4 Algorithms: Module Placement Strategies

This chapter elaborates on the algorithmic logic of the Edge-Ward Module Placement (EWMP) and Gateway Validation Module Placement (GVMP) strategies evaluated in this project. Both strategies aim to map application modules to fog devices within a hierarchical structure, but they differ significantly in how they handle resource constraints at the edge.

4.1 Edge-Ward Module Placement (EWMP)

EWMP is the baseline heuristic provided by iFogSim [6], designed to minimize latency by placing modules as close to the data source (edge) as possible. It follows a strictly vertical, bottom-up approach for placement decisions when encountering resource limitations.

Algorithm 4.1: Edge-Ward Module Placement (EWMP) Logic

```
Input: Application $App$, Set of FogDevices $F$, Leaf-to-root Paths $P$, Initial ModuleMapping $M_0$
Output: Final Module Placement Map $M_{final}$
1  Initialize $M_{current} \leftarrow M_0$
2  Initialize CurrentLoads $L$ for all $f \in F$ to 0
3  foreach path $p \in P$ do
4      Let $placedModules_p$ be the set of modules already placed along path $p$ (initially empty or
         based on $M_0$ for sensors/actuators)
5      foreach FogDevice $d$ in path $p$ (from leaf to root) do
6          Let $readyModules \leftarrow$ GetModulesReadyToPlace($App$, $placedModules_p$)
7          foreach AppModule $m$ in $readyModules$ do
8              if $m$ is already placed at device $d_{upstream} \in p$ where level($d_{upstream}$) > level($d$) then
9                  Continue to next module
10             end
11             $requiredLoad \leftarrow$ CalculateModuleLoad($m$, $App$, CurrentRates)
12             if $L[d] + requiredLoad \leq MaxMIPS(d)$ then
13                 PlaceModuleOnDevice($m$, $d$, $M_{current}$)
14                 $L[d] \leftarrow L[d] + requiredLoad$
15                 Add $m$ to $placedModules_p$
16                 Remove $m$ from $readyModules$
17             end
18         end
19     end
20 end
21 $M_{final} \leftarrow M_{current}$
22 return $M_{final}$
```

Key Characteristics:

- **Bottom-Up:** Considers devices starting from the edge.
- **Greedy:** Attempts to place a module on the first suitable device encountered moving upwards.
- **Vertical Only:** If a device lacks resources, the decision is deferred to its parent device without checking horizontal alternatives (siblings).

This can lead to situations where a module is pushed towards the cloud even if a nearby sibling device at the same fog level has ample capacity.

4.2 Gateway Validation Module Placement (GVMP)

GVMP, proposed in [9], enhances EWMP by introducing a 'sibling validation' step before resorting to upward placement. This aims to maximize resource utilization within the local fog region.

Algorithm 4.2: Gateway Validation Module Placement (GVMP) Logic

```
Input: Application $App$, Set of FogDevices $F$, Leaf-to-root Paths $P$, Initial ModuleMapping $M_0$
Output: Final Module Placement Map $M_{final}$
Initialize $M_{current} \leftarrow M_0$
Initialize CurrentLoads $L$ for all $f \in F$ to 0
foreach $path\ p \in P$ do
    Let $placedModules_p$ be the set of modules placed along path $p$
    foreach FogDevice d in path p (from leaf to root) do
        Let $readyModules \leftarrow$ GetModulesReadyToPlace ($App$, $placedModules_p$)
        foreach AppModule m in readyModules do
            if $m$ is already placed at $d_{upstream} \in p$ then
                Continue
            end
            $requiredLoad \leftarrow$ CalculateModuleLoad ($m$, $App$, CurrentRates)
            if $L[d] + requiredLoad \leq MaxMIPS(d)$ then
                PlaceModuleOnDevice ($m$, $d$, $M_{current}$)
                $L[d] \leftarrow L[d] + requiredLoad$
                Add $m$ to $placedModules_p$
            end
            else
                $siblingId \leftarrow$ ValidateGateway ($d$, $m$, $requiredLoad$, $L$)
                if $siblingId \neq -1$ then
                    PlaceModuleOnDevice ($m$, GetDeviceById($siblingId$), $M_{current}$)
                    $L[siblingId] \leftarrow L[siblingId] + requiredLoad$
                    Add $m$ to $placedModules_p$
                    /* Module placed on sibling, not shifted north */
                end
                else
                    $placedModules_p$.add(ShiftModuleNorth ($m$, $requiredLoad$, $d$, $L$, $M_{current}$))
                    /* ShiftNorth returns modules actually placed upwards */
                end
            end
            Remove $m$ from $readyModules$ after attempting placement
        end
        $readyModules \leftarrow$ GetModulesReadyToPlace ($App$, $placedModules_p$)
    end
end
$M_{final} \leftarrow M_{current}$
return $M_{final}$
```

Algorithm 4.3: GVMP: ValidateGateway Function Pseudocode

```
Function ValidateGateway(currentDevice d, module m, requiredLoad $l_m$, CurrentLoads L):
    parentId ← GetParentDevice (d.id)
    if parentId $= -1$ then
        return $-1$
    end
    parentDevice ← GetDeviceById (parentId)
    foreach siblingId in GetChildDevices (parentDevice) do
        if siblingId $= d.id$ then
            Continue
        end
        siblingDevice ← GetDeviceById (siblingId)
        if $L[siblingId] + l_m \leq MaxMIPS(siblingDevice)$ then
            return siblingId
        end
    end
    return $-1$
```

4.2.1 Core Logic The core logic mirrors EWMP's bottom-up traversal (Lines 6-9 in Algorithm 4.2). The key difference lies in the 'else' block starting at Line 17.

4.2.2 Sibling Validation When a module m cannot be placed on the current device d (Line 17), GVMP calls the 'ValidateGateway' function (Line 18, detailed in Algorithm 4.3). This function identifies the parent (gateway) of d and iterates through d's siblings (other children of the parent). For each sibling, it checks if adding the module's required load (l_m) would exceed the sibling's capacity ('MaxMIPS'). If a suitable sibling is found, its ID is returned.

4.2.3 Hierarchical Preference The placement decision follows a clear hierarchy:

1. **Local Placement:** If the current device d has resources, place the module there (Line 12).

2. **Sibling Placement:** If local placement fails, but 'ValidateGateway' finds a suitable sibling (Line 19), place the module on that sibling.

3. **Upward Placement/Shift:** Only if both local and sibling placements fail (Line 24), initiate the upward shift ('ShiftModuleNorth', Line 25). This function attempts placement on the parent, and potentially higher ancestors, potentially displacing other modules upwards as needed, similar to the mechanism in EWMP but invoked less frequently.

This structured approach ensures that local and regional resources are prioritized, aiming to reduce latency and network load compared to the strictly vertical approach of EWMP. The 'ShiftModuleNorth' function itself (not detailed in pseudocode here, but present in the provided Java code) handles the recursive upward movement and potential cascading shifts if parent nodes are also overloaded.

5 Experimentation and Results

This chapter details the experimental methodology and presents the results obtained from evaluating the proposed Gateway Validation Module Placement (GVMP) strategy against the baseline Edge-Ward Module Placement (EWMP) and Cloud-Only approaches. Evaluations were conducted using both the iFogSim simulation toolkit and a virtual testbed environment built with Docker containers to provide a comprehensive performance analysis.

5.1 Experimental Setup

5.1.1 Simulation Setup (iFogSim)

The iFogSim toolkit [6] was used for initial performance comparisons based on its simulation models.

- **Application:** The EEG Tractor Beam Game application (described in Section 3.2) was implemented.
- **Topology:** A hierarchical Fog-Cloud structure (Mobile $\rightarrow$ Gateway $\rightarrow$ Proxy $\rightarrow$ Cloud) was modeled. Device resources (MIPS, RAM, etc.) were based on standard configurations from [6, 9].
- **Configurations:** Scalability was tested using four configurations, varying the number of gateways and associated mobile devices, as detailed in Table 1.
- **Placement Strategies:** Three strategies were simulated: (1) Cloud-Only, (2) EWMP (Edge), and (3) GVMP. Specific module mappings for fixed components (Client, Connector) were applied as described in 'VRGameFog.java'.
- **Execution:** Simulations were run for each configuration-strategy pair, managed by 'VRGameFogAutomation.java'.

Table 1: iFogSim Simulation Configurations.

Configuration	Num Gateways ('numOfDepts')	Total Mobile Devices
Config-1	2	10
Config-2	4	20
Config-3	5	25
Config-4	6	30

5.1.2 Virtual Testbed Setup (Docker)

A virtual testbed environment was constructed using Docker to emulate the fog hierarchy and enable measurements under more realistic conditions.

- **Architecture:** As described in Section 3.4 and illustrated in Fig. 3, the testbed comprised Docker containers for Mobile (8 instances, 4 per gateway), Gateway (2 instances), Proxy (1 instance), and Cloud (1 instance) tiers, connected via dedicated Docker networks.
- **Application & Logic:** Python Flask applications ran within containers, utilizing shared modules for Client, Calculator, and Connector logic. Mobile containers simulated EEG data generation.
- **Resource Constraints:** CPU and memory limits were enforced on containers via 'docker-compose.yaml' (e.g., Mobile: 0.05 CPU, 256MB; Gateway/Proxy: 0.1 CPU, 512MB; Cloud: 1 CPU, 1G).
- **Network Emulation:** The Linux 'tc' command, executed via container entrypoint scripts, introduced configurable latency, jitter, and packet loss between tiers (e.g., Mobile$\leftrightarrow$GW: 50ms $\pm$ 5ms; GW$\leftrightarrow$Proxy: 100ms $\pm$ 10ms; Proxy$\leftrightarrow$Cloud: 300ms $\pm$ 30ms delay/jitter).
- **Placement Control:** Module placement was controlled by setting '*_PROCESSING_LEVEL' environment variables in the '.env' file, forcing different tiers to perform the final (L3) processing step.
- **Monitoring:** A Prometheus/Grafana/Loki stack collected and visualized metrics and logs from all application containers. Experiments involved running specific placement scenarios and observing the Grafana dashboards over a 15-minute interval.

5.2 Performance Metrics

The following key performance indicators were measured and analyzed:

- **Average Application Loop Latency (Simulation):** The average time taken for the critical user feedback loop (EEG $\rightarrow$ Client $\rightarrow$ Calculator $\rightarrow$ Client $\rightarrow$ Display) as reported by iFogSim. Lower values indicate better responsiveness.
- **Network Usage (Simulation):** The total simulated data volume transferred across all network links during the simulation run, reported by iFogSim. Lower values indicate better network efficiency.
- **End-to-End (E2E) Latency (Virtual Testbed):** The wall-clock time measured from data packet creation time ('creation_time' timestamp added by mobile) until the completion of L3 processing (Connector module finish time), wherever it occurs. Reported as p95 (95th percentile) and average values. Measured via Prometheus histogram 'e2e_processing_latency_seconds'.
- **CPU Utilization (Virtual Testbed):** The percentage of allocated CPU resources utilized by each container, normalized against its defined CPU limit (0-100%). Measured via Prometheus gauge 'cpu_utilization_percent' derived from cgroup statistics.
- **Module Performance (Virtual Testbed):** Execution rate (throughput), execution latency (p95), and error counts for each application module (Client, Calculator, Connector) on the tier where it executed. Measured via Prometheus counters and histograms 'module_*_total' and 'module_execution_latency_seconds'.
- **Final Processing Location (Virtual Testbed):** The distribution of L3 (Connector) module executions across the different tiers (Mobile, Gateway, Proxy, Cloud), indicating the outcome of the placement decision. Visualized as a pie chart derived from 'module_executions_totalmodule='connector''.

5.3 Simulation Results (iFogSim)

The iFogSim simulations provided quantitative comparisons of the placement strategies based on the simulator's models.

5.3.1 Average Application Latency

The latency of the critical application loop is crucial for user experience. Table 2 summarizes the average results, visualized in Fig. 4.

Table 2: iFogSim Simulation Results: Average Application Latency (ms).

Configuration	Cloud-Only	EWMP (Edge)	GVMP
Config-1	225.37	13.32	16.11
Config-2	225.69	13.09	16.42
Config-3	318.61	13.07	16.27
Config-4	557.09	13.13	16.30

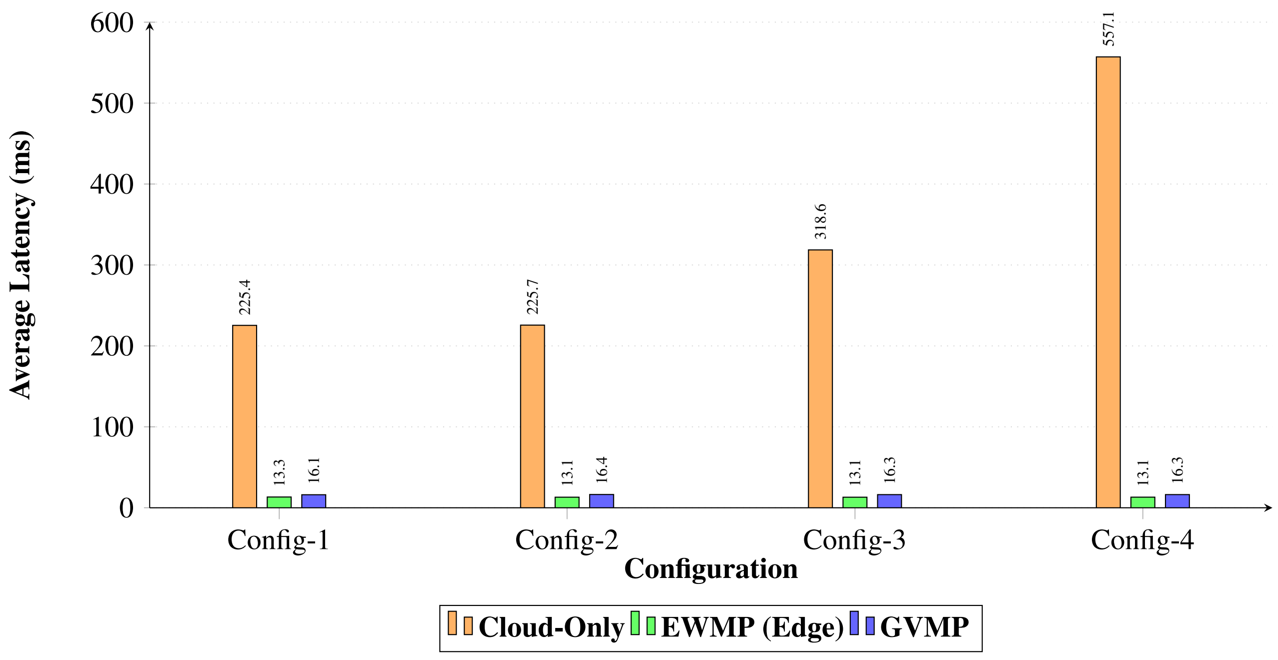

Figure 4: **iFogSim Simulation: Average Application Loop Latency Comparison.** The chart highlights the significant latency reduction achieved by both EWMP and GVMP deployment strategies compared to traditional Cloud-Only approach, with EWMP showing the lowest overall latency.

Analysis: The results clearly show the latency disadvantage of the Cloud-Only approach. Both EWMP and GVMP significantly outperform Cloud-Only, achieving low average loop latencies (around 13-16ms). EWMP shows marginally lower average latency than GVMP in these specific simulations, likely due to its immediate placement on the first available edge node in non-overloaded cases. However, the difference between EWMP and GVMP is small relative to the massive improvement over Cloud-Only.

5.3.2 Network Usage

Network usage reflects the efficiency of data transmission within the architecture. Table 3 and Fig. 5 present the total network usage results.

Table 3: iFogSim Simulation Results: Network Usage (Bytes) and GVMP Reduction vs EWMP.

Configuration	Cloud-Only (Bytes)	EWMP (Bytes)	GVMP (Bytes)	GVMP % Reduction vs EWMP
Config-1	196707.5	11272.0	1202.0	89.33%
Config-2	434867.5	21594.0	1454.0	93.27%
Config-3	555885.5	26755.0	1580.0	94.09%
Config-4	645372.5	31916.0	1706.0	94.65%

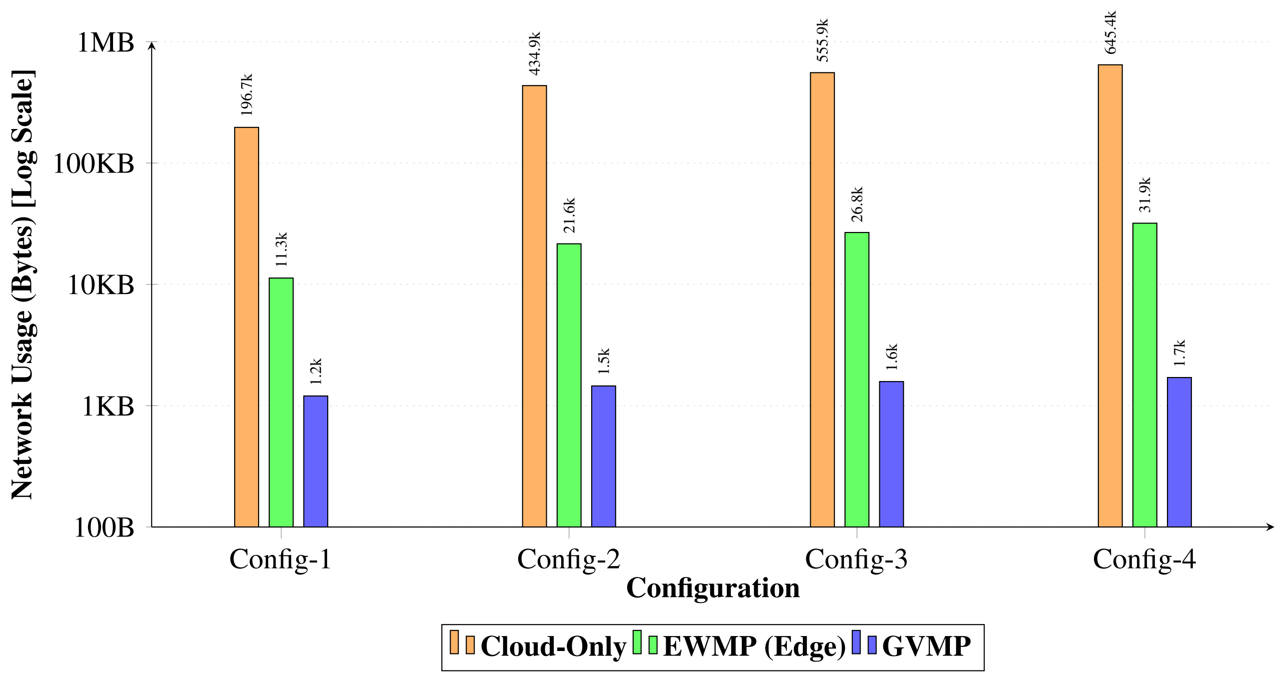

Figure 5: **iFogSim Simulation: Network Usage Comparison (Logarithmic Scale).** Network usage shown in Bytes on the y-axis, with data labels displayed in KiloBytes (kB). The logarithmic scale highlights the orders-of-magnitude reduction achieved by EWMP and especially GVMP compared to Cloud-Only.

Analysis: The network usage results highlight the primary strength of GVMP. While EWMP offers substantial savings over Cloud-Only, **GVMP achieves a dramatic further reduc tion in network traffic compared to EWMP (89-95% reduction)**. This is a direct consequence of GVMP's sibling validation mechanism, which successfully places modules within the local fog region more often, minimizing data flow towards the higher tiers (Proxy and Cloud). This efficiency becomes increasingly important as the number of devices (and thus potential network load) increases.

5.4 Virtual Testbed Results (Docker/Grafana)

The Docker-based virtual testbed, incorporating network emulation and resource constraints, allowed for validating performance trends in a more realistic setting. Different placement scenarios were emulated by configuring the maximum processing level allowed on each tier via environment variables. The results were captured using the Prometheus/Grafana monitoring stack.

5.4.1 End-to-End Latency Analysis

The End-to-End (E2E) latency, measured from data creation on the mobile device to the completion of L3 processing (Connector module), is a critical indicator of user-perceived responsiveness. Table 4 summarizes the approximate steady-state p95 and average E2E latencies observed for different final processing tiers, and Fig. 6 visualizes this data.

Table 4: Virtual Testbed: Approximate E2E Latency Summary (ms).

Final Processing Tier (L3 Location)	Approx. p95 Latency (ms)	Approx. Average Latency (ms)
Mobile	16.1	8.65
Gateway	482	230
Proxy	3640	845
Cloud	5600	1580

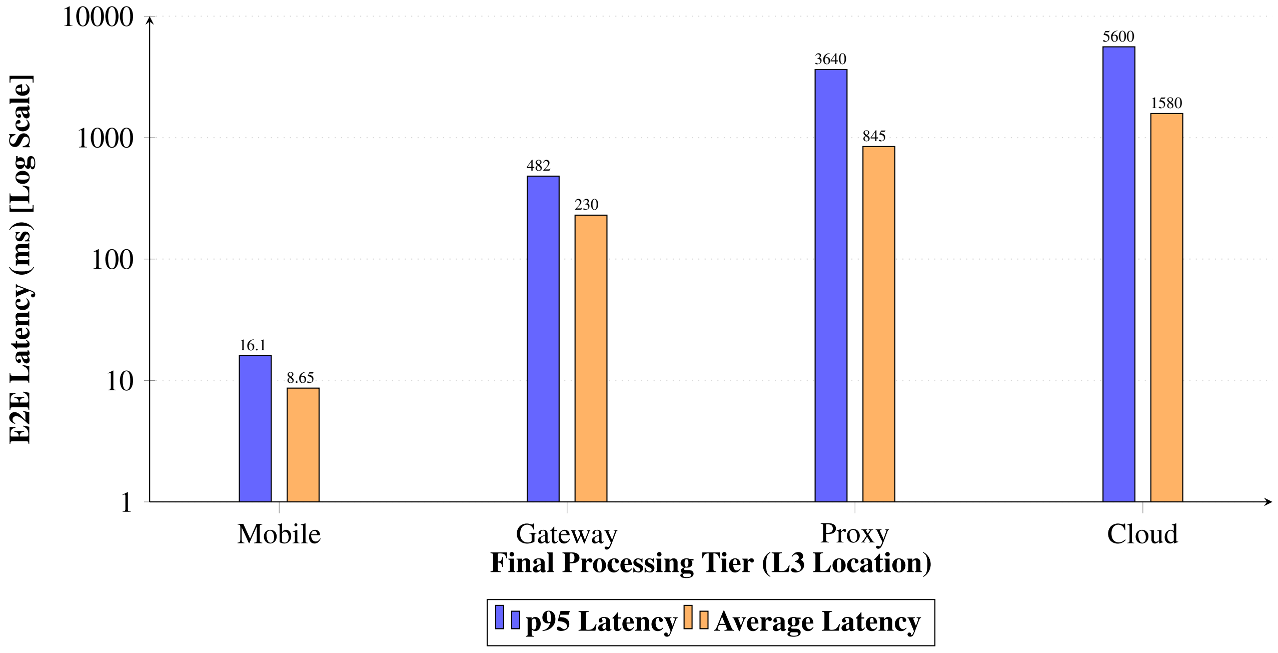

Figure 6: **Virtual Testbed: End-to-End Latency Comparison (Logarithmic Scale).** E2E latency increases by approximately two orders of magnitude as processing moves from Mobile (8-16ms) to Cloud (1580-5600ms). The logarithmic scale highlights both the dramatic increase and the widening gap between average and p95 latency as distance from the edge increases.

5.4.2 CPU Utilization Analysis

Monitoring CPU utilization reveals how the computational workload of the EEG application is distributed across the different tiers of the emulated Fog hierarchy, depending on the module placement configuration. Table 5 presents approximate average CPU utilization percentages observed in the virtual testbed under four distinct scenarios, each forcing the completion of the entire application pipeline (up to L3 - Connector) at a different tier. These scenarios were configured using the '*_PROCESSING_LEVEL' environment variables. Figure 7 visualizes this load distribution.

Table 5: Virtual Testbed: Approximate Average CPU Utilization (%) under Different Placement Scenarios.

Scenario (L1-L3 Completed On)	Mobile CPU (%)	Gateway CPU (%)	Proxy CPU (%)	Cloud CPU (%)
Mobile	95%	5%	5%	<1%
Gateway	20%	60%	5%	<1%
Proxy	20%	10%	60%	<1%
Cloud	20%	10%	10%	5%

Note: Values are estimations based on Grafana observations. CPU percentages reflect load relative to container limits.

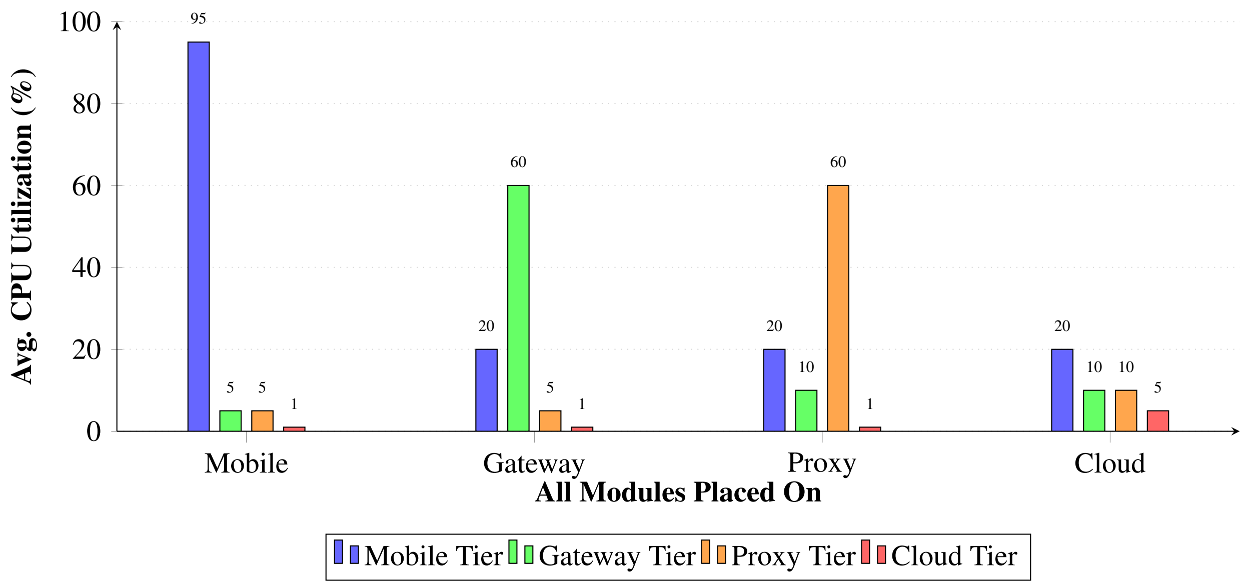

Figure 7: **Virtual Testbed: CPU Utilization Comparison.** Average CPU load distribution across tiers under different scenarios, indicating where the primary computational effort (up to L3) occurred.

Analysis: The CPU utilization results (Fig. 7, Table 5) clearly illustrate how the computational load shifts across the Fog hierarchy based on module placement.

- When the entire application pipeline (L1-L3) is configured to run on the **Mobile** tier, the mobile containers exhibit very high CPU utilization (approx. 95%), indicating they handle the bulk of the processing (Client, Calculator, and Connector tasks), while upstream tiers (Gateway, Proxy, Cloud) remain mostly idle.

- When processing is allowed up to the **Gateway** tier (L1 , L2 & L3 on Gateway), the Gateway containers show significantly increased CPU load (approx. 60%), while the Mobile containers handle a moderate load (Sensor simulation, approx. 20%), and Proxy/Cloud remain low.

- Similarly, forcing L3 completion onto the **Proxy** concentrates the load there (approx. 60%), with Mobile and Gateway handling sensor simulation and request passthrough respectively.

- In the **Cloud** completion scenario (emulating Cloud-Only or overflow), the edge tiers (Mobile, Gateway, Proxy) exhibit lower utilization as they primarily forward data, while the Cloud container shows increased (though still moderate in this test) utilization corresponding to the final processing steps.

These results highlight the direct impact of placement decisions on resource consumption at each tier. Strategies like GVMP, which aim to utilize Gateway and sibling resources before resorting to the Proxy or

Cloud, inherently promote better load distribution within the fog layer compared to EWMP, which might overload specific initial edge nodes or shift load vertically towards the cloud prematurely. Efficient CPU utilization across the fog infrastructure is crucial for scalability and avoiding performance bottlenecks.

5.5 Discussion

The experimental results from both the iFogSim simulation and the Docker-based virtual testbed provide compelling evidence supporting the effectiveness of the GVMP strategy.

The primary advantage of GVMP, dramatically highlighted by the iFogSim results (Table 3, Fig. 5), is its significant reduction in network usage compared to EWMP. By intelligently checking sibling device availability before resorting to upward placement, GVMP minimizes data transfers to higher, more distant tiers like the proxy and cloud. This reduction (89-95%) is critical for alleviating network congestion, improving scalability, and potentially lowering data transmission costs in large-scale IoT deployments.

Regarding latency, while the iFogSim **average loop latency** (Table 2, Fig. 4) showed EWMP performing marginally better in some static scenarios, the virtual testbed's **end-to-end latency** measurements (Table 4, Fig. 6) clearly illustrate the substantial time penalty associated with each upward network hop required when local resources are exhausted. GVMP's mechanism is designed to avoid these costly hops whenever a suitable local alternative (a sibling) exists. Therefore, in dynamic environments with fluctuating loads where EWMP might prematurely push tasks to the cloud, GVMP is expected to deliver better average E2E performance by keeping tasks within the lower-latency fog region more consistently.

The CPU utilization results from the testbed (Table 5, Fig. 7) visually confirm the load distribution effects. GVMP inherently promotes better utilization of the collective resources within a local fog cluster (device + siblings + gateway) compared to EWMP's strictly vertical approach. This prevents bottlenecks on single edge devices and reduces unnecessary load on the central cloud.

The concordance between the trends observed in simulation (especially network usage reduction) and the virtual testbed (latency impact of placement, CPU load distribution) strengthens the conclusions. The testbed adds a layer of practical validation, incorporating factors like OS/container overhead and real network stack behavior under emulated conditions, confirming that the benefits suggested by simulation translate into measurable effects in a more realistic environment.

Overall, the experiments demonstrate that GVMP offers a superior balance between latency optimization and network/resource efficiency compared to EWMP for heterogeneous fog environments, particularly as system scale increases. Its sibling-aware approach represents a significant improvement for deploying demanding IoT applications effectively.

6 Conclusion and Future Work

6.1 Conclusion

This project successfully addressed the challenge of efficient module placement in IoT-Fog environments by proposing and evaluating the Gateway Validation Module Placement (GVMP) strategy. GVMP enhances the baseline Edge-Ward Module Placement (EWMP) by incorporating a novel "sibling-aware" mechanism, where gateways validate resource availability on neighboring fog nodes before considering upward placement. Through comparative evaluations using both iFogSim simulation and a Docker-based virtual testbed, we demonstrated GVMP's effectiveness for a latency-sensitive EEG application.

Key findings confirmed that GVMP significantly reduces network usage (up to 95% vs. EWMP) by prioritizing local and sibling resources, thus minimizing traffic to higher tiers. While maintaining competitive low latency comparable to EWMP in static simulations, GVMP's logic is better positioned to optimize end-to-end latency in dynamic scenarios by avoiding unnecessary, high-latency upward shifts. Testbed results further validated GVMP's benefits in practical E2E latency and more balanced CPU load distribution within the fog layer. In conclusion, GVMP offers a superior balance of network efficiency and latency optimization, representing a promising, scalable, and resource-aware solution for deploying demanding IoT applications in heterogeneous Fog computing infrastructures.

6.2 Future Work

While this project successfully demonstrated the advantages of GVMP, several avenues exist for future research and enhancement:

1. **Dynamic Adaptation and Migration:** Extend GVMP to incorporate dynamic runtime adjustments. Implement mechanisms to monitor real-time device load and network conditions, enabling adaptive module migration based on GVMP's principles (e.g., migrating from an overloaded sibling to another available one or to the parent gateway) after initial placement.

2. **Energy Efficiency Analysis:** Integrate energy consumption models into both the iFogSim evaluation and the virtual testbed analysis (potentially through estimations based on CPU load and device power profiles). Evaluate the energy trade-offs associated with placing modules on different fog device types (e.g., low-power gateways vs. more powerful edge servers).

3. **Advanced Sibling Selection:** Enhance the 'ValidateGateway' function. Instead of selecting the first available sibling, incorporate more sophisticated criteria, such as selecting the sibling with the lowest current load, lowest network latency from the original device, or specific required hardware capabilities (beyond just CPU).

4. **General Topology Support (e.g., Mesh):** Adapt the GVMP strategy, particularly the sibling validation and upward shifting logic, to function effectively in non-hierarchical or more complex network topologies, such as mesh networks where devices may have multiple neighbors or parents. This would involve discovering relevant nearby nodes based on network proximity and latency rather than just strict parent-child relationships.

5. **Real-Hardware Testbed:** Validate GVMP on a physical testbed composed of actual edge hardware (e.g., Raspberry Pis, Jetson Nanos) to assess its performance under real-world network conditions, operating system behavior, and hardware limitations, comparing these results against the simulation and Docker emulation findings.

Addressing these areas would further solidify GVMP's applicability and robustness for practical Fog computing deployments.

References

[1] Flavio Bonomi, Rodolfo Milito, Preethi Natarajan, and Jiang Zhu. Fog computing: A platform for internet of things and analytics. In N. Bessis and C. Dobre, editors, *Big Data and Internet of Things: A Roadmap for Smart Environments*, pages 169–186. Springer, 2014.

[2] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile Cloud computing (MCC '12)*, pages 13–16, 2012.

[3] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, and Rajkumar Buyya. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience (SPE)*, 41(1):23–50, 2011.

[4] Amir Vahid Dastjerdi and Rajkumar Buyya. Fog computing: Helping the internet of things realize its potential. *Computer*, 49(8):112–116, 2016.

[5] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660, 2013.

[6] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K. Ghosh, and Rajkumar Buyya. iFogSim: A toolkit for modeling and simulation of resource management techniques in internet of things, edge and fog computing environments. *Software: Practice and Experience (SPE)*, 47(9):1275–1296, 2017.

[7] Redowan Mahmud and Rajkumar Buyya. Modelling and simulation of fog and edge computing environments using iFogSim toolkit. In Rajkumar Buyya and Satish Narayana Srirama, editors, *Fog and Edge Computing: Principles and Paradigms*, chapter 17. Wiley, 2019.

[8] James Manyika, Michael Chui, Peter Bisson, Jonathan Woetzel, Richard Dobbs, Jacques Bughin, and Dan Aharon. The internet of things: Mapping the value beyond the hype. Technical report, McKinsey Global Institute, June 2015. Accessed: 2025-04-07.

[9] Sanjaya Kumar Panda, Divyanshu Tyagi, and Rashmi Ranjan Rout. GVMP: A novel internet of things application module placement strategy for heterogeneous infrastructure using iFogSim. In *2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT)*. IEEE, July 2024. Accessed: 2025-04-07.

[10] Mohit Taneja and Alan Davy. Resource aware placement of iot application modules in fog-cloud computing paradigm. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 1222–1228, 2017.

[11] John K. Zao, Tchin Tze Gan, Chun Kai You, Sergio José Rodríguez Méndez, Cheng En Chung, Yu Te Wang, Tim Mullen, and Tzyy Ping Jung. Augmented brain computer interaction based on fog computing and linked data. In *2014 International Conference on Intelligent Environments (IE)*, pages 374–377, 2014.