

**Multi-Objective Privacy-Preserving Function Placement in an Edge-Cloud
FaaS Marketplace**

Tusa, F., Griffin, D., Kiss, T. and Rio, M.

This is a copy of the author's accepted version of a paper subsequently published in the proceedings of ICCCN 2026 - 35th International Conference on Computer Communications and Networks, Honolulu, Hawaii, USA, 27 - 30 Jul 2026.

The final published version will be available online at:

<https://ieeexplore.ieee.org/>

© 2026 IEEE . This manuscript version is made available under the CC-BY 4.0 license

<https://creativecommons.org/licenses/by/4.0/>

For the purpose of open access, the author(s) has applied a Creative Commons Attribution (CC BY) license to any Accepted Manuscript version arising.

The WestminsterResearch online digital archive at the University of Westminster aims to make the research output of the University available to a wider audience. Copyright and Moral Rights remain with the authors and/or copyright owners.

Multi-Objective Privacy-Preserving Function Placement in an Edge-Cloud FaaS Marketplace

Francesco Tusa*, David Griffin[†], Tamas Kiss*, and Miguel Rio[†]

*School of Computer Science and Engineering, University of Westminster, London, United Kingdom

Email: {f.tusa, t.kiss}@westminster.ac.uk

[†]Department of Electronic and Electrical Engineering, University College London, London, United Kingdom

Email: {d.griffin, miguel.rio}@ucl.ac.uk

Abstract—The shift toward modular microservices has driven the success of serverless Function-as-a-Service (FaaS) as a way to offload infrastructure management. However, meeting the diverse needs of emerging applications—from ultra-low latency control to budget-constrained processing—requires FaaS to transcend centralized clouds and span the edge-cloud continuum. Yet, this introduces two challenges: orchestrating diverse user demands across heterogeneous capabilities and protecting sensitive requests from the underlying infrastructure. We propose a Privacy-Preserving FaaS Marketplace using decentralized orchestration agents. Providers autonomously trade function offers, enabling multi-objective placement that dynamically satisfies strict user QoS constraints. Crucially, Paillier homomorphic encryption blinds function identifiers during resolution to preserve client privacy. Simulations demonstrate our system improves on traditional latency-minimizing FaaS gateways by optimizing placement and proactively shielding clients from SLA violations. Results validate seamless orchestration in the Edge-Cloud continuum while enabling key trade-offs between agent state aggregation, function placement accuracy, and network overhead.

Index Terms—Decentralized Orchestration, Edge-Cloud Continuum, Homomorphic Encryption, Multi-Objective Function Placement, Privacy-Preserving FaaS.

I. INTRODUCTION

The widespread adoption of Serverless Computing and the Function-as-a-Service (FaaS) model [1] allows developers to offload infrastructure provisioning to cloud providers, focusing entirely on writing granular event-driven business logic. However, as real-time IoT and emerging applications demand ultra-low latency, computational resources must expand beyond centralized datacenters to span the Edge-Cloud continuum [2], [3]. Deploying fine-grained serverless functions across a highly distributed environment introduces orchestration challenges [4]. Today’s static, centralized orchestration engines or latency-based gateways are ill-equipped to handle the dynamic availability of resources across several providers, the network volatility of the continuum, or the diverse Quality of Service (QoS) requirements of different clients [5].

To address the performance and coordination challenges of a multi-provider Edge-Cloud continuum, we argue that the natural evolution is toward an open, decentralized marketplace. By providing economic incentives for independent providers to contribute idle edge capacity, this ecosystem naturally breaks vendor lock-in and reduces fragmentation.

Rather than relying on a centralized scheduler, the marketplace utilizes decentralized brokers to evaluate autonomous provider advertisements against multi-objective client constraints and execution costs, optimizing both *infrastructural yield* and *economic efficiency* across the continuum. However, utilizing such untrusted, multi-provider infrastructures to handle FaaS workloads introduces a severe privacy paradox [6]. Exposing fine-grained function requests in plaintext allows marketplace stakeholders to monitor invocation rates and sequences. Analyzing this metadata can reveal sensitive business workflows or private user intent, making standard decentralized orchestration a fundamental security risk [7].

To resolve this tension between distributed performance and client confidentiality, we propose a *Privacy-Preserving FaaS Marketplace*. Our approach transforms the serverless ecosystem into an open hub that integrates a homomorphic matchmaking substrate [8] to eliminate the vulnerabilities of decentralization. Within this substrate, *Agents* “blindly” propagate and aggregate serverless function offers alongside their execution capabilities. This aggregated state enables brokers to privately resolve client execution requests—without exposing plaintext function identifiers—and optimize their dispatching to the chosen provider¹. Consequently, we formalize a novel location- and QoS-aware mechanism for multi-objective function placement across the continuum based on client constraints and preferences.

While we reuse our previously published homomorphic matching substrate without modification [8], **the primary contributions of this paper** are (i) a decentralized FaaS Marketplace architecture, and (ii) a new multi-objective aggregation and dispatching mechanism based on QoS hyperrectangles.

To validate our approach, we simulated the Marketplace on a real-world Edge-Cloud topology built using a geographical database. Our evaluation demonstrates the system’s effectiveness in satisfying complex, multi-objective client constraints, improving on latency-minimizing FaaS gateways. Specifically, we show that our multi-objective approach prevents severe SLA violations in cloud deployments and eliminates sub-optimal placement penalties inherent to the heterogeneous

¹While many distributed-systems communities refer to this runtime decision as *routing* or *provider selection*, the FaaS literature commonly uses the term *placement*, since selecting a host and instantiating a function are often tightly coupled. Accordingly, we use *function placement* to denote the process of selecting a provider and dispatching the request to it.

continuum. Furthermore, we validate the operational feasibility of our decentralized architecture, quantifying the fundamental engineering trade-offs between state aggregation scalability and function placement precision.

The remainder of this paper is structured as follows. Section II discusses related work. Section III presents the decentralized marketplace architecture. Section IV formalizes the system model and the underlying privacy-preserving substrate, which is then realized by the dynamic orchestration and cost minimization algorithms detailed in Section V. Section VI outlines the simulation methodology and presents the experimental evaluation. Finally, Section VII concludes the paper.

II. RELATED WORK

The migration of serverless FaaS architectures from centralized clouds to the edge addresses latency bottlenecks and supports modern workloads [9]. Recent efforts focus on orchestration across the Edge-Cloud continuum [10], introducing QoS-aware offloading policies [5] and performance-driven load balancing for heterogeneous edge nodes [11]. To mitigate vendor lock-in [12], federated fabrics [13], continuum slicing techniques [14], and distributed frameworks [15] serve workloads across multi-provider clusters. However, these solutions typically rely on a trusted orchestration layer that observes plaintext execution requests, leaving a critical privacy vulnerability unresolved.

Simultaneously, the economic and performance optimization of serverless environments—such as strategies to mitigate cold-start penalties [16] or demystify execution costs [17]—has driven the exploration of decentralized resource trading. Emerging large-scale initiatives, such as the project in [18], are actively pioneering this shift by utilizing distributed orchestration agents, blockchain-based trust, and self-organizing swarms to manage microservices across the continuum. However, while these agent-based architectures provide highly robust, trustless frameworks for general applications, their reliance on heavyweight consensus or complex self-organization algorithms is fundamentally incompatible with the ephemeral, millisecond-scale scheduling specifically required by dynamic FaaS workloads.

Finally, to address user privacy, secure publish/subscribe architectures protect subscriber interest from untrusted infrastructure providers [7]. While Fully Homomorphic Encryption (FHE) [19] and Private Information Retrieval (PIR) [20] offer robust theoretical guarantees for data privacy, their extreme computational overheads remain prohibitive for millisecond-scale FaaS placement. Instead, our publish/subscribe privacy mechanism builds directly upon the more lightweight Paillier cryptosystem to enable brokers to resolve function identifiers homomorphically [8].

In conclusion, while existing literature treats performance optimization, resource trading, and execution privacy as isolated domains, our FaaS Marketplace bridges this gap by combining a privacy-preserving substrate with a comprehensive orchestration engine. This enables multi-objective function

placement across spatial, performance, and economic dimensions without compromising client privacy.

III. SYSTEM ARCHITECTURE

As introduced in our seminal work [21], the FaaS Marketplace comprises three main logical pillars: a Function Development Platform for service programming, a Function Repository for managing service metadata, and a decentralized Function Orchestration Engine. In this paper, we focus specifically on the architecture, mathematical formalization, and multi-objective placement strategies of the *Function Orchestration Engine*. This serves as the critical bridge across the continuum, dynamically and “blindly” allocating client function execution requests to optimal infrastructure providers.

As shown in Figure 1, the engine is implemented as a distributed network of *Agents* (also referred to as *Brokers*). They form a hierarchical topology mapping the physical infrastructure into distinct geographic operational regions (i.e., National Cloud $\rightarrow$ Regional Fog $\rightarrow$ City (Local) Edge).

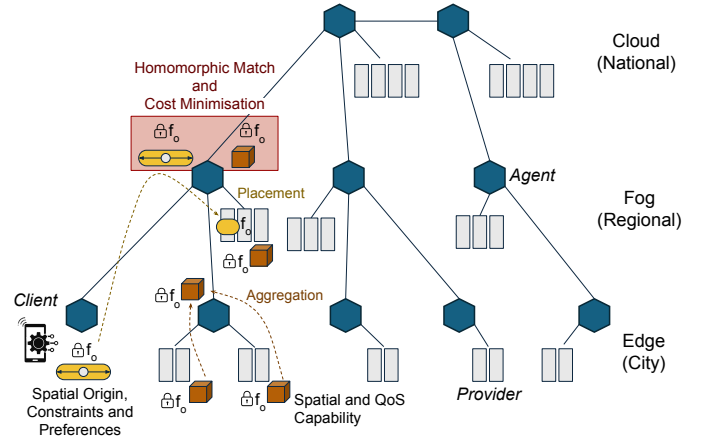


Fig. 1. The Privacy-Preserving FaaS Marketplace Architecture

The Marketplace operates within an economic ecosystem based on supply, demand, and intelligent brokerage, governed by three main actors:

- **Providers (P):** Entities operating computational infrastructure, ranging from resource-constrained edge nodes to massive cloud hyperscalers. Within the marketplace economy, they act as resource suppliers. They inject *Service Offers* into the agent network, specifying the serverless functions they support, along with a multidimensional **Spatial and QoS Capability** (e.g., geographic boundaries, latency limits, reliability guarantees, and financial cost models). Stale offers are evicted via a standard Time-To-Live (TTL) expiration mechanism.
- **Clients (C):** End-users, IoT gateways, or automated microservices seeking to execute specific function-based workloads. They act as consumers, emitting *Service Requests* into the marketplace. These requests mathematically define the client’s **Spatial Origin** (used initially to enforce strict provider geographic coverage limits, and subsequently to calculate topological latency penalties),

strict multi-objective QoS **Constraints** (absolute SLA boundaries), and a set of **Preferences** (e.g., dynamically prioritizing latency minimization over financial cost).

- **Agents (A):** Intelligent entities organized in a decentralised spatial tree that negotiate trades between clients and providers by executing the core matchmaking logic. They dynamically aggregate provider capabilities to construct an orchestration state without learning the plaintext identities of the advertised functions. To ensure the client’s specific execution request is never exposed to the infrastructure providers, Agents resolve the request via a secure **Homomorphic Match**, followed immediately by a constrained multi-criteria **Cost Minimization** to dispatch the workload toward the optimal provider.

IV. SYSTEM MODEL AND PRIVACY SUBSTRATE

To formalize the marketplace functionality across the continuum, we first establish its multi-domain governance and threat model. We then introduce the cryptographic primitives that enable homomorphic resolution, followed by the mathematical model underpinning the privacy-preserving advertisement of provider offers and the multi-objective placement of client requests.

A. Governance and Threat Model

Unlike single-domain, centralized orchestrators, our Marketplace models this environment as a multi-domain ecosystem. Because we architecturally decouple the decentralized geospatial overlay of *Agents* from the physical execution layer, the marketplace can function as a federated orchestration substrate governed by independent brokerages or neutral cloud consortia (e.g., Gaia-X). This structure inherently encourages *Providers* to monetize idle edge capacity while maintaining operational autonomy; meanwhile, *Clients* benefit from mitigated vendor lock-in and the dynamic satisfaction of their strict multi-objective QoS constraints and preferences.

While this federated structure provides economic and operational benefits, dispatching requests through a decentralized overlay managed by diverse entities means clients cannot implicitly trust the intermediate infrastructure. Consequently, the marketplace operates under an *honest-but-curious* threat model. We assume that independent Providers and Agents will faithfully execute the orchestration protocol to maintain their market standing and secure financial compensation. However, as commercially driven entities, they are incentivized to profile user behavior and may attempt to infer sensitive client workflows by analyzing function identifiers.

To mitigate this vulnerability, the marketplace strictly decouples function request placement from actual function execution. Agents match blinded client requests to infrastructure capabilities based solely on QoS metadata. The specific identity of the requested function is resolved only by the final destination Provider. Although identifiers are blinded, intermediate entities might attempt to infer the target function from plaintext QoS constraints; formalizing protections against this metadata leakage is reserved for future work.

B. Homomorphic Encryption and Blind Function Resolution

At system initialization, Clients and Providers retrieve Paillier security parameters from a Homomorphic Encryption Parameter Service (HEPS) [8]. They utilize these parameters to perform local “self-blinding” operations, generating semantically secure ciphertexts of their function names. Prior to encryption, the string-based function identifiers are deterministically encoded into large integers compatible with the Paillier plaintext space. Agents orchestrate the system entirely over these ciphertexts using a homomorphic Match primitive.

Because Paillier only permits inequality verification between blinded values, checking for an exact string match must be mathematically decomposed into a double inequality check: $f_r \geq f_o$ and $f_r < f_o + 1$. Consequently, Providers encrypt two blinded values for their offered function f_o : $b_{m_o} = E(f_o)$ and $b1_{m_o} = E(f_o + 1)$. Clients encrypt a single value for their requested function f_r : $b_{m_r} = E(f_r)$.

To facilitate secure orchestration, we encapsulate this double-inequality verification within a homomorphic Match primitive. This primitive enables the Agent to blindly compare the encrypted identifier b_{m_r} of a client’s request (or another incoming offer) against the blinded values $\{b_{m_o}, b1_{m_o}\}$ representing an indexed function string and its increment, without ever decrypting the underlying plaintext identifiers. By evaluating these conditions locally, the Agent securely verifies exact equality (a decrypted difference of 0) or determines the strict alphabetical ordering (-1 or 1) if the match fails [8].

To complete the execution, Providers maintain a private local registry. When a request arrives, the Provider resolves the encrypted identifier locally to retrieve the corresponding container URI. It is important to note that while this architecture preserves *orchestration privacy*—ensuring that intermediate Agents observe only opaque ciphertexts and QoS metadata—it does not inherently provide *host execution privacy*. During the service onboarding phase, a Provider must map the blinded identifier to a local container binary; therefore, an honest-but-curious Provider learns the function’s identity in order to instantiate the corresponding container. While preventing host-level introspection is outside the scope of this work, the proposed orchestration substrate remains fully complementary to emerging Confidential Computing paradigms, which isolate opaque workloads within hardware-based Trusted Execution Environments (TEEs).

C. Formalizing Offers, Requests, and Placement State

To represent the spatial and operational capabilities of the continuum, each **Service Offer** O_{f_o} generated by a Provider p for an offered function f_o is formalized as:

$$O_{f_o} = \langle \{b_{m_o}, b1_{m_o}\}, p_{id}, \mathcal{H}_p(f_o) \rangle \quad (1)$$

Where $\{b_{m_o}, b1_{m_o}\}$ provides the blinded values required by the Agent for homomorphic matching, p_{id} is the provider’s network routing identifier, and $\mathcal{H}_p(f_o)$ is the provider’s **Capability Hyperrectangle** within a unified D -dimensional space.

The first S dimensions of $\mathcal{H}$ represent the geographic region of coverage of the offer (e.g., bounded latitude and longitude intervals). The remaining $D - S$ dimensions represent

orthogonal QoS metrics. As offers propagate through the decentralized Agent overlay, maintaining individual records for N distinct providers causes severe state explosion, which would require a computationally expensive heuristic for frontier exploration at runtime. To avoid this high computational overhead, Agents dynamically aggregate offers to construct an *approximate Pareto front*. This compressed search space, denoted $\mathcal{M}$, is a set of disjoint *Capability Hyperrectangles* ($\mathcal{H}$) associated with a given function, where $|\mathcal{M}| \ll N$. This geometric aggregation applies the foundational principles of R-Trees [22], utilizing Minimum Bounding Rectangles to index multi-dimensional data. For every QoS dimension d (where $d \in [S+1, D]$), a hyperrectangle $\mathcal{H}_m \in \mathcal{M}$ encapsulates the capability mathematically through three distinct components:

- **QoS Bounds** ($[l^d, u^d]$): The closed interval representing the potential variation between the minimum and maximum QoS capabilities of the (merged) providers.
- **QoS Coverage** (I^d): A unidirectional containment region derived from the QoS Bounds, utilized for expected client QoS constraints strict satisfaction checks. For a minimization metric (e.g., latency), it extends from the best available capability up to the system maximum, defined as $I^d = [l^d, V_{\text{MAX}}^d]$. For a maximization metric (e.g., reliability), it extends from the absolute system minimum up to the best available capability, defined as $I^d = [0.0, u^d]$.
- **QoS Centroid** (c^d): A single coordinate representing the expected QoS yield of the merged capabilities. This center of mass is weighted by the discrete number of encapsulated *Service Offers* to prevent extreme capability outliers from skewing the aggregated state. It is used to compute the multi-objective cost of the merged offer.

To match workloads based on this aggregated state $\mathcal{M}$, a **Service Request** R_{f_r} submitted by a Client for a function f_r is formalized as:

$$R_{f_r} = \langle b_{m_r}, \vec{v}_{f_r}, \vec{w} \rangle \quad (2)$$

Where b_{m_r} is the blinded function identifier. The term $\vec{v}_{f_r} = \langle v_1, \dots, v_S, v_{S+1}, \dots, v_D \rangle$ is a strict execution constraint vector, where the first S scalar values denote the client's physical spatial location (e.g., latitude and longitude) and the remaining $D - S$ values define the absolute QoS limits. Finally, $\vec{w} = \langle w_{S+1}, \dots, w_D \rangle$ is a normalized weight vector ($\sum_{i=S+1}^D w_i = 1$) that provides clients with an intuitive, proportional mechanism to express strict capability trade-offs. This formulation is designed to drive the efficient Weighted Sum Scalarization utilized during request placement.

These mathematical definitions drive the three core operations of the orchestration engine described below.

Covering and State Aggregation. During provider offer propagation, Agents aggregate distinct Service Offers for the same blinded function through *geometric covering*, using the hyperrectangle QoS Coverage (I^d). If a newly advertised capability $\mathcal{H}_n$ falls within the existing spatial limits and its QoS Coverage is mathematically subsumed by an existing

hyperrectangle $\mathcal{H}_m \in \mathcal{M}$ across all QoS dimensions, it provides no marginal benefit and is suppressed. Formally, the capability is strictly pruned if the following condition is met:

$$I_n^d \subseteq I_m^d \quad \forall d \in [1, D] \quad (3)$$

Otherwise, the Agent attempts to merge the capability $\mathcal{H}_n$ with an overlapping candidate $\mathcal{H}_m \in \mathcal{M}$. This is achieved by calculating new absolute minimums and maximums to expand the aggregated QoS Bounds ($[l^d, u^d]$). To prevent false positives in future request placement, this merge is committed strictly if the resulting Aggregation Error (ε_{agg}) remains below an adjustable tolerance threshold (τ_{merge}). We define this error as the maximum penalty evaluated independently across three constraint checks: (i) *geographic spatial dead space*, computed as the ratio of empty physical area introduced by the merged bounding box relative to the original capabilities; (ii) *dimensional capability gaps*, calculated as the normalized expansion of the strict min/max QoS envelopes (e.g., latency and cost limits) required to encapsulate both providers; and (iii) *centroid shift*, operationally defined as the variance between the merged envelope's extreme bounds and its new density-weighted QoS Centroid. If $\varepsilon_{\text{agg}} \leq \tau_{\text{merge}}$, the newly merged capability replaces its original components in the spatial store.

Feasibility Checking (Pruning). At function placement time, the Agent calculates an optimistic network latency penalty (δ_{latency}) by applying a linear spatial-to-time conversion scalar to the geographic Euclidean distance (adjusted for longitudinal wrapping) between the client and the candidate's spatial bounding edge. This penalty is added to the latency dimension of the candidate's baseline QoS Coverage, yielding I_e^d . To prevent downstream SLA violations, the topological branch survives pruning only if the request constraints fall within these intervals. Formally, a candidate is valid if it satisfies the following feasibility condition:

$$v_d \in I_e^d \quad \forall d \in [S+1, D] \quad (4)$$

Multi-Objective Cost Minimization. For topological branches that survive the pruning phase, the Agent evaluates candidate (aggregated) offers $\mathcal{H}_m \in \mathcal{M}$ using a Multi-Criteria Decision Making (MCDM) approach [23]. The Agent applies Weighted Sum Scalarization directly to each candidate's QoS Centroid (c^d), recalculating the spatial latency penalty specifically against c^d to accurately reflect the expected execution cost. To achieve this, it first computes a normalized cost $S_d \in [0, 1]$ by standardizing the centroid against the client's absolute constraint vector $\vec{v}_{f_r}$, ensuring orthogonal metrics (e.g., latency and reliability) are mathematically comparable. These dimensional costs are then scaled by the client's preference weights $\vec{w}$ to yield the final Multi-Objective Cost for a candidate $\mathcal{H}_c$:

$$C(\mathcal{H}_c, \vec{v}_{f_r}, \vec{w}) = \sum_{d=S+1}^D (S_d \times w_d) \quad (5)$$

By minimizing this cost function at invocation time, the Agent ranks the performance of aggregated offers directly, rather than relying on the optimistic bounds defined by the QoS

Coverage region. In addition, this scalarization guarantees a deterministic $O(|\mathcal{M}| \times (D - S))$ execution time per branch. This incurs far lower overhead than computing or exploring a Pareto-optimal frontier for each request, with a computation time that would be incompatible with the ephemeral lifecycle of typical FaaS executions.

V. SYSTEM FUNCTIONALITY AND OPERATION

The theoretical system model and privacy substrate defined in Section IV are realized within the distributed *Function Orchestration Engine* through two continuous, interconnected execution phases. Phase 1 (Algorithm 1) applies the *Covering and State Aggregation* mechanisms to dynamically build the agent function orchestration state. Phase 2 (Algorithm 2) executes the *Feasibility Checking* and multi-criteria *Cost Minimization* to dispatch client requests towards suitable candidates. To achieve high-throughput orchestration while preserving client privacy, Agents decouple function identifier resolution from multidimensional capability evaluation. Each Agent maintains a two-tier local placement state:

- 1) *BlindedFunctionDirectory*: An encrypted, self-balancing binary search tree acting as the top-level namespace. Its keys are the homomorphically blinded function identifiers, and its values resolve to the specific capability index associated with that function.
- 2) *OfferIndex*: A function-specific spatial data structure (e.g., a R-Tree, QuadTree) that organizes the (aggregated) provider offers. While elements are indexed by the physical spatial dimensions (1 to S) for fast geographic querying, their values contain the complete D -dimensional *Capability Hyperrectangles* ($\mathcal{M}$) that bind offer geolocations to QoS metrics.

The traversal of an agent's function directory is performed via the method *BlindedFunctionDirectory.HSearch(BlindedKey)*. It encapsulates the *Match* primitive defined in Section IV to retrieve the *OfferIndex* associated with the specified key in $\mathcal{O}(\log_2(N_f))$ operations (with N_f being the size of the index). It accepts blinded identifiers from either an incoming Service Offer O_{f_o} or a Service Request R_{f_r} . This blinded information retrieval serves as the fundamental prerequisite for both orchestration phases. Once an *OfferIndex* is retrieved, the Agent processes its content based on the interaction context.

A. Phase 1: Privacy-Preserving Offer Processing

Algorithm 1 dictates how newly advertised multidimensional capabilities are dynamically aggregated and how the resulting updated boundaries are propagated to adjacent Agents of the *Function Orchestration Engine*. To prevent state explosion, each Agent applies two distinct geometric optimizations based on the formalisms established in Eq. 1.

First, an *Inbound Geometric Covering Filter* evaluates the newly advertised capability $\mathcal{H}_n$ against the spatial store. If it is fully subsumed by an existing $\mathcal{H}_m \in \mathcal{M}$, it provides no marginal benefit and is instantly pruned (Eq. 3).

Second, the Agent executes *Cascading State Aggregation*. To process the capability $\mathcal{H}_n$, the Agent utilizes the spatial

Algorithm 1 Privacy-Preserving Offer Processing

Require: Service Offer $O_{f_o} = \langle \{b_{m_o}, b1_{m_o}\}, p_{id}, \mathcal{H}_n \rangle$ received at Agent A .

- 1: **{Stage 1: Homomorphic Traversal & Inbound Filter}**
- 2: $\mathcal{M} \leftarrow A.BlindedFunctionDirectory.HSearch(\{b_{m_o}, b1_{m_o}\})$
- 3: **if** $\exists \mathcal{H}_m \in \mathcal{M}$ satisfies covering condition (Eq. 3) **then**
- 4: **return** {Redundant capability, prune instantly}
- 5: **{Stage 2: Cascading State Aggregation}**
- 6: **repeat**
- 7: merged $\leftarrow$ **false**
- 8: **for all** candidates $\mathcal{H}_m \in \mathcal{M}$ geographically intersecting $\mathcal{H}_n$ (queried via index) **do**
- 9: **if** Aggregation Error $\varepsilon_{agg} \leq \tau_{merge}$ **then**
- 10: Expand $\mathcal{H}_n$ QoS bounds $[l^d, u^d] \forall d$ to bound $\mathcal{H}_m$
- 11: Recompute c^d and $I^d \forall d \in [S + 1, D]$
- 12: Remove absorbed $\mathcal{H}_m$ from $\mathcal{M}$
- 13: merged $\leftarrow$ **true**
- 14: **break** {Restart spatial query with expanded $\mathcal{H}_n$ }
- 15: **until not** merged
- 16: Insert finalized $\mathcal{H}_n$ into $\mathcal{M}$; let O_p be the resulting offer
- 17: **{Stage 3: Outbound Covering & Propagation}**
- 18: **for all** peers P intersecting O_p 's spatial coverage **do**
- 19: **if not** $A.IsRedundantOutbound(O_p, P)$ {Eq. 3} **then**
- 20: **Forward** O_p to peer P

index of the *OfferIndex* to efficiently query $\mathcal{M}$ for geographically intersecting candidates. A candidate is merged with $\mathcal{H}_n$ strictly if the operation satisfies the Aggregation Error threshold ($\varepsilon_{agg} \leq \tau_{merge}$) defined in Section IV. Because a successful merge expands the aggregate's physical spatial boundaries, the resulting region might bridge the geographic gap with another previously disjoint hyperrectangle in $\mathcal{M}$. Consequently, this process operates as a cascading loop, iteratively querying the spatial tree with the newly expanded boundaries until the aggregation reaches equilibrium.

Finally, the resulting capability is readied for topological propagation. This outbound phase is strictly constrained by an *Outbound Geometric Covering Filter* ($A.IsRedundantOutbound$). Before transmitting, the Agent re-applies the covering condition (Eq. 3) to verify if the capability is already subsumed by the aggregated state previously forwarded to that specific topological peer. By pruning redundant outbound updates, this final check effectively suppresses network flooding.

B. Phase 2: Privacy-Preserving Request Dispatching

Algorithm 2 governs the multi-objective placement of client workloads (Eq. 2). Following the resolution of the blinded function identifier via *HSearch*, the Agent retrieves the corresponding offer state set $\mathcal{M}$, which unifies both locally attached providers and aggregated topological branches into a single candidate pool. Request dispatching then occurs in two sequential stages. First, the Agent executes *Feasibility Checking*. For each candidate $\mathcal{H}_c \in \mathcal{M}$, it calculates and injects the dynamic network latency penalty ($\delta_{latency}$) to establish the effective QoS Coverage (I_e^d), exactly as formalized in

Section IV. Any candidate failing to satisfy the strict feasibility condition (Eq. 4) is instantly pruned.

Finally, all surviving capabilities form a valid candidate set $\mathcal{V}$, which then undergoes *Cost Minimization*. If $\mathcal{V} \neq \emptyset$, the Agent calculates the Multi-Objective Cost $C(\mathcal{H}_c, \vec{v}_{f_r}, \vec{w})$ for each candidate using its QoS Centroid (c^d), exactly as formulated in Eq. 5. The request is strictly forwarded to the topological next-hop (a local provider or peer Agent) that minimizes this scalarized cost. Conversely, if $\mathcal{V} = \emptyset$, the constraints are physically unsatisfiable within the accessible continuum, and the request is dropped to prevent downstream SLA violations.

Algorithm 2 Privacy-Preserving Request Resolution

Require: Client Request $R_{f_r} = \langle b_{m_r}, \vec{v}_{f_r}, \vec{w} \rangle$ received at Agent A .

- 1: **{Stage 1: Homomorphic Tree Traversal}**
 - 2: $\mathcal{M} \leftarrow A.\text{BlindedFunctionDirectory}.\text{HSearch}(b_{m_r})$
 - 3: **if** $\mathcal{M} = \emptyset$ **then**
 - 4: **Drop** R_{f_r} {Function unavailable in continuum}
 - 5: **return**
 - 6: **{Stage 2: Feasibility Checking}**
 - 7: $\mathcal{V} \leftarrow \emptyset$ {Initialize valid candidate set}
 - 8: **for all** candidates $\mathcal{H}_c \in \mathcal{M}$ **do**
 - 9: $\delta_{\text{latency}} \leftarrow \text{dist}(v_{1..S}, \mathcal{H}_c^{1..S})$ {Spatial distance}
 - 10: Inject δ_{latency} into QoS Coverage of $\mathcal{H}_c$ to yield I_e^d
 - 11: **if** $v_d \in I_e^d \ \forall d \in [S+1, D]$ {Eq. 4} **then**
 - 12: $\mathcal{V} \leftarrow \mathcal{V} \cup \{\mathcal{H}_c\}$
 - 13: **{Stage 3: Cost Minimization (Scalarization)}**
 - 14: **if** $\mathcal{V} \neq \emptyset$ **then**
 - 15: Select $\mathcal{H}^* \in \mathcal{V}$ minimizing $C(\mathcal{H}^*, \vec{v}_{f_r}, \vec{w})$ {Eq. 5}
 - 16: **Dispatch** R_{f_r} to next-hop of $\mathcal{H}^*$
 - 17: **else**
 - 18: **Drop** R_{f_r} {Unsatisfiable QoS constraints}
-

VI. EXPERIMENTAL EVALUATION

To validate the feasibility and architectural benefits of the FaaS Marketplace, we developed a bespoke, open-source Java simulation framework². The simulator explicitly models the decentralized orchestration process, the aggregation of *Spatial and QoS Capabilities* into *Capability Hyperrectangles* ($\mathcal{H}$), and the multi-objective cost minimization logic formalized in Sections IV and V. To represent the physical constraints of the Edge-Cloud continuum, the spatial topology is anchored to a real-world geographic representation of the US using the GeoNames database³, constructing a multi-layer hierarchical tree (National $\rightarrow$ Regional $\rightarrow$ City). Within this substrate, we model two distinct infrastructures: a centralized *Cloud Topology* featuring 25 hyperscalers, and a fully dispersed *Edge-Cloud Continuum* comprising 2,775 infrastructure providers distributed proportionally to actual populated places across Cloud, Fog, and Edge tiers.

The simulated environment processes a steady-state workload of 1,000,000 client execution requests. Provider *Spatial and QoS Capabilities* are defined across four orthogonal dimensions (compute latency, execution cost, reliability, and bandwidth). Baseline capabilities are anchored to the provider’s physical tier and adjusted at runtime to model diverse availability states (e.g., warm-optimized vs. cold-start penalties). Conversely, client *Service Requests* express a wide range of QoS requirements capturing heterogeneous latency, bandwidth, reliability, and cost requirements. Requests include SLA constraints ($\vec{v}$) and customized preference weights ($\vec{w}$) for the multi-objective cost function (Eq. 5). Client demands are subjected to runtime probabilistic skewing to ensure that requests are not neatly partitioned into Cloud-, Fog-, or Edge-specific classes; instead, they span a spectrum of demand profiles, reflecting the diversity and variability typical of real FaaS workloads. To isolate and evaluate the multi-objective matching dynamics without the confounding variable of homomorphic lookup sparsity, we assume a uniform service topology where providers offer all requested functions.

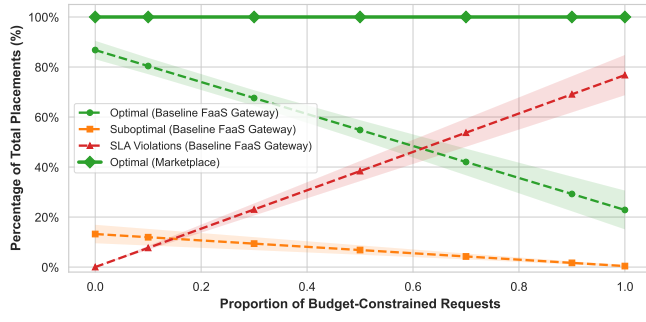
Because the Marketplace engine integrates the homomorphic matching primitives and binary tree traversal mechanisms benchmarked in our previous work [8], [21], we omit an isolated cryptographic performance analysis here and focus on the spatial and QoS placement dynamics. Given the low average number of topological hops required for successfully delivered requests, the cumulative homomorphic overhead introduced at each Agent remains minimal during function placement. Furthermore, our ongoing research into hardware acceleration indicates that executing these matching primitives on specialized GPUs will allow an Agent to perform 9.6 million matches per second on a BlindedFunctionDirectory storing 10 billion function names, with each individual match requiring only 1.7 ms.

A. Experiment Key Performance Metrics

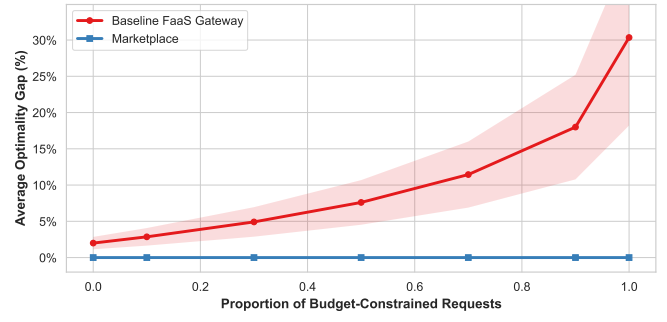
We define several key metrics for our evaluation. *Proactive Drops* represent requests rejected at the ingress because no reachable provider satisfies the multi-objective constraints, saving network bandwidth. Conversely, *False Positive Dispatches* occur when requests are forwarded based on a highly merged OfferIndex, only to be dropped later when more granular matching fails. For the baseline, we also track *SLA Violations*, representing requests that are successfully placed but ultimately violate the user’s hard QoS constraints (e.g., exceeding the budget limit). For successfully dispatched requests, we distinguish between *Optimal Placements* (assigned to the mathematically best provider) and *Suboptimal Placements* (assigned to a valid but non-optimal provider due to offer abstraction). To quantify cost degradation due to state abstraction, we define the *Average Optimality Gap* as the relative percentage deviation from the theoretical global optimum. Finally, computational and memory overheads are measured via *Total Constraint Evaluations* and *Average OfferIndex Size*. Using this framework, we investigate the Marketplace architecture through three scenarios focusing on dispatching

²<https://github.com/francesco-tusa/dnsplus/tree/faas-marketplace>

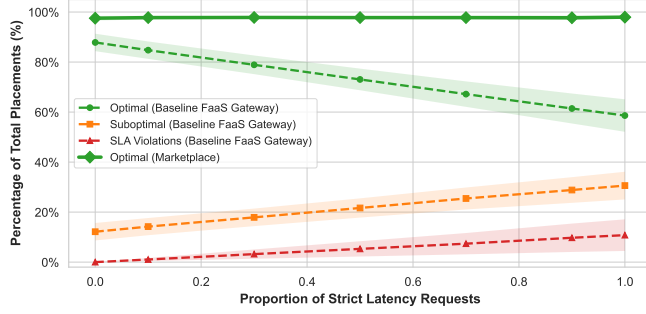
³<https://www.geonames.org>



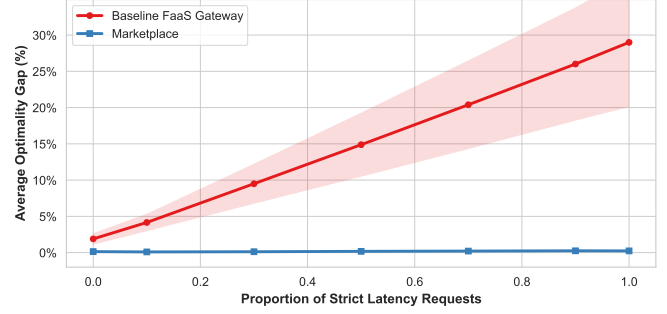
(a) Placement Composition (Cloud)



(b) Average Optimality Gap (Cloud)

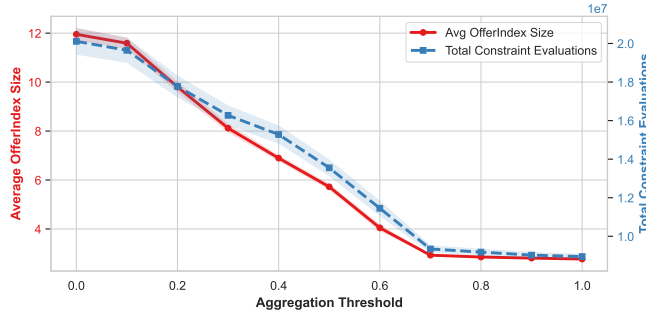


(c) Placement Composition (Edge-Cloud)

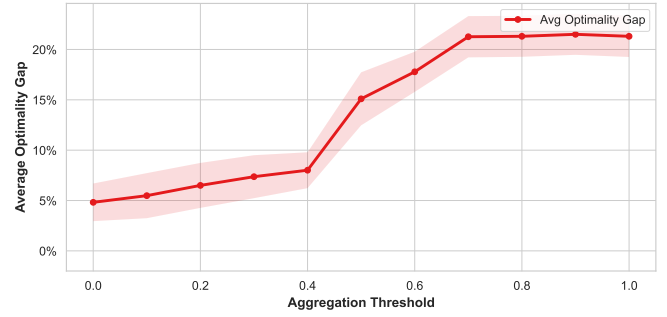


(d) Average Optimality Gap (Edge-Cloud)

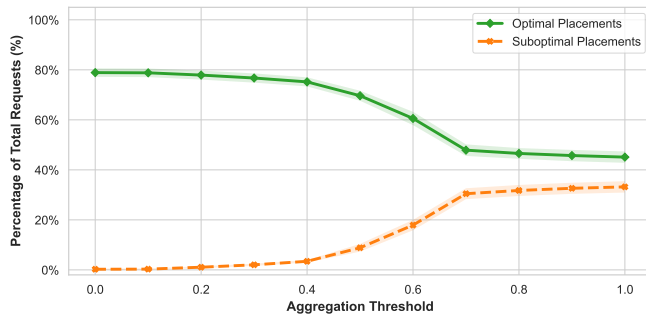
Fig. 2. Latency-Minimizing vs. Multi-Objective Placement in the Cloud (Top) and Edge-Cloud (Bottom).



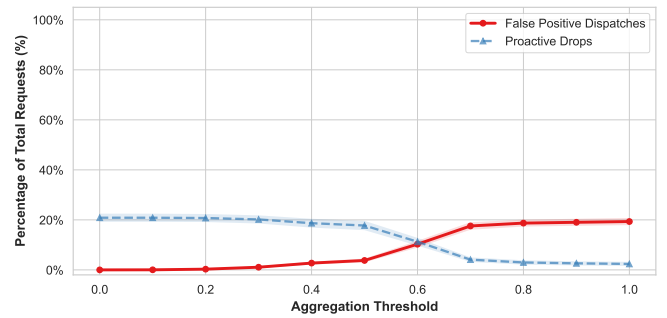
(a) Storage and Request Processing Overhead



(b) Average Optimality Gap



(c) Placement Optimality



(d) Unsatisfiable Requests

Fig. 3. Marketplace Trade-offs: State Compression vs. Placement Accuracy.

effectiveness, efficiency, and scalability trade-offs. All reported results represent the mean of 30 independent runs, with shaded areas indicating 95% confidence intervals.

B. Budget Constraints in the Cloud

First, we compare how a baseline latency-minimizing FaaS gateway processes user requests versus our Marketplace’s multi-objective strategy. To ensure realistic settings that balance state compression with computational overhead, the marketplace aggregation threshold is fixed at $\tau_{\text{merge}} = 0.25$ for these comparative evaluations. Today’s multi-region serverless infrastructures typically use latency to place requests at the nearest geographic datacenter (analogous to latency-based Geo-DNS routing) to minimize network delay. However, this approach ignores orthogonal constraints, such as economic optimization. Even across a Cloud topology, execution costs are heterogeneous; a geographically proximate datacenter may charge a premium due to regional pricing or warm functions, penalizing cost-sensitive clients. By increasing the proportion of strictly budget-constrained requests (i.e., those with rigid execution cost limits) within this multi-region cloud scenario, we observe the performance degradation incurred by proximity-only function placement.

As shown in Figure 2(a), the baseline inadvertently subjects a substantial percentage of total requests to SLA violations because it merely dispatches workloads to the closest nodes regardless of their potential financial premiums. Conversely, the Marketplace proactively evaluates the multi-objective constraint vector ($\vec{v}$), identifying clients’ unfeasible economic requirements and dropping unresolvable requests at the ingress (yielding high *Proactive Drops*). Furthermore, Figure 2(b) demonstrates that for successfully placed requests, the Marketplace effectively minimizes the multi-objective cost via scalarization, maintaining a 0% *Average Optimality Gap*; the baseline suffers an exponentially increasing percentage deviation from the theoretical global optimum.

C. Latency Constraints in the Continuum

While the previous experiment demonstrated that the baseline FaaS gateway causes severe SLA violations due to violated cost constraints, shifting to the Edge-Cloud continuum exposes a different inefficiency. We evaluate this by increasing the proportion of low-latency (≤ 30 ms) requests with relaxed budgets across a heterogeneous infrastructure.

As shown in Figure 2(c), the baseline avoids the SLA violations seen in the Cloud scenario because the nearest Edge nodes happen to fall within the relaxed budget limits. However, while the hard budget constraints are not strictly violated, cost optimization is entirely ignored. By prioritizing absolute minimum latency, the baseline bypasses highly cost-effective nearby nodes that still safely satisfy the 30 ms SLA, forcing requests onto unnecessarily expensive hardware. Consequently, *Optimal Placements* plummet and are entirely displaced by *Suboptimal Placements*.

Figure 2(d) quantifies this single-objective limitation. Unlike the homogeneous Cloud—where the *Average Optimality Gap*

increases exponentially with constraint pressure—the continuum’s heterogeneity incurs a steady, linear penalty. Because the baseline consistently traps requests in premium, local edge nodes rather than utilizing cost-effective upstream fog providers that still satisfy the SLA, its optimality gap degrades proportionally with the workload, reaching a maximum deviation of nearly 30%, as seen already in the previous experiment. Conversely, the Marketplace multi-objective placement strategy effectively bypasses nearby inefficient nodes. By leveraging the full spectrum of available resources, it maintains a near-zero optimality gap and nearly 100% optimal placement rate for all delivered requests.

D. State Aggregation Trade-offs

Finally, we evaluate the trade-offs of decentralized agent state aggregation. With the Marketplace operating within this distributed infrastructure, we adjust the aggregation threshold (τ_{merge}) from 0.0 (no aggregation) to 1.0 (maximum compression) to control how individual provider capabilities merge into generalized *Capability Hyperrectangles*.

Figure 3(a) highlights the computational benefits: higher thresholds reduce the *Average OfferIndex Size* and *Total Constraint Evaluations*. However, this efficiency comes at a cost. Because agents evaluate the QoS centroids (c^d) of aggregated hyperrectangles with expanded QoS Bounds rather than precise provider metrics, the placement granularity decreases. Figures 3(b) and 3(c) illustrate this penalty. Higher compression degrades the system’s ability to identify the absolute best provider, thereby increasing the *Average Optimality Gap* and *Suboptimal Placements*.

Moreover, Figure 3(d) illustrates the system’s response to unfeasible requests—those possessing QoS constraints that no available provider can satisfy. Higher aggregation thresholds induce a critical increase in *False Positive Dispatches*. Because QoS Bounds are over-compressed at the network’s edge, agents erroneously dispatch unresolvable requests deeper into the topology. These requests eventually fail when they reach downstream agents that enforce stricter matching on smaller QoS Bounds. Consequently, these requests are dropped mid-flight—wasting network bandwidth and computation along the path—rather than being efficiently shielded at the ingress via *Proactive Drops*.

Discussion. In summary, our evaluation demonstrates the necessity and efficacy of the FaaS Marketplace. First, we establish that a latency-minimizing FaaS gateway is fundamentally ill-equipped to handle heterogeneous economic constraints in the Cloud, leading budget-constrained clients to routinely suffer financial *SLA violations*. The Marketplace resolves this by enforcing strict multi-objective feasibility, preemptively converting unresolvable requests into *Proactive Drops*. Second, we demonstrate that applying latency-minimizing placement to the highly heterogeneous Edge-Cloud continuum triggers severe optimization failures. The considered baseline FaaS gateway ignores cost-effective provider alternatives, trapping requests in premium Edge nodes and causing a massive surge in *Suboptimal Placements*. The Marketplace overcomes

these single-objective limitations by evaluating all dimensions simultaneously, bypassing proximate but inefficient providers and maintaining near-zero optimality gaps. Finally, we quantify the trade-offs required to scale this decentralized architecture. While state aggregation exponentially reduces computational overhead, over-compressing capability boundaries inherently degrades placement precision. This introduces a critical engineering balance: excessive compression increases *Suboptimal Placements* and wastes network bandwidth via *False Positive Dispatches*.

VII. CONCLUSIONS AND FUTURE WORK

This paper presented a Privacy-Preserving FaaS Marketplace built upon a decentralized multi-agent architecture. By mapping QoS constraints to a multidimensional metric space and applying a multi-objective cost minimization strategy, the system enables intelligent function allocation across the Edge-Cloud continuum without relying on centralized orchestrators.

Simulations validate the architecture on two deployment scenarios, demonstrating that the Marketplace successfully overcomes the limitations of latency-minimizing placement. It prevents financial SLA violations in the Cloud and maintains optimal placement across the heterogeneous continuum. The system achieves this by matching multi-dimensional workload constraints to the appropriate infrastructure tier. Furthermore, our evaluation quantifies the critical storage and processing trade-offs required to scale decentralized orchestration.

While our current model operates within an honest-but-curious boundary, future extensions will explore lightweight metadata obfuscation to protect the plaintext QoS constraints required for spatial routing. Furthermore, because our empirical baselines assume uniform function availability, future iterations will model sparse service distributions (e.g., via Azure traces). This will allow us to evaluate how topological sparsity impacts the aggregation threshold, specifically quantifying memory state growth from rejected merges versus False Positive Dispatches from merged hyperrectangles. Finally, we will integrate GPU-accelerated matching primitives to further optimize the orchestration pipeline.

ACKNOWLEDGEMENTS

This work was partially supported by the Swarmchestrator EU research project (No. 101135012) and Huawei Technologies Co., Ltd.

REFERENCES

- [1] P. Castro, V. Ishakian, V. Muthusamy, and A. Slominski, "The Rise of Serverless Computing," *Commun. ACM*, vol. 62, no. 12, p. 44–54, 2019.
- [2] L. Bittencourt, R. Immich, R. Sakellariou, N. Fonseca, E. Madeira, M. Curado, L. Villas, L. DaSilva, C. Lee, and O. Rana, "The Internet of Things, Fog and Cloud continuum: Integration and challenges," *Internet of Things*, vol. 3–4, pp. 134–155, 2018.
- [3] M. S. Aslanpour, A. N. Toosi, C. Cicconetti, B. Javadi, P. Sbarski, D. Taibi, M. Assuncao, S. S. Gill, R. Gaire, and S. Dustdar, "Serverless Edge Computing: Vision and Challenges," in *2021 Australasian Computer Science Week Multiconference*, ser. ACSW '21. Association for Computing Machinery, 2021.
- [4] F. Tusa, S. Clayman, A. Buzachis, and M. Fazio, "Microservices and serverless functions—lifecycle, performance, and resource utilisation of edge based real-time IoT analytics," *Future Generation Computer Systems*, vol. 155, pp. 204–218, 2024.
- [5] G. Russo Russo, D. Ferrarelli, D. Pasquali, V. Cardellini, and F. Lo Presti, "QoS-aware offloading policies for serverless functions in the Cloud-to-Edge continuum," *Future Generation Computer Systems*, vol. 156, pp. 1–15, 2024.
- [6] H. Shafiei, A. Khonsari, and P. Mousavi, "Serverless Computing: A Survey of Opportunities, Challenges, and Applications," *ACM Comput. Surv.*, vol. 54, no. 11s, pp. 1–32, 2022.
- [7] S. Cui, S. Belguith, P. D. Alwis, M. Asghar, and G. Russello, "Collusion Defender: Preserving Subscribers' Privacy in Publish and Subscribe Systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 03, pp. 1051–1064, 2021.
- [8] F. Tusa, D. Griffin, and M. Rio, "DNS++: Dynamic Name Resolution with Homomorphic Encryption Based Privacy," in *2023 32nd International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2023, pp. 1–7.
- [9] Y. Li, Y. Lin, Y. Wang, K. Ye, and C. Xu, "Serverless Computing: State-of-the-Art, Challenges and Opportunities," *IEEE Transactions on Services Computing*, vol. 16, no. 2, pp. 1522–1539, 2023.
- [10] A. Ullah, T. Kiss, J. Kovács, F. Tusa, J. Deslauriers, H. Dagdeviren, R. Arjun, and H. Hamzeh, "Orchestration in the Cloud-to-Things compute continuum: taxonomy, survey and future directions," *Journal of Cloud Computing*, vol. 12, no. 1, p. 135, 2023.
- [11] M. S. Aslanpour, A. N. Toosi, M. A. Cheema, M. B. Chhetri, and M. A. Salehi, "Load balancing for heterogeneous serverless edge computing: A performance-driven and empirical approach," *Future Generation Computer Systems*, vol. 154, pp. 266–280, 2024.
- [12] D. Mo, R. Cordingley, D. Chinn, and W. Lloyd, "Addressing Serverless Computing Vendor Lock-In through Cloud Service Abstraction," in *2023 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2023, pp. 193–199.
- [13] R. Chard, Y. Babuji, Z. Li, T. Skluzacek, A. Woodard, B. Blaiszik, I. Foster, and K. Chard, "funcX: A Federated Function Serving Fabric for Science," in *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*. Association for Computing Machinery, 2020, p. 65–76.
- [14] F. Tusa and S. Clayman, "End-to-end slices to orchestrate resources and services in the cloud-to-edge continuum," *Future Generation Computer Systems*, vol. 141, pp. 473–488, 2023.
- [15] S. Ristov, S. Brandacher, M. Hautz, M. Felderer, and R. Breu, "CODE: Code once, deploy everywhere serverless functions in federated FaaS," *Future Generation Computer Systems*, vol. 160, pp. 442–456, 2024.
- [16] P. Silva, D. Fireman, and T. E. Pereira, "Prebaking functions to warm the serverless cold start," in *Proceedings of the 21st International Middleware Conference*, 2020, pp. 1–13.
- [17] F. Liu and Y. Niu, "Demystifying the Cost of Serverless Computing: Towards a Win-Win Deal," *IEEE Transactions on Parallel and Distributed Systems*, pp. 1–15, 2023.
- [18] T. Kiss, A. Ullah, G. Terstyanszky, O. Kao, S. Becker, Y. Verginadis, A. Michalas, V. Stankovski, A. Kertesz, E. Ricci *et al.*, "Swarmchestrator: Towards a fully decentralised framework for orchestrating applications in the cloud-to-edge continuum," in *International Conference on Advanced Information Networking and Applications*. Springer, 2024.
- [19] C. Marcolla, V. Sucasas, M. Manzano, R. Bassoli, F. H. Fitzek, and N. Aaraj, "Survey on fully homomorphic encryption, theory, and applications," *Proceedings of the IEEE*, vol. 110, no. 10, pp. 1572–1609, 2022.
- [20] G. Alfarano, K. Khathuria, and V. Weger, "A survey on single server private information retrieval in a coding theory perspective," *Applicable Algebra in Engineering, Communication and Computing*, vol. 34, 2021.
- [21] F. Tusa, "Privacy-Preserving FaaS: A Marketplace for the Serverless Edge-Cloud Continuum," in *2025 34th International Conference on Computer Communications and Networks (ICCCN)*, 2025, pp. 1–2.
- [22] A. Guttman, "R-trees: A Dynamic Index Structure for Spatial Searching," *ACM SIGMOD Record*, vol. 14, no. 2, pp. 47–57, 1984.
- [23] C.-L. Hwang and K. Yoon, *Multiple Attribute Decision Making: Methods and Applications*. Springer-Verlag, 1981.