FRESCO: Fast and Reliable Edge Offloading with Reputation-based Hybrid Smart Contracts

Josip Zilic, *TU Wien*, Vincenzo De Maio, *TU Wien*, Shashikant Ilager, *University of Amsterdam*, Ivona Brandic, *TU Wien*

***Abstract*—Mobile devices offload latency-sensitive application tasks to edge servers to satisfy applications' Quality of Service (QoS) deadlines. Consequently, ensuring reliable offloading without QoS violations is challenging in distributed and unreliable edge environments with diverse resource and reliability levels. We propose FRESCO, a fast and reliable edge offloading framework that utilizes a blockchain-based reputation system, which enhances the reliability of offloading in the distributed edge. The distributed reputation system tracks the historical performance of edge servers, while blockchain through a consensus mechanism ensures that sensitive reputation information is secured against tampering. However, blockchain consensus typically has high latency, and therefore we employ a Hybrid Smart Contract (HSC) as a *reputation state manager* that automatically computes and stores reputation securely on-chain (i.e., on the blockchain) while allowing fast offloading decisions off-chain (i.e., outside of blockchain). The *offloading decision engine* uses a reputation score from HSC to derive fast offloading decisions, which are based on Satisfiability Modulo Theory (SMT). The SMT can formally guarantee a feasible solution that is valuable for latency-sensitive applications that require high reliability. With a combination of an on-chain HSC reputation state manager and an off-chain SMT decision engine, FRESCO offloads tasks to reliable servers without being hindered by blockchain consensus. In our experiment, FRESCO reduces response time by up to 7.86 times and saves energy by up to 5.4% compared to all baselines while minimizing QoS violations to 0.4% and achieving an average decision time of just 5.05 milliseconds.**

***Index Terms*—edge offloading, reputation, hybrid smart contract, satisfiability modulo theory.**

I. Introduction

LATENCY-SENSITIVE mobile applications are subject to strict Quality of Service (QoS) requirements to enhance the user experience [1]–[3]. Such applications are resource-intensive, and executing them on resource-limited and battery-powered mobile devices can cause QoS violations. A typical solution to improve performance is to offload applications' tasks to edge servers [4], [5]. However, reliability is an issue for edge servers, due to (1) *limited resources* that cannot compensate for unstable connections and lack of support systems (e.g., cooling and backup power [6], [7]), and (2) *volatile workloads* which yields inconsistent performance for shared multi-tenant edge environments [8]. Consequently, offloading to unreliable edge servers can cause failures [6], [9] and thus postpone or prolong offloading and potentially violate QoS.

Estimating reliability at the edge is challenging due to *geo-distribution*, with diverse resources and reliability levels [10], [11], and *devices mobility* where changing edge environments and interacting with previously unknown servers is a norm [6], [12].

Many offloading works employ blockchain-enabled [13]–[15] or blockchain-based reputation systems [16]–[19] to achieve reliable offloading. As a trustworthiness metric, reputation scores can be associated with edge servers based on past performance and stored in a distributed database for informative offloading decision-making. Integrating a reputation with blockchain is sensible in malicious environments where different actors can tamper with a reputation to unjustifiably inflate selected servers while downgrading others to gain incentives unfairly, potentially leading to end-user QoS violations [16]–[19]. However, there are a few challenges of edge offloading with blockchain-based reputation systems remain unresolved, such as challenge **(C1)** without providing *formal guarantee* about the feasibility of offloading decisions which is important for applications that require high reliability, like latency-sensitive ones, **(C2)** encountering edge servers in *distributed edge* environments that have diverse reliability levels and with whom did not have any prior experience, **(C3)** not addressing reliability in terms of *edge failures*, and **(C4)** neglecting the impact of long-latency blockchain consensus on offloading decisions, especially for *latency-sensitive* applications.

To address these challenges, we introduce a FRESCO edge offloading framework, which optimizes both offloading and reliability in distributed unreliable edge scenarios for latency-sensitive applications. FRESCO consists of offloading and reliability components. Regarding offloading, we employ an *offloading decision engine* based on a formal method called satisfiability modulo theory (SMT) which is deployed on the mobile device. The SMT addresses challenge **(C1)** by providing formal proof assurance that relevant resource limitations and timing constraints are satisfied, which fits resource-constrained and latency-sensitive settings. SMT relies on constraints and logic rather than environmental variables like heuristics or machine learning, which makes it an environment-agnostic approach, suitable for distributed scenarios (C2).

Concerning reliability, to address challenge **(C3)**, thanks to the performance-tracking feature, the blockchain-based reputation system is re-purposed for estimating the reliability levels in terms of failures rather than trustworthiness. However, blockchain-based systems are slowly responsive due to consensus protocols that conflict with our latency objective. To enable a blockchain-based reputation system for latency-sensitive applications **(C4)**, we employ a hybrid smart contract (HSC) as a *reputation state manager*. HSC allows off-chain (i.e., outside of blockchain) transactions like fast offloading decisions that require performance while retaining secured on-chain storage (i.e., on the blockchain) of sensitive reputation information against malicious tampering. The HSC, deployed on the blockchain, is queried by mobile devices to retrieve secured reputation scores about servers' reliability levels to the SMT-based decision engine for reliable offloading decisions. In summary, FRESCO bypasses slow consensus with HSC for fast off-chain offloading decisions on reliable edge servers while preserving a secured on-chain reputation.

We evaluated FRESCO against Skype availability traces, simulated latency-sensitive applications, dynamic queuing workload model, and scalable infrastructure from the OpenCellID dataset. FRESCO reduces response time by up to 7.86 times and saves energy by up to 5.4% compared to all baselines while minimizing QoS violations to 0.4% and achieving a low-latency average decision time of 5.05 milliseconds.

The rest of the paper is structured as follows. Section II presents methodologies, while Section III presents the system model. Section IV formalizes the problem and presents our algorithm. In Section V, we present our experiment and evaluation results. In Section VI, we discuss the assumptions and limitations of our approach. Finally, the related work and conclusion are in Sections VII and VIII.

II. Motivation and Background

A. Motivational use case

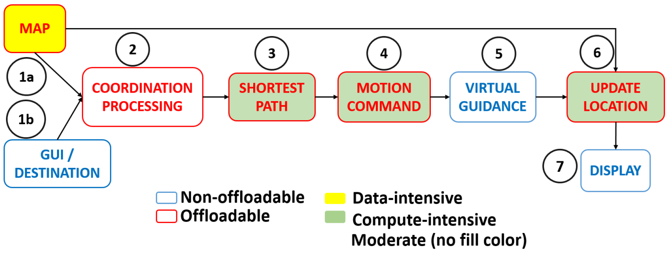


Fig. 1: A motivational use case: NaviAR mobile augmented reality application and its tasks

Latency-sensitive mobile applications are usually resource-intensive and require low-latency execution. An example is a Mobile Augmented Reality (MAR), where any significant delay hinders the users' experience [2], [3]. One of the typical MAR applications is personal live navigation called NaviAR. NaviAR supports users with real-time navigation by displaying virtual path information over the physical environment.

Figure 1 shows a typical NaviAR execution flow [3]. The application is represented as a Directed Acyclic Graph (DAG) to model execution order and task interdependencies. The NaviAR consists of heterogeneous tasks where some are offloadable while others are not due to dependency on local device functions (e.g. camera). First, the destination location is taken as input from the user, upon which the map is loaded (steps 1a and 1b). Afterward, the geo-information is processed to identify the current and destination locations on the map (step 2). Then, the shortest route is calculated (step 3) based on which motion commands (i.e., left, right) are generated to navigate the user (step 4). Motion commands are rendered visually to guide the user in the physical environment (step 5). Finally, the user location is constantly updated (step 6) on the display until the user reaches the final destination (step 7).

Resource-intensive tasks (e.g. `MAP`, `SHORTEST PATH`) require offloading to the nearby edge servers to achieve desired performance [3]. Failures on edge servers can affect offloading, causing additional delay [11]. Identifying reliable edge servers is of paramount importance to ensure a good user experience. Furthermore, during mobility, mobile devices connect to different edge servers which have varying levels of resources and reliability. It necessitates a reliable offloading functionality that is robust to changing environmental conditions.

B. Blockchain-based Reputation Systems and Hybrid Smart Contracts

Blockchain is a decentralized network that secures transactions through consensus, where all participating nodes agree on the current blockchain state. It is difficult to tamper transactions without compromising with a majority of nodes on a large-scale public blockchain (e.g. Ethereum). Thus, a public blockchain with consensus ensures tampering-resistance.

A smart contract is a self-executing program that automatically enforces agreed rules when certain events or conditions on the blockchain are met. Thanks to the tamper-resistant property of the blockchain, smart contracts can securely execute transactions that include sensitive information. However, the blockchain imposes long latencies and limits functionalities that smart contracts can provide by excluding non-deterministic operations (e.g. floating-point arithmetic) [20]. Additionally, blockchain is self-contained and accepts only transactions that occur on-chain. Overall, it is unsuitable for complex, latency-sensitive, and off-chain applications.

Hybrid smart contracts (HSC), on the other hand. allow transactions to happen off-chain, avoiding slow consensus. Offloading decisions, which are performance-critical, are conducted off-chain while the sensitive reputation information, which requires trust, is secured on-chain. For instance, reputation information, as a subjective belief about the consistency of past performance, can be considered sensitive information to identify reliable servers for executing tasks. Reputation can be maliciously tampered with to take a competitive advantage over other servers which can lead to less efficient and reliable offloading and potentially to QoS violations. Therefore, a trust-sensitive reputation system is deployed on-chain as an HSC while latency-sensitive offloading is performed off-chain.

III. System Model

Figure 2 illustrates the edge offloading lifecycle model, which manages offloaded tasks and estimates the reliability level of edge servers based on monitored performance. The two main components of our solution are the *reputation state manager* and the *offloading decision engine*. The reputation state manager is deployed as an HSC on the public blockchain network, estimates the critical reliability level of edge servers as a reputation score, and stores it securely on a public blockchain thanks to a consensus mechanism. The decision engine offloads tasks to an off-chain cluster based on reputation scores retrieved from the reputation state manager. The decision engine is often exposed as an intermediate central third-party service [5], making it vulnerable as a single point

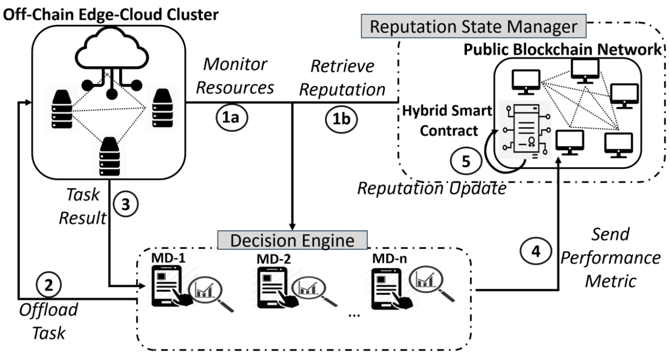


Fig. 2: Edge offloading lifecycle model

of failure in an unreliable environment. In our system, the decision engine is deployed on the mobile device, therefore its design choices should ensure a limited overhead, to guarantee fast decision time even on limited-resource mobile devices.

The lifecycle is executed as follows. In steps 1a and 1b, the mobile device retrieves the reputation score from HSC and monitors resources on the off-chain cluster. Based on the procured information, the mobile device calculates offloading decisions and offloads tasks to the off-chain cluster in step 2. Task results are returned to the mobile device after execution in step 3. The mobile device records the performance metric (e.g., response time) and sends it to the HSC on the blockchain for evaluation in step 4. Finally, HSC compares the received performance metric to the deadlines and updates the reputation score accordingly. The lifecycle is repeated until the application is terminated. Noteworthy to mention, is that blockchain consensus is triggered only upon reputation update but not at reputation retrieval, which makes cached reputation score accessible in (near-)real-time.

A. Queuing and response time

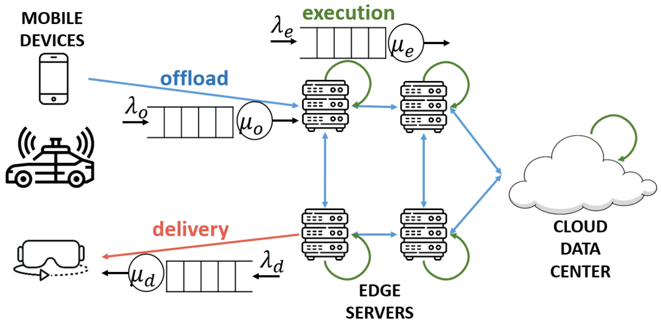


Fig. 3: Dynamic queuing workload model

The workload on the shared infrastructure can be highly dynamic, where response times are hard to predict due to heterogeneous resources and tasks. To describe such dynamic behavior, we employ a queuing theory. Figure 3 illustrates the dynamic queuing workload model at the edge, which consists of three queuing parts. The *task offloading queue* models the task offloading, where multiple mobile task sources generate and offload tasks to remote servers through a shared communication channel. The *task execution queue* model the execution of the task, where the servers share their resources to execute multiple tasks. The *task result delivery* queue models the delivery of task results, where the results are sent back to the sources through the shared channel. The response time is defined as $RT(v,t,h) = T_o(h,t) + T_e(v,t) + T_d(h,t)$ where t is a task, h is a communication channel between pairs that can correspond to devices and servers, $v \in V$ where $V = N \cup \{m\}$ is a set of task execution nodes where a task can be executed on remote edge and cloud servers N and local mobile device m, $T_o(h,t)$ is offloading latency, $T_e(v,t)$ is execution latency and $T_d(h,t)$ is delivery latency. We assume that the channels are on distinct frequencies to avoid interference [21] and employ a nonpreemptive First-Come-First-Served queuing policy that makes the performance predictable, which is important for high-reliability applications.

1) Communication latency: The shared channels in offloading and delivery are modeled as M/M/1 queues, which emulates practical transmission due to fair sharing and different bandwidths [21]. We will use the term communication and symbol $T_c(h,t)$ when referring to offloading and delivery.

The arrival process relates to task generation, modeled as Poisson with an arrival rate $\lambda_c(s)$ where $s \in S$ is a node in the task load generation set $S = G \cup N \cup \{m\}$ which consists of mobile devices G that have the sole function of generating tasks, remote servers N and mobile device m which has decision engine deployed. Task sizes are sampled from the exponential distribution with task size rate $data(t)$, accounting for the diversity of tasks' t resources. Generated tasks occupy shared resources, where bandwidth utilization $U_c(h,t)$ is defined as a ratio of generated tasks $\lambda_c(s) \cdot data(t)$ and total bandwidth $bw_{total}(h)$:

$$U_c(h,t) = \sum_{s \in D_c} \frac{\lambda_c(s) \cdot data(t)}{bw_{total}(h)} \tag{1}$$

where $D_c \subset S$ represents a subset of load generators in a specific communication direction, like $D_o = G \cup \{m\}$ represents task generators and mobile devices that generate tasks in the offloading channel $c = o$, or only remote servers $D_d = N$ on a delivery channel when task results are delivered after execution $c = d$.

The waiting time $w_c(h,t)$ is a delay due to resource sharing between tasks, which is a ratio of the current enqueued tasks and the available bandwidth (difference between total and utilized bandwidth):

$$w_c(h,t) = \sum_{s \in D_c} \frac{\lambda_c(s) \cdot data(t)}{bw_{total}(h) - bw_{util}(h)}, \tag{2}$$

The communication service time $\mu_c(h,t)$ models the actual transmission between devices and remote servers. Communication is subject to the Shannon-Hartley theorem [22] which defines the maximum data transmitted over a noisy link. Hence, the communication service time $\mu_c(h,t)$ is:

$$\mu_c(h,t) = \frac{data(t)}{bw_{avail}(h) \cdot \log_2(1 + \frac{p_c(h)}{n_0 \cdot bw_{avail}(h)})} \tag{3}$$

where n_0 is the noise spectral density and p_c channel transmission power. Finally, the total communication latency $T_c(h,t)$ is a sum of communication waiting time $\mu_c(h)$ and service time $w_c(h)$ such as $T_c(h,t) = \mu_c(h,t) + w_c(h,t)$.

2) Execution latency: The execution on the shared infrastructure is represented as a queuing M/M/1 network. Each server is a queue with independent rates and is interconnected with other queues to form a network [23]. Remote server $n \in N$ utilization is accumulated load and is defined as:

$$U_e(v) = \sum_{s \in S} \sum_{t \in \mathcal{T}(v)} \frac{\lambda_e(s) \cdot MI(t)}{MIPS(v)}, \tag{4}$$

where $MI(t)$ is the number of instructions for task t and $MIPS(v)$ represents the capacity in terms of millions of instructions per second, $\lambda_e(v)$ is the arrival rate of the task and $\mathcal{T}(v)$ is a set of tasks assigned to the server v.

The waiting time $w_e(v)$ is the delay in task execution due to resource contention and is defined as:

$$w_e(v,t) = \frac{\sum_{s \in S} \lambda_e(s) \cdot MI(t)}{1 - U(v)} \tag{5}$$

The actual execution is defined as the service time $\mu_e(n,t)$, which is the ratio between the task load t and the server capacity v:

$$\mu_e(v,t) = \frac{MI(t)}{MIPS(v)}, \tag{6}$$

Finally, we define execution latency $T_e(v,t)$ based on the execution waiting and service times as $T_e(v,t) = w_e(v,t) + \mu_e(v,t)$.

B. Battery lifetime

Mobile devices are battery-powered, thus energy saving is critical. We introduce energy models of local execution and network transmission, major drivers of mobile energy consumption [24]. We assume a mobile multicore execution power model [25] with power states [26]:

$$p_e(m) = b_{cores(m)} + \sum_{i=0}^{cores(m)} (\beta_{U_e(m)} \cdot U_e(m)) + \beta_{base} \cdot \frac{T_{idle}}{C} \tag{7}$$

where $cores(m)$ is the number of CPU cores on mobile device m, $U_e(m)$ utilization per core, $\beta_{U_e(m)}$ and β_{base} are energy coefficients for the operating and idle power states, $b_{cores(m)}$ is a CPU power baseline for a specific number of cores, T_{idle} and C are idle state time duration and number of power state transitions.

The power model for network transmission $p_c(h_m)$ is derived from the Shannon-Hartley theorem:

$$p_c(h_m) = n_0 \cdot bw_{avail}(h_m) \cdot (2^{\frac{Ch(h_m)}{bw_{avail}(h_m)}} - 1). \tag{8}$$

where $bw_{avail}(h_m)$ is the bandwidth on the channel h_m of the mobile device m, and $Ch(h_m)$ is a channel capacity that is an effective limit on bandwidth due to noise. Subsequently, we can define the total energy consumption on the mobile device as the sum of local execution and transmission energy consumption models, defined as $E(v,t,m,h_m) = T_e(v,t) \cdot p_e(m) + T_c(h_m) \cdot p_c(h_m)$. Finally, the battery lifetime of the device $BL(v,t,m,h_m)$ is defined as the ratio between the differentiation of the full battery $bcap$ and the total energy consumption until the time instant τ and full battery capacity as $BL(v,t,m,h_m) = \frac{bcap - \sum_\tau E(v,t,m,h_m)}{bcap}$.

C. Resource utilization cost

Edge and cloud are commercial services that bring monetary costs to mobile users who utilize resources owned by resource providers. Including the monetary objective in the decision-making is important to validate the approach in practical commercial environments where budgetary constraints can impact performance. The utilization cost is defined as:

$$PR(v,t) = \begin{cases} 0 & \text{if local} \\ T_e(v,t) \cdot cost_r(v,t) & \text{if cloud} \\ T_e(v,t) \cdot (cost_r(v,t) + cost_e) & \text{if edge} \end{cases} \tag{9}$$

The first case of local execution has no cost since no remote resources are rented. The second case brings cost when cloud resources are rented for task execution latency time $T_e(v,t)$. The cloud price $cost_r(v,t)$ for the execution task t on the target server v is defined:

$$cost_r(v,t) = cost_{cores}(v) \cdot MI(t) + cost_{stor}(v) \cdot data(t) \tag{10}$$

where $cost_{cores}(v)$, $cost_{stor}(v)$, and $data(t)$ represent cost units for CPU and data storage. The third case accounts for renting edge servers for execution latency time $T_e(v,t)$ where the price includes edge price penalty $cost_e$ for using low-latency service [27].

IV. FRESCO Offloading Solution

A. Reputation state manager

The blockchain-based reputation state manager distributes task incentives to encourage servers' participation in resource-sharing and successful task completion. Task incentives are computed based on the task completion time, meaning that shorter completion results in higher rewards. The rewards stimulate servers to perform task executions reliably and efficiently and compete with each other by offering better performance. The task incentive $inc_\tau(v,t,h)$ at time instant τ is defined as:

$$inc_\tau(v,t,h) = max\{\frac{\nabla - RT(v,t,h)}{\nabla}, 0\} \tag{11}$$

where ∇ is a timing constraint. The task incentive is normalized $[0,1]$ to prevent potential blockchain overflow.

The reputation model has to adhere to blockchain consensus restrictions. The consensus requires that on-chain updates are deterministic, to reach an agreement between blockchain nodes. Therefore, stochastic and floating-point arithmetic is not allowed on-chain [20]. Also, resource and time consumption on the blockchain is limited to prevent resource saturation.

To address the consensus determinism requirement and limited resource consumption, we define a linear reputation model:

$$R_\tau(v,t,h) = R_{\tau-1}(v,t,h) \cdot (1-\omega) + \omega \cdot inc_\tau(v,t,h) \quad (12)$$

where $R_\tau(v,t,h)$ is the current reputation score, $R_{\tau-1}(v,t,h)$ is a previous reputation score, and ω is a weight factor to balance between new and old reputation. Although the model stores only the last score reputation, implicitly it accounts for multiple past values. It can be expanded to the equivalent formula, which tracks historical reputation performance by storing past reputation scores:

$$\begin{aligned} R_\tau(v,t,h) = inc_1(v,t,h) \cdot (1-\omega)^{\tau-1} + \\ \sum_{i=0}^{\tau-2} \omega(1-\omega)^i inc_{\tau-1}(v,t,h) \end{aligned} \quad (13)$$

Reputation scoring ensures that only reliable servers are selected for offloading. Combining both incentives and reputation scores ensures a balanced trade-off where reputation is used as a long-term performance indicator and incentives as immediate short-term rewards to stimulate continuous improvement in server reliability and prioritize reliable servers.

To summarize, the presented reputation-incentive dual approach is encoded as an HSC on the blockchain to asses the reliability level of servers based on past performance. It also ensures trust against reputation malicious tampering for gaining incentives unfairly. The reputation update is according to the presented reputation model based on provided time measurements that are acquired off-chain from mobile devices.

B. Offloading decision engine

Our goal is to efficiently offload tasks to minimize application response time and resource costs and maximize device battery. Therefore, we transform these individual objectives as a constraint optimization problem:

$$\begin{aligned} \min \quad & \sum_{t\in A}\sum_{v\in V} RT(v,t,h) \\ \max \quad & \sum_{t\in A}\sum_{v\in V} BL(v,t,m,h_m) \\ \min \quad & \sum_{t\in A}\sum_{v\in V} PR(v,t) \\ \text{s.t.} \quad RT(v,t,h) \le \nabla, \quad & \forall v \in V, t \in A \\ BL(v,t,m,h_m) > 0, \quad & \forall v \in V, t \in A \\ PR(v,t) \le pr, \quad & \forall v \in V, t \in A, \\ AP_\tau \le D & \end{aligned} \quad (14)$$

where A is a task set of certain application, and AP_τ is an overall application response time until τ time instant. ∇, D and pr represent task timing constraint, application time deadline, and price constraint that can be application-dependant (e.g., 1500 ms reaction time in a traffic safety [28]), user-defined, or defined by developers for testing purposes. Battery lifetime is limited on mobile device m; thus, the goal is to avoid total discharge. Therefore, our main objective function is a linear combination of the objectives mentioned earlier:

$$\begin{aligned} score(v,t,m,h) = \alpha(RT(v,t,h) - RT(\hat{v},t,h)) + \\ \beta(BL(v,\hat{t},m,h_m) - BL(v,t,m,h_m) + \\ \gamma(PR(v,t) - PR(\hat{v},t))) \end{aligned} \quad (15)$$

where α, β, and γ are user-defined weight factors for response, battery, and resource cost respectively ($\alpha + \beta + \gamma = 1$). Objectives with caret symbols are local optimum values. The goal is to find server n that minimizes the value of the $score$. The weight factors can be fine-tuned according to user preferences and subject to sensitivity analysis. However, we fix the weight factors and justify them accordingly in our experimental evaluation, in the subsection V-B3.

1) SMT encoding: Encoding is necessary to translate Equations 14 and 15 into a form, known as SMT formulas, that a target solver can automatically solve. The SMT combines first-order Boolean logic with constraint programming to express resource constraints and deadlines of real-time system [29]. The SMT is lighter than machine learning solutions that are usually exposed as central third-party services [5], and it is suitable for less powerful devices [30]. Additionally, we encode infrastructure capacities, task requirements, and servers' reputation as in Equation 16. It combines them all and uses an SMT solver to find a reliable edge server.

$$\begin{aligned} reputation: (R_\tau(v,t,h) \ge rp) \wedge (0 \le rp \le 1) \\ batteryLife: (BL(v,t,m,h_m) \cdot bcap - E(v,t,m,h_m)) \ge 0 \\ storageLimit: \sum_{t\in O_\tau} data(t) \le stor(v) \\ cpuLimit: \sum_{t\in O_\tau} MI(t) \le cpu(v) \\ memoryLimit: \sum_{t\in O_\tau} mem(t) \le mem(v) \\ taskReady: \sum_{t\in O_\tau} (\delta_{in}(t) = \emptyset \wedge t \notin O_{<\tau}) \end{aligned} \quad (16)$$

The *reputation* constraint refers to server reputation which has to be above a certain threshold. To determine the reputation threshold rp, we apply k criteria from [31] where top k servers with the highest reputation score will be considered. We take a reputation score, which is minimum among k servers as the reputation threshold rp. Here, the *batteryLife* constraint verifies that the mobile device's battery is not drained completely. The *storageLimit* constraint verifies that the input and output data of all offloaded tasks O_τ until time instant τ does not exceed storage capacity on v target server. Similarly, CPU and memory capacities are labeled as *cpuLimit* and *memoryLimit* respectively. Finally, the *taskReady* label indicates that the application task is ready for offloading only when tasks' input dependencies $\delta_{in}(t)$ on prior tasks are completed (i.e., empty set) and the current task t was not part of a previous executed task set $O_{<\tau}$ before time instant τ. Finally, we combine Equation 14, 15, and 16 with logical AND

Algorithm 1 FRESCO Algorithm

```
1: procedure FRESCO($candList, currSite, reps, tasks, constr, \alpha, \beta, \gamma$)
2:     $transactions = list()$
3:     for each $task$ in $tasks$ do
4:         for each $candSite$ in $candList$ do
5:             if $RT(task, candSite, currSite) \leq optRT$ then
6:                 $optRT = RT(task, candSite, currSite)$
7:             end if
8:             if $EC(task, candSite, currSite) \leq optEC$ then
9:                 $optEC = EC(task, candSite, currSite)$
10:            end if
11:            if $PR(task, candSite, currSite) \leq optPR$ then
12:                $optRT = PR(task, candSite, currSite)$
13:            end if
14:        end for
15:        for each $candSite$ in $candList$ do
16:            $score(candSite) = \alpha(RT(task, candSite, currSite) - optRT) + \beta(EC(task, candSite, currSite) - optEC) + \gamma(PR(task, candSite, currSite) - optPR)$
17:        end for
18:        do
19:            if $candList.empty()$ then
20:                break
21:            end if
22:            $selSite = SMTSOLVING(score, candList, reps, constr)$
23:            if $OFFLOAD(selSite, task)$ then
24:                $d = compTaskConstrMeasure(selSite, task)$
25:                $transactions.append((d, selSite))$
26:                break
27:            end if
28:            $transactions.append((0, selSite))$
29:            $score.pop(selSite)$
30:            $candList.pop(selSite)$
31:            $reps.pop(selSite)$
32:        while True
33:    end for
34: return $transactions$
35: end procedure
```

operator into a single SMT logical formula. The final result of verifying the formula should be a reliable server location for offloading. However, solving the optimization function in Equation 14 is NP-hard, which is very time-consuming and impractical for real-time systems. We propose an online algorithm based on a heuristic in the next section, which can find a feasible solution in a reasonable amount of time.

C. FRESCO Algorithm

The offloading algorithm needs to solve the objective function and respect application deadlines, task timing constraints, and resource constraints. Therefore, we propose the FRESCO algorithm (Algorithm 1) for performing reliable edge offloading decisions. Inputs are the list of candidate servers, the server where the previous task was executed ($currSite$), the reputation scores per server, a list of tasks, a list of constraints, and user-defined weights. First, we declare a transaction list recording every offloading attempt and its associated constraint (line 2). Then, we compute local optima for each objective (lines 6-12), which are used to calculate servers' optimization score (line 16) (first *for* loop on line 3). We iterate until the candidate list is empty or the task is successfully offloaded (*do-while* loop on line 18). If the candidate list is empty (line 19) then it exits from the *do-while* loop and returns accumulated transactions. Otherwise, the SMT solver on line 22 selects the server. If offloading fails, then the server is removed from the candidate list and its associate objective values (lines 28-31) and loops back on line 18. If offloading succeeds, the difference between execution time and the constraint ∇ is computed (line 24) and appended to the transaction list (line 25). The offloading transaction list is returned (line 34) and forwarded to the HSC for reputation update.

The computational complexity of the FRESCO depends on $|T|$, which is the cardinality of the set of application tasks T, and $|N|$, which is the cardinality of the set of nodes N. This can be seen by the *for* loop on line 3, that is executed $|T|$ times, and *for* loops on lines 4, 15, that iterate over N set. Also, *do-while* loop on line 18 is executed $|N|$ times in the worst case. However, the most impacting factor on FRESCO complexity is the complexity of the SMT solver (SMTSOLVING function on line 22). Since SMT solving generalizes the boolean satisfiability problem (SAT), which is known to be NP-complete, solving SMT is NP-hard. The SMT solving complexity depends on multiple factors, such as heuristic space search, clause learning, and problem size and structure [32]. Therefore, the selection of an SMT solver has a strong impact on performance [33]. Works like [30] show the applicability of SMT solvers to latency-critical settings such as mobile edge offloading. We will empirically evaluate FRESCO overhead, including the SMT solver in our experiment.

V. Experimental Evaluation

A. Implementation and testbed

Simulated off-chain edge cloud clusters and decision engine are developed in Python and evaluation is performed on a laptop machine with a dual-core CPU of 2.8GHz and 16 GB RAM. The Ganache blockchain emulator and HSC reputation state manager contract are deployed on an AMD64 server with a 40-core 1.80GHz CPU and 128Gb RA. The basic distributed testbed setting reflects our edge offloading lifecycle model presented in III. The decision engine connects to the Ganache when reputation needs to be updated and stored during runtime. Ganache executes the blockchain consensus and returns confirmation. The infrastructure is simulated based on the OpenCellID dataset [34] which contains radio cell tower locations geographically distributed over vast areas. The workload on the nodes is simulated through the queuing network (Section III-A). For SMT solving, we use Z3 as SMT solver [33]. We used the Ganache [1] blockchain emulator as a blockchain and implemented a real-world HSC in the Solidity[2]. Using an emulator instead of a real blockchain is due to the limited number of Ethereum tokens available, which prevents repeated experiments for statistical significance. We assume Proof-of-Authority (PoA) consensus, popular in both private and public Ethereum whose consensus delay is around 4 seconds [35]. Developers usually use this type of consensus to get easy access and fast feedback. We have open-sourced our prototype publicly[3].

B. Experimental design and setup

1) Computing and networking infrastructure: Table I shows the target infrastructure configuration. It reflects our infrastructure's configurations of different edge, cloud, and mobile

[1] https://archive.trufflesuite.com/ganache/

[2] https://soliditylang.org/

[3] https://github.com/jzilic1991/hybrid-edge-blockchain

TABLE I: Computing infrastructure

Node class	CPU cores	CPU (GHz)	RAM (GB)	Storage (GB)
ED server	8	2100	8	300
EC server	16	2800	16	150
ER server	4	1800	8	150
CD server	64	2400	128	1000
Mobile device	2	1800	8	16

TABLE II: Empirical latency measurements as constraints and deadlines from real-world applications in milliseconds

	Intra(D=108)		**MobiAR(D=400)**		**NaviAR(D=800)**	
∇	Proc	Net	Proc	Net	Proc	Net
Edge	18	15	2-20	15	250-300	300-400
Cloud	2-20	90	1	300	2-20	1000-1500
Mobile	300	0	300	0	800	0

TABLE III: Task specifications

Type	CPU	Input data	Output data
DI	100-200 M cycles	15-20 KB	25-30 KB
CI	550-650 M cycles	4-8 KB	4-8 KB
Moderate	100-200 M cycles	4-8 KB	4-8 KB

TABLE IV: Intrasafed task specifications

Task	Type	RAM	Offloadable
LOAD_MODEL	Moderate	1 GB	False
UPLOAD	DI	1 GB	True
ANALYZE	CI	4 GB	True
AGGREGATE	CI	2 GB	True
SEND_ALERT	Moderate	1 GB	True

devices. We classified servers into several classes to capture resource heterogeneity. The mobile device has limited resources compared to other nodes. The ED is an edge database server that has fast-speed network access and large data storage capacity to handle data-intensive (DI) tasks; the second one represents a computational-intensive server (EC) that has a high number of CPU cores to cope with computational-intensive (CI) tasks, and the third one represents an edge regular server (ER) with moderate resource capacities. The cloud server is the most resourceful one but has higher latency.

We adopt processing and network latencies as application QoS deadlines from three real-world use cases, described in Table II. In Table II, ”Proc” indicates the processing timing constraint, while ”Net” is the networking timing constraint. Note that we distinguish task timing constraint ∇ from application deadline D. We measure QoS violations against application deadlines.

2) Mobile DAG applications: Mobile applications are modeled as DAGs which is a common method of mobile application modeling [30], [36]. These applications exhibit a pipeline workflow structure, which is typical for AI-based applications. Table III specifies task categories from which the applications are constructed, while Tables IV, V, and VI describe structures of selected applications. We selected the following applications because they are latency-sensitive, and are part of an emerging market where edge computing is a key technology enabler.

(i) Intrasafed: It is a traffic safety application [28], which employs an AI-based object detection that detects pedestrians at intersections, notifying drivers in real-time to prevent accidents. We simulated the application in our simulator with latency measurements from the original work, presented in Table II. It has a deadline of D = 108 ms for the average drivers' notification latency via 5G networks. **(ii) MobiAR:** It is a generic AR object detection application [2], which we extracted its application structure and executed in our simulator. The real latency measurements are extracted from the work and presented in Table II. The application requires a deadline of D = 400 ms to meet the applications' inference latency. **(iii) NaviAR:** It is an AR live navigation executed on AR HoloLens glasses [3]. We simulated the structure in our simulator backed by latency measurements as constraints listed in Table II. It requires a deadline of 800 ms which is equal to the local execution time on AR glasses.

3) Parameters: Parameters used in our experiment are defined in Table VII. The Poisson task arrival rate λ range is selected so it can scale to different workload intensities. α, β, and γ values are selected as a representative case of the user's preferences about preferring fast response and willingness to pay a higher price for it ($\alpha > \beta > \gamma$) since we target latency-sensitive applications. BL is the initial device battery capacity. Reputation weight factor ω is taken from [20] which accounts for a relatively conservative reputation system to mitigate volatility in a crowdsourced system. $cost$ coefficients for CPU and storage are taken from Google Cloud [37] which is one of the most commonly used cloud providers. Energy coefficients of β_{base}, β_{U_e} and p_{cores} are taken from [25], [26] which are validated against real mobile equipment.

4) Datasets: We employed the Skype availability dataset [38] to model the system's availability. The motivation for selecting Skype dataset is because it shares edge characteristics like geo-distribution, heterogeneity, large number of nodes, and it constitutes the middle ground in availability ratio (60-70%) and latency (up to $\sim$50 ms) compared to other infrastructure [7]. Traces are collected over 2,081 servers for 400 days and contain availability time intervals that are associated with each node. Nodes have different lifespans and hence they are normalized within the $[0, 1]$ time range interval. Adopting such availability datasets from distributed systems that share similar characteristics is common in edge computing research [7], [39] due to the lack of publicly available datasets.

Edge and cloud deployment follow cellular base station locations from OpenCellID. OpenCellID is an open cellular database containing datasets of cell tower geolocations that mobile operators publicly publish. It is used in generating infrastructure topologies under edge computing settings [40]. We selected a dataset that contains around $3,500$ cell tower locations and randomly filtered them out to match the number of $2,081$ Skype nodes for one-to-one availability trace mapping. We clustered the entire network into 30 cell clusters using the k-means clustering algorithm as illustrated in Figure 4. In such a deployment, location-based mobility is simulated where a mobile device visits each cell cluster and offloads tasks on remote servers. Mobile device dwelling time in each cell is evenly distributed throughout the entire simulation. Each cell cluster location has edge node classes such as ER, ED, and

TABLE V: MobiAR task specifications

Task	Type	RAM	Offloadable
UPLOAD	Moderate	1 GB	False
EXTRACT	CI	2 GB	True
PROCESS	CI	2 GB	True
DATA	DI	1 GB	True
DOWNLOAD	DI	1 GB	False

TABLE VI: NaviAR task specifications

Task	Type	RAM	Offloadable
MAP	DI	1 GB	True
GUI	Moderate	1 GB	False
COORDINATION	CI	4 GB	True
SHORTEST_PATH	CI	2 GB	True
MOTION_COMMAND	CI	1 GB	True
VIRTUAL_GUIDANCE	Moderate	1 GB	False
RUNTIME_LOCATION	CI	1 GB	True
DISPLAY	Moderate	1 GB	False

EC which are randomly associated with OpenCellID nodes and a single accessible cloud server. Remote servers have an associated reputation score, which is stored on a public blockchain that is globally accessible.

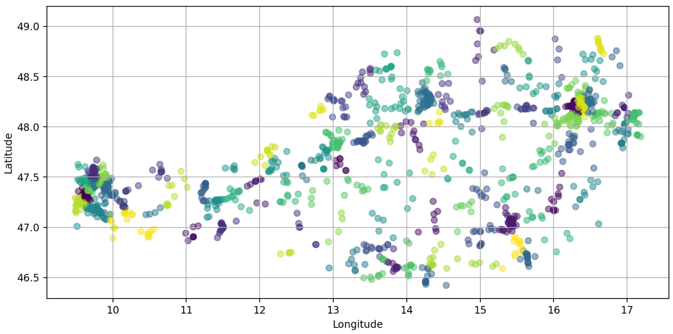


Fig. 4: Cell tower locations from OpenCellID dataset [34]

5) Baselines: We compare *FRESCO* with the following three baseline algorithms.

- ***MINLP*** is a mixed integer non-linear programming-based method that formulates constraint offloading optimization problems without reputation. The MINLP approach is the most common modeling method for offloading optimization [41].
- ***SQ EDGE*** [31] considers reputation and queuing time on edge nodes, and it is utilized in blockchain-based vehicular ad-hoc networks. The method considers only local and edge offloading, as in naive offloading approaches used when resources are limited for decision-making.
- ***MDP*** is a common method for modeling offloading [5]. Reputation is encoded as transition probability, remote servers represent states, and objectives are modeled as reward functions. The modeling is similar to existing work that targets reliable offloading [36].

C. Analysis of results

For each experimental run, we execute 100 applications sequentially and average results over 100 runs to obtain statistically significant results.

TABLE VII: Simulation and algorithmic parameters

Parameter	Value
λ	[10, 20]
α	0.5
β	0.4
γ	0.1
BL	1000
ω	0.3
$cost_{cores}$	0.023
$cost_{stor}$	0.776
β_{base}	625.25 10^{-3}
β_{U_e}	6.9305 10^{-3}
p_{cores}	0.073 10^{-3}

1) ***Response time:*** Figure 5 illustrates the response time performance of offloading decision engines in Intrasafed, MobiAR, and NaviAR applications respectively. The worst-performing decision engine is MDP whose response time is 53.11, 73.86, and 104.06 seconds for three applications, respectively. Whereas the SQ EDGE decision engines have average response times of 41.99, 46.96, and 65.12 seconds. However, SQ Edge has a higher deviation in its response time compared to others (6.16, 4.03, and 5.99 seconds). Although the SQ EDGE approach is reputation-aware, its primary target is to identify malicious servers instead of reliable offloading in terms of QoS violations caused by failed or high-loaded sites. Thus, this leads to more volatile performance as observed. The MINLP decision engine yielded the second-best approach with 24.34, 28.91, and 18.72 seconds. The best performance was achieved with the NaviAR application (18.72 seconds) which is unexpected since NaviAR has the most complex structure. The possible explanation is that the edge servers in the last visited cells were more loaded which limits resource capacity. It could deter MINLP from taking offloading decisions on the edge and rather opt for local execution or select a far-distant cloud. 75% of the offloading attempts in the last cell were concentrated on cloud and mobile. Although MINLP does not perceive reputation, selecting both mobile devices and the cloud are safe for offloading and avoids offloading failures on the failure-prone edge which in the last two cells have limited availability $(12 - 25\%)$. Offloading failure would impose a longer response time as seen in Intrasafed and MobiAR applications. Lastly, our FRESCO solution outperforms other decision engines due to frequent offloading on more reliable servers which resulted in response time performance of 6.75, 11.61, and 17.81 seconds.

2) ***Battery lifetime:*** Figure 6 illustrates the battery performance of offloading decision engines in all three applications. The SQ EDGE decision engine drains the device battery the most, with 96.38%, 95.33%, and 93.34%. A higher rate of failed offloading attempts drains the energy more than the longer response time (Figure 5) in MDP whose battery lifetimes are 98.08%, 95.64%, and 93.03%. MINLP and FRESCO, on the other hand, have battery lifetimes that reflect response time performance from Figure 5. MINLP battery lifetimes are 98.28%, 97.54%, and 98.42% while FRESCO has the highest battery lifetime of 99.47%, 99.06%, and 98.43%.

3) ***Resource utilization cost:*** Figure 7 shows the resource utilization cost of all four approaches. Here. MDP incurs

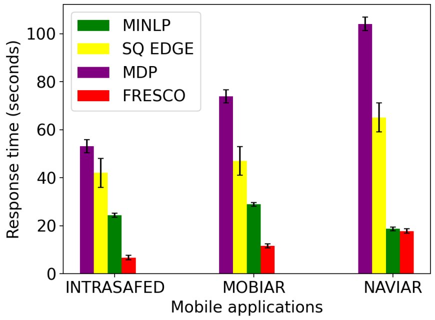


Fig. 5: Response time

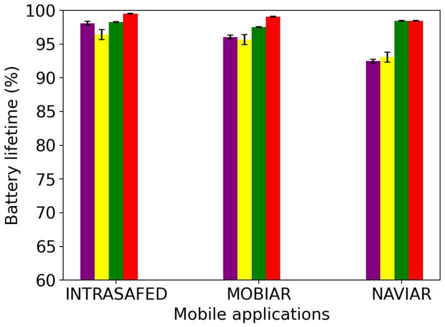


Fig. 6: Battery lifetime

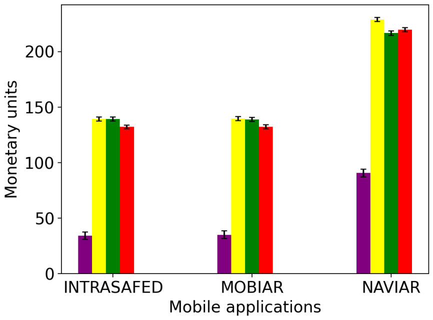
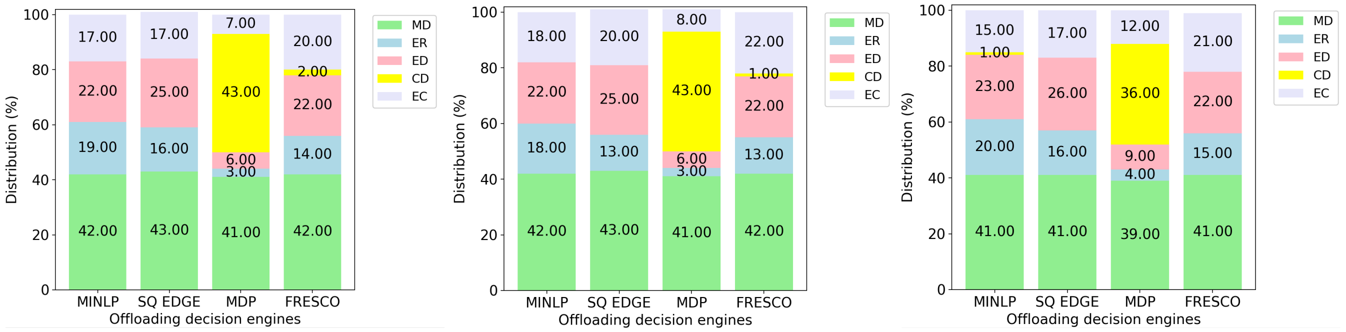


Fig. 7: Resource utilization cost



(a) Intrasafed (b) MobiAR (c) NaviAR

Fig. 8: Offloading distribution

lower resource utilization costs (ranging from 34.4 to 90.69 monetary units) although it has poor response time (see Figure 5). This is because it offloads to a highly available cloud node which has a lower utilization cost. SQ EDGE is the most expensive solution due to failed offloading attempts and fixed k parameter which does not scale with several nodes, and thus limits alternative servers for offloading consideration and can lead to potentially higher costs. According to SQ EDGE, best k sites are the most reputable ones but can be highly loaded and limit re-offloading alternatives in case of failed or untimely execution. SQ EDGE monetary costs are 139.3, 139.16, and 228.23 units for three applications. When comparing MINLP and FRESCO, FRESCO emerged as the second-best cost-effective solution and cheaper than MINLP in Intrasafed (132.22 vs 139.33 units) and MobiAR (132.38 vs 139.08 units) use cases but worse when offloading NaviAR (219.89 vs 216.76 units). In NaviAR's case, MINLP is slightly cheaper than FRESCO because it offloaded a minor portion of tasks on the cloud which is cheaper than Edge. FRESCO did not yield the overall best cost-effectiveness because hyperparameters are tuned so it prefers faster and energy-efficient solutions rather than low-cost sites.

*4) **Offloading distribution**:* Figure 8a) shows the offloading distribution analysis for three applications. This analysis shows the distribution of application tasks to different nodes in our infrastructure. In the Intrasafed use case, the MDP was worse-performant because most of the offloading decisions were targeting highly available cloud (43%) instead of expensive edge servers and thus is the cheapest solution (Figure 7) and not necessarily the least energy-efficient (Figure 6). SQ EDGE, with a similar offloading distribution composition to MINLP, only considers k sites with the highest reputation and selects the shortest queue waiting time. k parameter is fixed and does not scale with several nodes which can limit the number of alternative servers and exclude viable ones that are less loaded and sufficiently reliable. MINLP, on the other hand, does not restrict its offloading options. FRESCO, comparably to the previous two aforementioned baselines, is more flexible and utilizes all site types, including cloud (2%) for CI tasks when edge servers are less reliable, also less reliant on moderate ER sites (14% compared to 19% and 16% in MINLP and SQ EDGE respectively), and utilizes resource-rich EC sites more frequently (20% compared to 17% in MINLP and SQ EDGE). FRESCO's offloading distribution composition is similar in the MobiAR case (Figure 8b) and reflects FRESCO's higher performance in both applications. In the NaviAR case (Figure 8c), MINLP and FRESCO have different offloading distribution compositions but performance-wise are comparable (Figure 5). FRESCO balances reliability and performance, where the most reputable servers are not necessarily efficient ones. MINLP is reputation-oblivious but selecting the most efficient servers can be sometimes beneficial if the underlying infrastructure is more reliable and has fewer failures or less volatile load.

*5) **QoS violations**:* Figure 9 illustrates QoS violation results. MDP has the highest violation rate because of frequent offloading on highly available clouds. Leading to fewer failures but frequent violations. The next better performant solution is SQ EDGE, with a violation rate between 18.9% (in the NaviAR case) and 15.9% (in the MobiAR case). MINLP shows better performance with violation rates of 12.3%, 9.2%,

TABLE VIII: Hybrid smart contract usage cost on Ethereum

Function name	Gas consumption (Wei)
registerNode	$21,503$ Wei
unregisterNode	$21,204$ Wei
getNodeCount	$21,604$ Wei
getNode	$21,204$ Wei
updateNodeReputation	$21,638$-$29,984$ Wei
getReputationScore	$21,204$ Wei
resetReputation	$21,484$-$25,544$ Wei

and 0.1% in Intrasafed, MobiAR, and NaviAR respectively. FRESCO has the lowest violation rates in Intrasafed and MobiAR use cases with 7.1% and 3.8% and has a violation rate of 0.4% with a standard deviation of 0.48% in the NaviAR case which is comparable with MINLP.

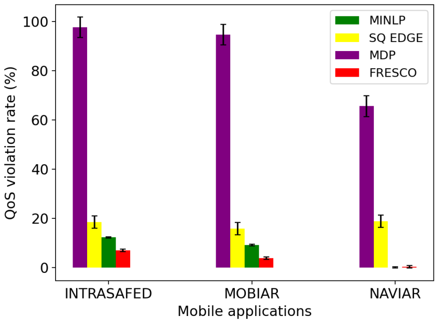


Fig. 9: QoS violations

*6) **HSC overhead**:* HSC blockchain usage costs are expressed as gas consumption and are called Wei. The results are presented in Table VIII for each function. Where range is expressed, it refers to executing from 1 to 30 offloading transactions, as multiple offloading transactions are typically executed for those functions. All HSC functions consume slightly above $21,000$ Wei, which is typical on the Ethereum [42].

*7) **Offloading decision overhead**:* Figure 10 illustrates offloading decision time overhead across different infrastructure sizes on a logarithmic scale. The SQ EDGE is the least complex algorithm since selecting the first k nodes and computing their estimated queue waiting time is relatively straightforward in comparison to other decision engines. The average decision time overhead is 0.048 milliseconds. FRESCO and MINLP decision time overhead are 5.05 milliseconds and 6.57 milliseconds with standard deviations of 5.16 milliseconds and 3.07 milliseconds respectively, making them comparable. MDP has the highest overhead, which is an average of 1373.83 milliseconds, due to state space explosion when offloading on a larger number of nodes. In summary, FRESCO has an average decision time overhead of 5.05 milliseconds, which makes it suitable for offloading latency-sensitive applications.

Summary: FRESCO decreases average response time up to $7.86x$, and increases battery up to 5.4% compared to baselines. It also achieves a low deadline violation rate of 0.4% while maintaining competitive utilization costs. With approximately typical blockchain consumption ($\approx$ $21,500$ Wei) and low average decision time overhead (5.05 milliseconds), FRESCO is suitable for offloading latency-sensitive applications.

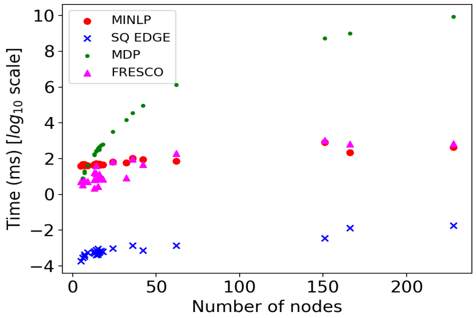


Fig. 10: Offloading decision time in logarithmic scale

TABLE IX: Overview of state-of-the-art literature

Publication	OFF	(H)SC	BLOCK(-REP)	REL
[2], [43], [44]	✓	✗	✗	✗
[13]–[15], [19], [31]	✓	✗	✓	✗
[11], [36], [45]	✓	✗	✗	✓
[46], [47]	✗	✓	✗	✗
[16]–[18], [48]	✗	✗	✓	✓
[49]	✓	✗	✓	✗
This work	✓	✓	✓	✓

VI. Discussion and Limitations

Limitations of our experiment are the usage of a blockchain emulator and the selection of a single consensus mechanism. In the former, the real Ethereum requires tokens, which carries financial cost and prevents us from collecting sufficient traces to strengthen the experimental evaluation. In the latter, the selected lightweight PoA consensus is used in real Ethereum [35], has relatively shorter latencies compared to computational-intensive ones (e.g., Proof-of-Work), and does not test our solution on varied consensus latencies. Also, the choice to use public blockchain instead of private blockchain is that we targeted more open and public settings instead of being contained within certain organizations (e.g., enterprises).

Hyperparameter optimization of FRESCO could potentially increase the performance but also introduce overhead, thus endangering online applicability. User-defined hyperparameters enable flexibility to fit different needs (e.g., preferring cheaper but slower service over expensive and faster one). The empirically obtained hyperparameters reflect a representative scenario where latency is preferred over other objectives, which is typical for latency-sensitive applications. Theoretically, without relying on user-defined values, mobile devices need to adjust them by incorporating data-driven optimization.

VII. Related Work

Table IX compares FRESCO with the literature concerning offloading OFF, smart contracts $(H)SC$, blockchain-enabled or based-reputation $BLOCK(-REP)$, and reliability REL.

Offloading survey [43] shows that applying deep reinforcement learning solutions (DRL) and their variants has become common recently due to their adaptability to changing and dynamic conditions. They do not consider edge failures and are deployed as an intermediary between devices and servers which is risky in the unreliable environment that can lead to a single-point-of-failure, sometimes requiring model re-training when topology or environment changes drastically [44].

Blockchain-enabled edge offloading approaches [13]–[15] enhance reliability and efficiency in mobile edge computing and vehicular edge networks where blockchain and smart contracts make offloading more secure and trustworthy. They do not provide specifics about real-world smart contract implementation or they deploy an offloading framework as a smart contract which is unrealistic since smart contracts prohibit nondeterministic and stochastic computations that conflict with the determinism requirement of blockchain consensus protocols. Blockchain-based reputation offloading was proposed for specific scenarios that require fast responses like vehicular networks [31] and IIoT [19]. Solutions are applied in private environments (e.g. factories, enterprises) while ignoring the consensus overhead, and using reputation against malicious actors rather than against failures in unreliable environments. Also, other blockchain-based reputation works are applied for selecting reliable edge servers, ranging from IoT [48], federated edge learning [18] to vehicular networks [16]. However, they did not target reliability in edge offloading in terms of edge failures and do not employ HSC, which forces them to compute reputation off-chain rather than on-chain which can lead to potential risks (e.g. collusion) [50].

Some edge offloading approaches [11], [45], [51] tried to solve reliability issues in terms of failure and recovery probabilistic models, or predict edge failures based on historical data [36]. The aforementioned works did not prove or evaluate their solutions in distributed unreliable edge scenarios where the device moves and reacts to different environments.

Works [46], [47] implemented smart contracts on hybrid blockchain architecture to reconcile conflicting objectives such as trust on one side and performance on the other side. The works are not applied in edge offloading context.

In conclusion, none of the works applied a blockchain-reputation system for enhancing reliability in the edge offloading context. FRESCO uniquely ensures trust for sensitive reputation information on-chain and allows fast performance for latency-sensitive applications off-chain.

VIII. Conclusion

We investigated edge offloading of latency-sensitive mobile applications on the distributed unreliable edge. Edge offloading is formulated as a constraint optimization problem that balances between response time, battery, and utilization monetary cost objectives. Formulation incorporates critical QoS deadlines that have to be respected, and reputation scores to identify reliable edge servers based on past performance.

The FRESCO consists of a reputation state manager and a decision engine. The reputation state manager is implemented as a hybrid smart contract, which stores sensitive reputation scores on-chain against tampering, and enables an SMT-based decision engine to compute offloading decisions off-chain on reliable servers without being hindered by blockchain consensus. The presented solution balances reliability and performance where trust is required against tampering.

FRESCO was evaluated against baselines with simulated applications, a dynamic queueing workload, Skype availability traces, and large-scale infrastructure from the OpenCellID. We also discussed limitations like the blockchain emulator, hyperparameter optimization, and using a single consensus mechanism which will be addressed in our future work.

Acknowledgments

This work is partially funded by Josip Zilic's netidee scholarship by the Internet Foundation Austria.

References

[1] D. Marimon, C. Sarasua, P. Carrasco, R. Álvarez, J. Montesa, T. Adamek, I. Romero, M. Ortega, and P. Gascó, "Mobiar: tourist experiences through mobile augmented reality," *Telefonica Research and Development, Barcelona, Spain*, 2010.

[2] J. Ren, L. Gao, X. Wang, M. Ma, G. Qiu, H. Wang, J. Zheng, and Z. Wang, "Adaptive computation offloading for mobile augmented reality," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 5, no. 4, pp. 1–30, 2021.

[3] Y. Wang, T. Yu, and K. Sakaguchi, "Context-based mec platform for augmented-reality services in 5g networks," in *2021 IEEE 94th Vehicular Technology Conference (VTC2021-Fall)*. IEEE, 2021, pp. 1–5.

[4] Q. Li, S. Wang, A. Zhou, X. Ma, F. Yang, and A. X. Liu, "Qos driven task offloading with statistical guarantee in mobile edge computing," *IEEE Transactions on Mobile Computing*, vol. 21, no. 1, pp. 278–290, 2020.

[5] H. Lin, S. Zeadally, Z. Chen, H. Labiod, and L. Wang, "A survey on computation offloading modeling for edge computing," *Journal of Network and Computer Applications*, vol. 169, p. 102781, 2020.

[6] T. Long, Y. Ma, Y. Xia, X. Xiao, Q. Peng, and J. Zhao, "A mobility-aware and fault-tolerant service offloading method in mobile edge computing," in *2022 IEEE International Conference on Web Services (ICWS)*. IEEE, 2022, pp. 67–72.

[7] A. Aral and I. Brandić, "Learning spatiotemporal failure dependencies for resilient edge computing services," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 7, pp. 1578–1590, 2020.

[8] S. Tuli, G. Casale, and N. R. Jennings, "Pregan: Preemptive migration prediction network for proactive fault-tolerant edge computing," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2022, pp. 670–679.

[9] H. Wu, "Performance modeling of delayed offloading in mobile wireless environments with failures," *IEEE Communications Letters*, vol. 22, no. 11, pp. 2334–2337, 2018.

[10] L. Zhao, B. Li, W. Tan, G. Cui, Q. He, X. Xu, L. Xu, and Y. Yang, "Joint coverage-reliability for budgeted edge application deployment in mobile edge computing environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 3760–3771, 2022.

[11] J. Liang, B. Ma, Z. Feng, and J. Huang, "Reliability-aware task processing and offloading for data-intensive applications in edge computing," *IEEE Transactions on Network and Service Management*, vol. 20, no. 4, pp. 4668–4680, 2023.

[12] Y. Siriwardhana, P. Porambage, M. Liyanage, and M. Ylianttila, "A survey on mobile augmented reality with 5g mobile edge computing: Architectures, applications, and technical aspects," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 2, pp. 1160–1192, 2021.

[13] J. Ma, Y. Yi, W. Zhang, Y. Sun, and G. Zhang, "Blockchain-based task offloading for mobile edge computing networks with server collaboration," *2024 5th Information Communication Technologies Conference (ICTC)*, pp. 221–226, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:271407903

[14] J. Shi, J. Du, Y. Shen, J. Wang, J. Yuan, and Z. Han, "Drl-based v2v computation offloading for blockchain-enabled vehicular networks," *IEEE Transactions on Mobile Computing*, vol. 22, pp. 3882–3897, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:247090469

[15] Y. Zhou, X. Li, H. Ji, and H. Zhang, "Blockchain-based trustworthy service caching and task offloading for intelligent edge computing," *2021 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:246478911

[16] L. Sun, Q. Yang, X. Chen, and Z. Chen, "Rc-chain: Reputation-based crowdsourcing blockchain for vehicular networks," *Journal of network and computer applications*, vol. 176, p. 102956, 2021.

[17] Z. Zhou, M. Wang, C.-N. Yang, Z. Fu, X. Sun, and Q. J. Wu, “Blockchain-based decentralized reputation system in e-commerce environment,” *Future Generation Computer Systems*, vol. 124, pp. 155–167, 2021.

[18] J. Kang, Z. Xiong, X. Li, Y. Zhang, D. Niyato, C. Leung, and C. Miao, “Optimizing task assignment for reliable blockchain-empowered federated edge learning,” *IEEE Transactions on Vehicular Technology*, vol. 70, no. 2, pp. 1910–1923, 2021.

[19] S. Iqbal, R. M. Noor, A. W. Malik, and A. U. Rahman, “Blockchain-enabled adaptive-learning-based resource-sharing framework for iiot environment,” *IEEE Internet of Things Journal*, vol. 8, no. 19, pp. 14 746–14 755, 2021.

[20] A. Battah, Y. Iraqi, and E. Damiani, “Blockchain-based reputation systems: Implementation challenges and mitigation,” *Electronics*, vol. 10, no. 3, p. 289, 2021.

[21] Q. Fan and N. Ansari, “Towards workload balancing in fog computing empowered iot,” *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 1, pp. 253–262, 2018.

[22] O. Rioul and J. C. Magossi, “On shannon’s formula and hartley’s rule: Beyond the mathematical coincidence,” *Entropy*, vol. 16, no. 9, pp. 4892–4910, 2014.

[23] M. Bramson, *Stability of queueing networks*. Springer, 2008.

[24] M. Tawalbeh, A. Eardley *et al.*, “Studying the energy consumption in mobile devices,” *Procedia Computer Science*, vol. 94, pp. 183–189, 2016.

[25] F. A. Ali, P. Simoens, T. Verbelen, P. Demeester, and B. Dhoedt, “Mobile device power models for energy efficient dynamic offloading at runtime,” *Journal of Systems and Software*, vol. 113, pp. 173–187, 2016.

[26] Y. Zhang, Y. Liu, X. Liu, and Q. Li, “Enabling accurate and efficient modeling-based cpu power estimation for smartphones,” in *2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*. IEEE, 2017, pp. 1–10.

[27] V. De Maio and I. Brandic, “Multi-objective mobile edge provisioning in small cell clouds,” in *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, 2019, pp. 127–138.

[28] I. Lujic, V. D. Maio, K. Pollhammer, I. Bodrozic, J. Lasic, and I. Brandic, “Increasing traffic safety with real-time edge analytics and 5g,” in *Proceedings of the 4th International Workshop on Edge Systems, Analytics and Networking*, 2021, pp. 19–24.

[29] Z. Cheng, H. Zhang, Y. Tan, and Y. Lim, “Smt-based scheduling for multiprocessor real-time systems,” in *2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)*. IEEE, 2016, pp. 1–7.

[30] C. Avasalcai, C. Tsigkanos, and S. Dustdar, “Resource management for latency-sensitive iot applications with satisfiability,” *IEEE Transactions on Services Computing*, vol. 15, no. 5, pp. 2982–2993, 2021.

[31] S. Iqbal, A. W. Malik, A. U. Rahman, and R. M. Noor, “Blockchain-based reputation management for task offloading in micro-level vehicular fog network,” *IEEE Access*, vol. 8, pp. 52 968–52 980, 2020.

[32] R. Robere, A. Kolokolova, and V. Ganesh, “The proof complexity of smt solvers,” in *Computer Aided Verification: 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part II 30*. Springer, 2018, pp. 275–293.

[33] A. Høfler, “Smt solver comparison,” *Graz, July*, p. 17, 2014.

[34] “”opencellid, 2021, (https://opencellid”,” openCellID, 2021, (https://opencellid.org/).

[35] “””ethereum test network” https://medium”,” ”Ethereum Test network” https://medium.com/coinmonks/ethereum-test-network-21baa86072fa (Accessed: 2024-02-07).

[36] J. Zilic, V. De Maio, A. Aral, and I. Brandic, “Edge offloading for microservice architectures,” in *Proceedings of the 5th International Workshop on Edge Systems, Analytics and Networking*, 2022, pp. 1–6.

[37] “””cloud storage pricing” https://cloud”,” ”Cloud Storage pricing” https://cloud.google.com/storage/pricing (Accessed: 2022-30-11).

[38] S. Guha and N. Daswani, “An experimental study of the skype peer-to-peer voip system,” Cornell University, Tech. Rep., 2005.

[39] Z. N. Samani, N. Mehran, D. Kimovski, and R. Prodan, “Proactive sla-aware application placement in the computing continuum,” in *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2023, pp. 468–479.

[40] B. Xiang, J. Elias, F. Martignon, and E. Di Nitto, “A dataset for mobile edge computing network topologies,” *Data in Brief*, vol. 39, p. 107557, 2021.

[41] C. Feng, P. Han, X. Zhang, B. Yang, Y. Liu, and L. Guo, “Computation offloading in mobile edge computing networks: A survey,” *Journal of Network and Computer Applications*, vol. 202, p. 103366, 2022.

[42] “””what is gwei? the cryptocurrency explained” https://www”,” ”What Is Gwei? The Cryptocurrency Explained” https://www.investopedia.com/terms/g/gwei-ethereum.asp (Accessed: 2024-02-07).

[43] Z. Zabihi, A. M. Eftekhari Moghadam, and M. H. Rezvani, “Reinforcement learning methods for computation offloading: a systematic review,” *ACM Computing Surveys*, vol. 56, no. 1, pp. 1–41, 2023.

[44] H. Zhang, R. Wang, W. Sun, and H. Zhao, “Mobility management for blockchain-based ultra-dense edge computing: A deep reinforcement learning approach,” *IEEE Transactions on Wireless Communications*, vol. 20, no. 11, pp. 7346–7359, 2021.

[45] C. Liu and K. Liu, “Toward reliable dnn-based task partitioning and offloading in vehicular edge computing,” *IEEE Transactions on Consumer Electronics*, vol. 70, no. 1, pp. 3349–3360, 2023.

[46] C. Molina-Jimenez, I. Sfyrakis, E. Solaiman, I. Ng, M. W. Wong, A. Chun, and J. Crowcroft, “Implementation of smart contracts using hybrid architectures with on and off–blockchain components,” in *2018 IEEE 8th International Symposium on Cloud and Service Computing (SC2)*. IEEE, 2018, pp. 83–90.

[47] E. Solaiman, T. Wike, and I. Sfyrakis, “Implementation and evaluation of smart contracts using a hybrid on-and off-blockchain architecture,” *Concurrency and computation: practice and experience*, vol. 33, no. 1, p. e5811, 2021.

[48] Y. Yu, S. Liu, L. Guo, P. L. Yeoh, B. Vucetic, and Y. Li, “Crowdr-fbc: A distributed fog-blockchains for mobile crowdsourcing reputation management,” *IEEE Internet of Things Journal*, vol. 7, no. 9, pp. 8722–8735, 2020.

[49] S. Deng, G. Cheng, H. Zhao, H. Gao, and J. Yin, “Incentive-driven computation offloading in blockchain-enabled e-commerce,” *ACM Transactions on Internet Technology (TOIT)*, vol. 21, no. 1, pp. 1–19, 2020.

[50] X. Deng, J. Liu, L. Wang, and Z. Zhao, “A trust evaluation system based on reputation data in mobile edge computing network,” *Peer-to-Peer Networking and Applications*, vol. 13, pp. 1744–1755, 2020.

[51] K. Peng, B. Zhao, M. Bilal, and X. Xu, “Reliability-aware computation offloading for delay-sensitive applications in mec-enabled aerial computing,” *IEEE Transactions on Green Communications and Networking*, vol. 6, no. 3, pp. 1511–1519, 2022.

IX. Biography Section

Josip Zilic is a Pre-Doctoral Researcher at the High-Performance Research Group at TU Wien. His research focuses on applying formal methods in edge offloading to ensure and guarantee performance for latency-sensitive and high-reliability mobile applications. In 2023, he received netidee scholarship from Internet Foundation Austria for his edge offloading doctoral thesis proposal.

Vincenzo De Maio received his PhD in 2016 at the University of Innsbruck, Austria. His research in the area of parallel and distributed systems comprises energy-aware Cloud computing and scheduling. Since 2017, he has been a postdoctoral researcher at the High-Performance Computing Research Group of TU Wien. He authored different conferences and journal publications on energy efficiency and modeling for Cloud, Edge, and Quantum Computing.

Shashikant Illager is an assistant professor at the Informatics Institute, University of Amsterdam, Netherlands. He is a member Multiscale Networked Systems research group. He works at the intersection of distributed systems, energy efficiency, and machine learning. His recent research explores the energy efficiency and performance optimization of data-intensive and distributed AI applications.

Ivona Brandic is a Professor at TU Wien. In 2015 she was awarded FWF START prize, the highest Austrian award for young researchers. She received her PhD degree in 2007 from Vienna University of Technology. In 2011 she received the Distinguished Young Scientist Award from the Vienna University of Technology for her project on the Holistic Energy Efficient Hybrid Clouds. Her main research interests are cloud computing, large-scale distributed systems, energy efficiency, QoS, and autonomic computing.