MARQ: Engineering Mission-Critical AI-based Software with Automated Result Quality Adaptation

Uwe Gropengießer†, Elias Dietz†, Florian Brandherm†, Achref Doula†, Osama Abboud§, Xun Xiao§, Max Mühlhäuser†

Technical University of Darmstadt, Germany
§Huawei Technologies, Germany

***Abstract*—AI-based mission-critical software exposes a blessing and a curse: its inherent statistical nature allows for flexibility in result quality, yet the mission-critical importance demands adherence to stringent constraints such as execution deadlines. This creates a space for trade-offs between the Quality of Result (QoR)—a metric that quantifies the quality of a computational outcome—and other application attributes like execution time and energy, particularly in real-time scenarios. Fluctuating resource constraints, such as data transfer to a remote server over unstable network connections, are prevalent in mobile and edge computing environments—encompassing use cases like Vehicle-to-Everything, drone swarms, or social-VR scenarios. We introduce a novel approach that enables software engineers to easily specify alternative AI service chains—–sequences of AI services encapsulated in microservices aiming to achieve a predefined goal—–with varying QoR and resource requirements. Our methodology facilitates dynamic optimization at runtime, which is automatically driven by the MARQ framework. Our evaluations show that MARQ can be used effectively for the dynamic selection of AI service chains in real-time while maintaining the required application constraints of mission-critical AI software. Notably, our approach achieves a 100× acceleration in service chain selection and an average 10% improvement in QoR compared to existing methods.**

***Index Terms*—Mission-critical AI, Quality of Result, Edge Computing, Approximate Computing, Software Engineering**

I. Introduction

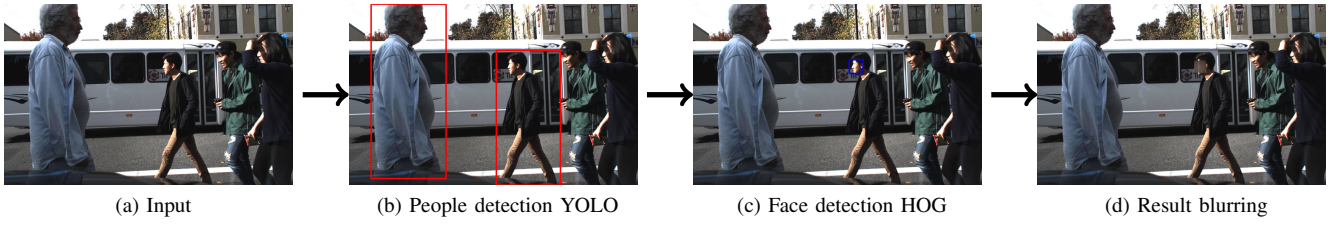
Technological advancements are transforming end devices from mere data consumers into active data producers. This shift encompasses consumer applications such as autonomous driving, where vehicle safety is enhanced through cooperative perception mechanisms [1]–[3], as well as large-scale industrial applications within the Industrial Internet of Things (IIoT). In the IIoT, for example, vast quantities of sensor data are processed to predict component failures [3], [4]. Additionally, the realm of AI technologies is witnessing the development of numerous innovative methods and solutions [5]. Traditionally confined to large data centers, there is a shift towards making AI systems more resource-efficient and bringing them closer to the end-user, which edge computing can facilitate [6], [7]. Edge computing involves deploying micro data centers closer to clients than traditional cloud data centers for latency-critical applications [4], [8], [9]. However, these micro data centers often possess less computing power and elasticity compared to traditional cloud data centers [9], [10]. Consequently, the likelihood of exceeding the capacity at an edge location, such as during nearby sports events, is higher due to limited resources.

Edge computing enables the execution of mission-critical applications directly at the data generation points by leveraging its geographical proximity to end-users [4], [8], [9]. This approach facilitates the deployment of mission-critical AI applications in sectors like IIoT and Vehicle-to-Everything (V2X) communications. By "mission-critical applications", we refer to those in which a computation failure could significantly affect the entire system [11]. A failure implies that a component of the mission-critical application does not respond or only responds with delays, thus demanding stringent requirements such as maximum execution times [11]. When mission-critical applications face stringent deadlines, redirecting excess load to larger cloud data centers or other edge locations may undermine the purpose of edge computing due to increased latency. In such scenarios, clients must either accept delayed responses or adjust the resource demands of an application. To adapt resource demands at runtime, we make use of approximate computing techniques. The concept of approximate computing assumes that many application types can tolerate controlled errors [12]–[14].

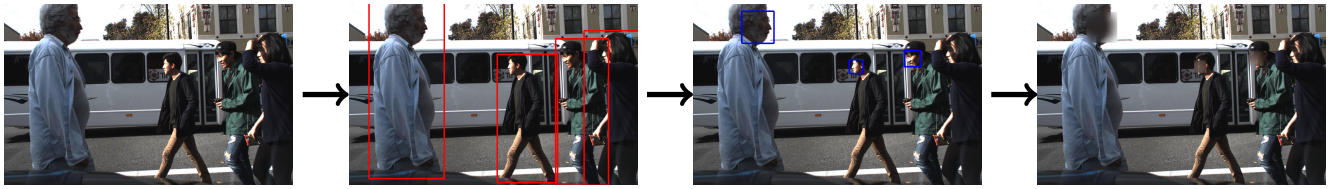
Ensuring that AI applications meet strict runtime constraints remains a significant challenge. Existing solutions utilize approximation techniques tailored to specific AI application types, such as tensor-based, without considering system-wide architecture [19], [20], which is essential for establishing a dynamic, mission-critical AI service ecosystem. Some approaches have investigated offloading entire applications to edge environments, optimizing for execution time or energy consumption [21]–[23]. Still, there remains a gap in executing AI applications under changing conditions essential for mission-critical AI services at the edge.

This paper introduces MARQ, the first framework, to our knowledge, that supports the engineering and execution of AI-based service chains capable of runtime adaptation, multi-objective optimization, and adherence to constraints such as maximum execution time or energy limits. One of the key metrics MARQ uses to evaluate computational outcomes is Quality of Result (QoR). MARQ facilitates dynamic adaptation when applications have strict execution requirements, such as deadlines. This involves the selection of different subtasks and adjusting execution parameters within the microservice

1558-1225/25/$31.00 ©2025 IEEE
DOI 10.1109/ICSE55347.2025.00182



(a) Input (b) People detection YOLO (c) Face detection HOG (d) Result blurring



(e) Input (f) People detection RCNN (g) Face detection CNN (h) Result blurring

Fig. 1: Example of an AI service chain: From input image to blurred faces, using different approximations (input data from the Roboflow dataset [15], person recognition from OpenMMLab [16], face detection from Face Recognition API [17], and bilateral blurring from ImageMagick [18]).

chain to balance multiple criteria. The goal is to achieve trade-offs among critical factors like QoR, energy use, and timing constraints. By using multi-criteria optimization, MARQ flexibly adapts to changing conditions, such as latency shifts or resource fluctuations. It ensures predefined constraints are met while optimizing performance and result quality, making it suitable for mission-critical applications.

After we have presented a motivating example for our work and given a definition for QoR in (Section II), we

- present the design of MARQ, including its overall architecture and components (Section III);
- introduce a comprehensive graph-based model to capture mission-critical AI service chains, including the different execution attributes and the QoR (Section IV), and the algorithm used to dynamically select the appropriate service chain at runtime (Section V);
- build a system prototype for MARQ and evaluate it with synthetic and real-world workload experiments (Section VI). Overall, our evaluations demonstrate a significant acceleration in decision-making, up to 100× faster than existing methods, while maintaining a low probability of deadline breaches and improving average QoR by over 10%.

Section VIII summarizes related work. Section IX draws final conclusions.

II. Motivating Example

A fundamental concept of our work is the use of QoR to quantify the arbitrary applications' result quality and to balance it with other parameters such as time or energy. Previous publications have used the concept of QoR [21], [22], [24], [25]. The QoR is determined either in discrete categories (e.g., 1 to 5) [25] or based on different metrics, such as the F1 score [21]. However, the exact methodology for determining the QoR for a broader range of applications remains unclear. We define QoR as follows:

*Definition 1 (**Quality of Result**): QoR is a measurable value that quantifies the result of a task, specified as a percentage with a domain-specific expression.*

For example, in an AI-based object recognition application, the quantification can be the average accuracy [26]. For Big Data applications, the quantification can be based on the relative error [27]–[29].

Using the example of recognizing people and faces, it becomes clear that processing can be performed using different approaches. Fig. 1 illustrates an image processing workflow. In this example, we present two different executions of the same application, both aiming for the same goal—ensuring data privacy compliance in vehicular driving functions—but utilizing different implementations for individual tasks. Specifically, for this example, we use an implementation of You Only Look Once (YOLO) [30] (see Fig. 1b) with Region-based Convolutional Neural Network (R-CNN) [31] (see Fig. 1f) as an alternative that can be used for adaption. For facial recognition of identified persons, we employ Convolutional Neural Network (CNN) (see Fig. 1g) as one method and Histograms of Oriented Gradients (HOG) [32] (see Fig. 1c) as an alternative.

As Fig. 1 shows, these algorithmic alternatives offer different result qualities with identical input data, which leads to different execution times. Each algorithm is encapsulated as a microservice and enables the creation of a dynamic microservice chain by selecting a sequence of microservices at runtime. While the first microservice chain, shown in Fig. 1a to Fig. 1d, performs the same task as the microservice chain shown in Fig. 1e to Fig. 1h, the first chain has a lower recognition rate. The recognition rate of the chain can be calculated as the multiplication of each stage [33]. Specifically, the first chain recognizes 50% of the objects at the beginning

and 50% of the objects in the second step, resulting in an overall accuracy of 25%. Whereas the second chain recognizes 100% of the people in the first step and 75% of the faces in the second step, resulting in an overall accuracy of 75%. Consequently, the second chain has a higher QoR.

Building on this knowledge, this paper explores the use of approximation techniques within algorithms to balance various conditions of an application so that its constraints can be consistently met. Constraints can be provided before the execution is started, and MARQ does not conceptually limit them. In the current implementation of MARQ, however, we focus on using the maximum execution time, maximum energy consumption, and a minimum value for QoR as constraints. In the previous example, QoR can be reduced to meet a deadline and a maximum energy threshold. Additionally, approximate computing allows for the use of specialized hardware, impacting execution time, energy consumption, and financial costs and enabling trade-offs between execution time and energy consumption [12], [24]. Our approach extends the concept of approximate computing by integrating QoR as a core consideration. We propose that different implementations of the same task can be used to meet execution goals—such as time, energy, *and* QoR—in line with specific constraints. By dynamically adapting the microservice chain and thus the resulting QoR, we can mitigate negative effects from external factors like latency spikes or limited hardware availability, which might otherwise risk violating critical constraints in mission-critical applications.

QoR is intricately connected to Quality of Service (QoS) and Quality of Experience (QoE). Unlike QoS, which in our case represents the measurable value of, i.e., latency and bandwidth and is therefore infrastructure-centric, QoR is the calculated quality of an outcome of an application and its parts. Trading the QoR for the QoS enables improvements in critical QoS aspects such as reliability and performance. If the latency time is of crucial importance, the transmission time can be reduced by relaxing the QoR requirements or reducing the amount of data generated, thus meeting the QoS goals. In user-centric scenarios, QoR has a direct impact on QoE. Approaches include the targeted reduction of the frame rate [34] or targeted sampling in stream processing tasks [28], [35], [36] to achieve a balance between functionality and user experience.

III. System Architecture

The dynamic selection of subtasks at runtime, such as the selection of a person recognition algorithm (as in the presented example of image processing, see Fig. 1), requires not only the modeling of the application and selection of subtasks (described in Section IV) but also a system architecture that enables the control of microservice chains at runtime.

The architecture consists of two main components: the QoR Manager and storage. The QoR Manager controls the microservice chains to ensure that application or user requirements are met. Figure 2 shows the architecture for an edge environment where different processing units are connected.

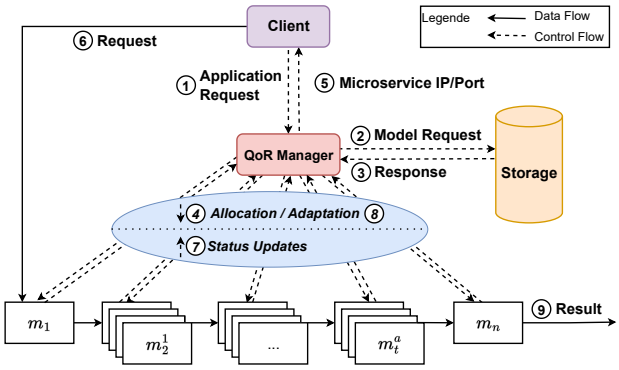


Fig. 2: MARQ architecture, including data and control flow.

The figure outlines the control and data flow from a request to the execution and customization of the microservice chain.

The QoR Manager consists of several parts: a decision logic determines the best path for the requested execution (see Section V), the resource manager keeps information about the used and free resources, and the application manager monitors the status of the requested applications so that adaptations can be made if necessary. When new servers are added to the edge, they register their resources with the QoR Manager, which can then manage multiple servers and applications simultaneously. The second main component of the architecture is the storage that stores the application graphs (described in Section IV), which serve as the basis for the runtime execution of the microservice chains.

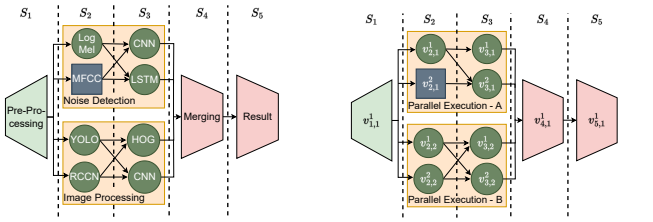
Execution flow: The request execution flow starts with a client sending a request to execute an application to the QoR Manager, including the application ID for unique identification and the constraints that must be met by the execution (1). In the example of the microservice chain from Section II, we use a maximum runtime as a constraint to be able to compare the various QoR results. The QoR Manager loads the associated application graph that was created in a so-called offline phase (Section IV). If the graph is not yet available in the memory of the QoR Manager, it is requested from the storage (2 + 3). Based on the application graph, the best execution path is determined (see Section V), fulfilling the specified constraints and balancing the various execution parameters without favoring or disfavoring individual parameters. At the same time, the QoR Manager allocates the resources for execution at the edge (4). The allocation is initially simple, and the microservices described in the graph are reserved. To start the actual processing, the QoR Manager returns the endpoint of the first microservice, consisting of IP address and port, to the client (5). The processing chain begins as soon as the client has sent the request (6) to the first microservice (m_1). The microservice encapsulates the AI service and provides a management component in addition to communication. This management component primarily monitors the core parameters that are necessary for the decision on adaptations, such as the latencies to subsequent microservices, knowing the chain's

next steps. In this way, the microservice can forward the results of an AI service directly to the next microservice in the chain without having to route the data flow via the QoR Manager. In addition, the QoR Manager receives continuous updates on the current workload, latencies, and other important metrics from both the microservices still being processed and currently processing (7). Based on this information, the QoR Manager adapts the current microservice chain from m_t^a to m_n at runtime if necessary by making changes such as selecting a different approximation type (8). The management components of the microservices then receive the information on the current following services in the form of IP address and port. Once all steps have been completed, the result is returned to the originally intended recipient (9).

Cost model for runtime decision: To decide at runtime whether an adaptation of the microservice chain is necessary, the QoR Manager must know the expected costs for the execution of each node $v \in V$ and each edge $e \in E$ in the application graph. An edge e represents a communication interface from one node v to another node v'. The QoR Manager knows the cost vector C, which represents the cost C_v of each node and the cost C_e of each edge, i.e., $C \in C_v \mid \forall v \in V \cup C_e \mid \forall e \in E$. To determine the initial value of the cost vector C, we use historical data obtained from similar hardware configurations. The approach is based on measurements of the same task on comparable systems, where the costs were averaged over a data set. In cases where historical data is not available, methods from cloud and grid computing [37], [38] or specific prediction models for deep learning models [39] could be used. However, our solution is sufficient to demonstrate the functionality of MARQ.

Dynamic adaptation and monitoring: Static, historical execution costs alone are insufficient to react to dynamic runtime conditions. Therefore, each microservice has a monitoring component that monitors the communication to its subsequent microservice. For example, the transmission cost C_e, representing the communication of an edge e, is transmitted from node v to the QoR Manager. Currently, we use a customized solution that measures latencies, resource availability, and QoR. The pre-execution value applies if the QoR cannot be determined at runtime. The monitoring method could be extended using systems such as OpenTelemetry [40] to support more complex scenarios. MARQ's decision process (see Section V) can consider any numerically measurable value to continuously update the costs in the graph and adjust the microservice chain in real-time.

Load avoidance and scalability of the QoR Manager: The QoR Manager processes multiple client requests in parallel and controls microservice chains at the edge. Our evaluation (Section VI) shows that our method is significantly faster than existing solutions, enabling more requests with the same resources. However, overloads can cause bottlenecks. To address this, we propose two options: First, multiple QoR Managers can be used, with each resource assigned to only one manager. Extending this with load balancing methods [41], [42] is possible but beyond the scope of this work. Second, the filter



(a) Motivating Example with parallel task extensions.

(b) Graph model with approximations and parallel paths.

Fig. 3: Generic graph model (following [21]) and privacy functionality from Section II with audio data processing.

function reduces the QoR Manager's load by sending updates only when monitored values exceed a defined threshold (Eq. (1)), where α represents acceptable deviations from the expected value c_i, and $x_i \in X$ is the current measured value.

$$\exists i \in \{0, ..., |C|\} | (x_i > (1 + \alpha_i) \cdot c_i) \\ \vee (x_i < (1 - \alpha_i) \cdot c_i) \tag{1}$$

IV. The Model

In addition to AI microservice chains such as image processing, shown as an example in Section II, many other use cases represent distributed, mission-critical AI services. Examples such as sensor fusion, advanced driving functions, remote control for autonomous driving [43], or predictive maintenance include more complex AI-based microservice chains. As already indicated in Section III in the form of costs, these types of applications have special characteristics that distinguish them from applications on a single computing unit. These characteristics include network properties, hardware diversity, connection types, and resource scarcity. We present a model designed to satisfy multi-criteria application-specific requirements such as time constraints, maximum energy consumption, and price constraints. We interpret these requirements as constraints that must be met and lead to a dynamic adaptation of the QoR for each subtask within the application, with the aim of being able to fulfill the requirements from the perspective of the overall application.

A. Model for Dynamic QoR Adjustments

The dynamic adaptation of QoR at runtime requires a model that has knowledge of the different parts of an application and can establish connections with the underlying system architecture, e.g., to estimate resource requirements and react appropriately to changes. In this context, we define a "sequence" as a chain of different microservices, each of which fulfills a specific purpose, is executed in sequence, and contributes to an overarching application goal. In addition, an application can consist of several parallel sequences. An example of the execution of parallel sequences is a web search in which the same search string is used to search for results in different formats (e.g., text, image, and video) on different AI

services simultaneously and made available to the requester as one result.

Building upon the modeling approach of Pandey et al. [21], called MobiDiC, which showed the advantage of using approximations for single-computer applications, our model addresses distributed application processing and enables more precise QoR control in response to external factors. Unlike existing approaches [21], [22], our model does not limit itself to a single factor, such as time. Our main challenges are dynamic adaptation at runtime, addressing multi-criteria requirements, and enabling parallelism in heterogeneous systems.

We use the structure of a Directed Acyclic Graph (DAG) to represent the different ways of executing the sequences. Figure 3a illustrates how the graph structure is built using the motivating example. The different sub-steps are represented as nodes, and the arrows symbolize the possible sequences of the application. We supplement this structure with parallel tasks as they can occur in real applications. Figure 3b shows the generic representation of an application as DAG, defined as $G(V, E)$. As described in Section III, an application comprises various nodes $v \in V$, each representing a microservice subtask to be executed. Edges between the nodes are represented as $e \in E$. To enable different types of connection (e.g., Bluetooth, cable, or WiFi) from one node v to another node v', there can be more than one edge between two nodes. A complete sequence is a sequence of nodes and edges described as a path $p = (v_1, e_1, v_2, e_2, \ldots, v_n, e_n, v_{n+1})$. Since applications can consist of parallel sequences, the model must handle this type of application. In Fig. 3, the parallel parts are labeled A and B. Modeling the parallel parts allows us to decide at runtime which sequence is to be taken by the parallel part.

Within a sequence, we assume there are different ways of varying approximations. In particular, we distinguish between alternative subtasks (as seen in the image processing example in Section II), i.e., subtasks that have the same goal but use different algorithms to achieve this goal, and different parameters that a subtask can receive as input. The parameters can be different input data but also configurations, such as in the example of TensorFlow applications, where the number of generated tensors can vary[19]. Figure 3 shows these alternative subtasks as circles, while squares visualize parameter variations. Trapezoids, on the other hand, describe application segments that do not allow alternative approaches. Examples of this type of subtask are I/O accesses or user queries.

The different edges and nodes with their respective costs must be clearly identifiable to enable a sequence of tasks in the graph. A concrete version of a task can be identified by $v^a_{s,t}$. Here, s indicates the stage in the sequence, t is the subtask to be completed, and a is the approximation available. To uniquely identify the transmission cost between two nodes, a particular edge can be identified by $e^a_{s,t,h}$, where h is an index that allows different connection types between the nodes to be considered.

As described in Section III, the graph G provides diverse cost factors for determining the optimal path through the graph. Each node v has a cost vector C_v that represents the task's processing costs, and each edge e has a cost vector C_e for modeling the communication.

B. Offline Generation of Application Graph

The underlying model enables the description of the overarching application, based on AI services, with its sequences, subtasks, approximations, and costs. The generation of the application graph with its individual subtasks and approximations takes place before the first execution. It is based on expert knowledge and is outside this work's scope. The QoR of the respective function can then be calculated using mathematical methods (see Section II).

Based on the specified graph, MARQ identifies the various parallel subgraphs, described in Fig. 3 with A and B. MARQ generates the subgraphs in the offline phase, as their structure does not change at runtime. By splitting them up, we enable accelerated processing and individual consideration of the costs to be complied with for each subgraph. In addition, the individual consideration of the subgraphs makes it possible to consider the choice of paths based on the expected costs of the other subgraphs. Decisions on which paths to choose are made dynamically in the online phase based on the costs and constraints of the application.

As already described, we use the index t of the different subtasks to identify the subgraphs. To generate the subgraphs, we traverse each vertex v within each stage s and approximation indices a that approximate the subtask t and the subtasks approximated by possible successor nodes. This results in several sequential permutations of successive applications. Subgraphs are then derived from the resulting permutations of subtask indices by including all corresponding approximations of the respective subtasks in the subgraph. The exact determination of the subgraphs is described in Eq. (2). G' is a sequential subgraph of G, assuming that G is a graph with parallel tasks and a set of endpoints V_{end}.

$$\begin{aligned} G' \subseteq G \mid (\forall v^a_{s,t}, v^{a'}_{s',t'} \in G' \mid s = s' \Rightarrow t = t') \\ \wedge [\forall s \in S \mid (\exists v^a_{s,t}, v^{a'}_{s',t'} \in G' \mid \\ (s \neq s' \wedge (v^a_{s,t}, v^{a'}_{s',t'}) \in E) \vee v^a_{s,t} \in V_{end})] \end{aligned} \tag{2}$$

Finally, as mentioned above, the costs for the nodes and edges are determined using the historical data and saved with the subgraphs.

V. Online Management of QoR

The QoR management process can be divided into two distinct phases: the "offline" phase and the runtime execution, also called the "online" phase. As detailed in Section IV-B, the offline phase involves one-time tasks that would not benefit from online execution. These tasks encompass activities like graph generation and the identification of subgraphs within the graph. In contrast, the online execution phase entails real-time decision-making based on the system's current state, aiming to satisfy user-specified conditions.

A. Comparing the Alternatives

To fulfill the conditions placed on the execution, we use two different approaches for the comparison of possible alternatives: the Technique for Order Preference by Similarity to Ideal Solution (TOPSIS) method [44] and the ε-constraint method [45]. These methods enable us to evaluate and select the most suitable solutions based on various criteria.

The TOPSIS method is a multi-criteria decision-making technique that assigns weights to different criteria to identify potential solutions. One notable advantage of this approach is its flexibility in allowing users to express their preferences by assigning weights to the criteria. However, it is limited because the solution proposed by TOPSIS may not always align with the user's specified conditions. This can occur when the user's conditions cannot be fully met due to inadequate weighting and the available data.

To be able to perform multi-criteria decisions based on TOPSIS, we first normalize the elements within the cost vectors C to prepare them for comparison. This normalization process is carried out as follows:

For each element c_{ij} in the cost matrix C we calculate the normalized value, c'_{ij}, using the equation:

$$c'_{ij} = \frac{c_{ij}}{\max_j(c_{ij})} \tag{3}$$

Here, j stands for the criterion under consideration (e.g., execution time, QoR, or power consumption), and i for the alternative to be evaluated. Parallel to the normalization, we calculate a vector with the maximum values and a vector with the minimum values of each criterion across all alternatives. This results in the cost vector for the ideal solution C^* and the cost vector for the anti-ideal solution C^-.

In the second step, weights can be assigned to each criterion to reflect their relative importance. This allows users to express their preferences and prioritize certain criteria over others. Applying the weights to the normalized costs, we obtain c''_{ij}, which is accomplished using the equation:

$$c''_{ij} = w_j \cdot c'_{ij} \tag{4}$$

In this case, w_j stands for the weight assigned to criterion j, and c'_{ij} is the normalized value obtained in the previous step.

In the TOPSIS method, two types of distances are calculated for each alternative: D_i^* and D_i^-. These distances measure how far each alternative is from the ideal solution and the anti-ideal solution, respectively. D_i^* is the Euclidean distance that measures how far an alternative is from the best possible scenario, where the alternative perfectly matches the ideal solution. It is calculated using the formula:

$$D_i^* = \sqrt{\sum_{j=1}^{n}(c''_{ij} - (C^*)_j * w_j)^2} \tag{5}$$

where n is the number of criteria. $(C^*)_j$ represents the normalized j-th criterion value of the ideal solution. Conversely, D_i^- measures how close each alternative is to the least desirable scenario or the worst possible values for each criterion. It is calculated similarly but using the values of the anti-ideal solution, C^-, as follows:

$$D_i^- = \sqrt{\sum_{j=1}^{n}(c''_{ij} - (C^-)_j * w_j)^2} \tag{6}$$

Finally, the relative closeness (R_i) of each alternative to the ideal solution is calculated by considering both D_i^* and D_i^-:

$$R_i = \frac{D_i^-}{D_i^* + D_i^-} \tag{7}$$

This measure helps in ranking the alternatives. A higher R_i value indicates that the alternative is closer to the ideal solution and further from the anti-ideal, making it a more desirable choice.

By calculating both D_i^* and D_i^-, the TOPSIS method provides a comprehensive way to evaluate alternatives not just by how close they are to the best possible outcome but also by how far they are from the worst outcome, ensuring a balanced view in multi-criteria decision-making. This dual consideration helps in making more robust and reliable decisions in complex scenarios.

Compared to the TOPSIS method, the ε-constraint method takes a different approach to simplifying the optimization problem. When using the ε-constraint method, all criteria except one are treated as constraints that can be predefined by the user rather than being subjects of direct optimization. This process effectively filters the pool of alternatives, isolating a subset of feasible options. Consequently, optimization efforts are then concentrated solely on one remaining parameter.

B. Pathfinding and Decision-Making at Runtime

To facilitate the dynamic determination of suitable paths within a graph, our methodology includes on-demand recalculations triggered by fluctuations in the cost metrics or the current status of the microservice chain's execution progress. This process entails confronting two principal challenges: (1) the computation of current costs associated with diverse paths and (2) the identification and selection of the most fitting paths based on these computed costs and specific application conditions.

The real-time recalculation of costs (1) and paths ensures responsiveness to emergent changes. As explained in Section III, such changes may stem from the progress in the computation of the microservice chain, changes in network conditions, or availability of resources. Given that cost vectors may encompass diverse cost types, including time and QoR, we deploy an aggregation function designated as $f(C_i, C_j, R)$. This function merges the cost vectors C_i and C_j by predetermined rules R, encompassing basic mathematical operations. The ability to specify these rules allows our method to combine different cost types for nodes and edges.

To determine the most suitable path between two vertices and to be able to select a path (2), we use a Dijkstra implementation similar to MobiDiC. Due to the changing costs at runtime, calculating the best paths and choosing the path must

Algorithm 1: Dijkstra-Pareto-optimal Pathfinding

Input : $G(V, E)$, f, R, v_{start}
Output : P_{pareto}

$P_{pareto} \leftarrow \emptyset$
$Q \leftarrow \{(v_{start}, 0)\}$ /* Initialize a priority queue, with 0 costs */
while Q *is not empty* **do**
  $(v, C_v) \leftarrow Q$.dequeueWithLowestCost()
  forall *neighbors* u *of* v*; with* $u, v \in G$ **do**
    $C_{tmp} \leftarrow f(C_e, C_u, R)$ /* Merge edge cost C_e with node cost C_u */
    $C_p \leftarrow f(C_v, C_{tmp}, R)$ /* Cost for complete path */
    if *!ParetoDominated*(C_p, P_{pareto}) **then**
      $Q \leftarrow Q \cup (u, C_p)$ /* Enqueue neighbor with updated cost */
      $P_{pareto} \leftarrow P_{pareto} \cup (u, C_p)$
    end
  end
end
return P_{pareto}
Function ParetoDominated(C, P)**:**
  forall $(u', C_{u'}) \in P$ **do**
    if $C_{u'}$ *dominates* C **then**
      return true
    else if C *dominates* $C_{u'}$ **then**
      Remove $(u', C_{u'})$ from P /* no longer considered */
  end
end

Algorithm 2: Pathfinding utilizing TOPSIS in Dijkstra

Input : $G(V, E)$, f, R, v_{start}
Output : p_{best}

$visited, C_{ideal}, C_{anti} \leftarrow \emptyset$
/* Initialize a list, with v_{start} as path and 0 cost */
$Q \leftarrow \{(v_{start}, 0)\}$
while Q *not empty* **do**
  /* Determine current ideal and anti-ideal solutions */
  $C_{ideal}, C_{anti} \leftarrow$ Solutions(Q)
  $(v, C_v) \leftarrow$ FindHighestProximity(Q, C_{ideal}, C_{anti})
  Remove (v, C_v) from Q
  $visited \leftarrow visited \cup \{(v, C_v)\}$
  forall *neighbors* u *of* v*; with* $u, v \in G$ **do**
    if $u \in visited$ **then**
      continue
    end
    $C_{tmp} \leftarrow f(C_e, C_u, R)$
    $C_p \leftarrow f(C_v, C_{tmp}, R)$
    $(u, C_{best}) \leftarrow Q.get(u)$ /* NIL, iff u not in Q */
    if $(u, C_{best}) \neq$ *NIL* **then**
      $C_{ideal}, C_{anti} \leftarrow$ Solutions$(Q \cup (u, C_p))$
      $p \leftarrow$ FindHighestProximity$(\{(u, C_p), (u, C_{best})\}, C_{ideal}, C_{anti})$
      if $p == (u, C_{best})$ **then**
        continue
      end
      Remove (u, C_{best}) from Q
    end
    $Q \leftarrow Q \cup \{(u, C_p)\}$
  end
  if *neighbors of* v *equals* $\emptyset$ **then**
    $p_{best} \leftarrow visited.get(v)$
  end
end
return p_{best}

be done repeatedly and online. Therefore, we integrate the calculation of costs into the pathfinding process. Furthermore, our implementation of the Dijkstra algorithm finds the shortest path by considering several parameters. To determine which path to a node is the most favorable, either an implementation based on the Pareto optimum (ε-constraint method) or the direct use of the decision maker (TOPSIS) is used.

As an integral part, we enhance the ε-constraint method by using an algorithm to generate Pareto-optimal paths (P_{pareto}) and the associated costs (C_p) at runtime. Algorithm 1 shows this approach. The paths in P_{pareto} represent the best trade-offs between multiple costs (C) and ensure that no other path can achieve better values in all costs simultaneously [46]. Our Pareto comparator is central in identifying optimal paths by evaluating the relations between two cost vectors C_i and C_j and determining whether C_i is dominated by, dominates, or is equal to C_j.

To fully execute the ε-constraint method, we proceed in the following order: First, we generate the Pareto-optimal paths $P_{pareto} \subseteq P$. Second, these paths are filtered according to whether they fulfill user-defined constraints called $l \in L$ (as described in Eq. (8)). In this case, Δ contains only the Pareto-optimal paths that fulfill the given constraints. Next, we search for the paths where the QoR is highest (as in Eq. (9)). Finally, from the set of paths that are Pareto-optimal, fulfill the constraints, and have the highest QoR (represented by Ω), we select the path with the lowest cost (as in Eq. (10)).

$$\Delta = \{p \in P_{pareto} | c_p \leq l_i\} \tag{8}$$

$$\Omega = \{p \in \Delta | Q_p = max(Q_q | q \in \Delta)\} \tag{9}$$

$$p* = argmin(C_p | p \in \Omega) \tag{10}$$

Unlike the ε-constraint method, where all possible paths must be known to decide which path fulfills the application constraints and is the best with respect to one parameter, TOPSIS can determine the approximately best path during the execution of Dijkstra. Our implementation of TOPSIS tends to rank the approximate Pareto-optimal paths highest, which streamlines the process by eliminating the need for a separate treatment of the Pareto-optimal paths and a subsequent TOPSIS application. Additionally, our implementation works with a relative weighting of costs/benefits and a relative ranking of the known paths. Algorithm 2 shows our approach. It should be emphasized that our suggested Dijkstra adjustment stores paths as costs to a node and not just the predecessor node.

TABLE I: Hardware configuration for tests

Name	Description	CPUs	RAM
Edge server 1	QoR Manager	4	8 GB
Edge server 2 - 4	Microservices	16	64 GB
Edge node	Client & JMH	24	32 GB

TABLE II: Test graph configuration

Name	Microservices	Stages	Alternatives	Subgraphs
small	10	5	2	2
normal	19	6	2	5
big	40	8	7	6
huge	120	14	8	12

Since we want to achieve a multi-criteria view of the costs, more than one path may be applicable. If we come to a node already known, we have to estimate which path is better.

The weights used in TOPSIS offer the possibility of significant influence on the decision-making process and express the trade-offs between criteria more effectively. However, a drawback of this approach is that the selected alternatives may not satisfy the user's constraints. To overcome this problem, the calculation can be adapted so that TOPSIS is executed iteratively with adjusted weights until a valid path is found.

VI. Evaluation

Mission-critical AI service chains should be able to react flexibly to changing external factors or (emerging) failure of a microservice at runtime. This results in various research questions for our evaluation, which we want to answer to be able to draw comparisons between MobiDiC, the state-of-the-art, and the methods provided by MARQ (TOPSIS and ε-constraint method), which we believe are essential. In the first step, we want to measure the decision-making process. To this end, we have divided the performance measurements into two research questions (RQ):

RQ1 - Single Node: Which method is faster in finding the best alternative on a stage s with respect to the task t, assuming that different criteria have to be considered and different possible alternatives exist?

RQ2 - Pathfinding: Using the Dijkstra algorithms described in Section V-B to find the best path through the graph, which method can make the fastest decision considering the different parallel and sequential parts?

In addition to the pure performance measurements, we also want to measure the influence on the QoR and deadline in the case of changing conditions at runtime. Therefore, we want to consider which method produces the best overall result under the existing conditions. To answer this, we conduct the following RQ:

RQ3 - Dynamics: What influence do the different decision methods have on the final QoR and adherence to the specified application conditions under changing circumstances due to fluctuating latencies and execution times?

A. Test Setup

We used four virtual machines (edge servers 1 - 4) on three separate servers in the same server room and one ThinkStation (edge node) for our experiments. Detailed hardware specifications for these servers can be found in Table I. The microservices relevant to our experiments were executed in Docker containers on their respective virtual machines and comprise the components described in Section III. Edge server 1 performs the role of the QoR Manager, as described in Section III, while edge servers 2 to 4 are responsible for operating the individual microservices. The edge node acts as a client and is used for performance measurements.

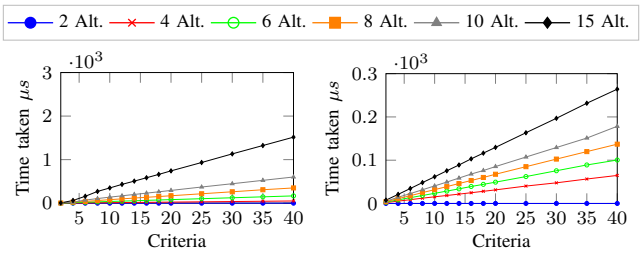
To evaluate the impact of traffic between different microservices, we use Pumba [47], a chaos testing tool that uses the Linux traffic control tool tc-netem [48] for traffic manipulation. This setup can simulate various network conditions, including fluctuating latencies. We utilized Java Microbenchmark Harness (JMH) [49] to measure the time required for the decision-making process accurately. The configurations used for each tool can be found in the respective sections.

In our evaluations, we perform measurements on different graph structures with varying levels of complexity. These graph structures correspond to distinct microservice chains and are detailed in Table II. The data in the table shows how many microservices each graph contains in total, how many stages the microservice chain is divided into, the maximum number of alternatives available per stage, and how many subgraphs the graph contains. To facilitate the reuse of these graph structures, but also of MARQ, we have made these graphs, including graphs with random network and load manipulations, available as base graphs as well as MARQ on Zenodo [50]. Our experimental combinations are informed by Luo et al. [51], who examined microservice call structures in an Alibaba data center, offering a comprehensive overview.

B. RQ1 - Single Node

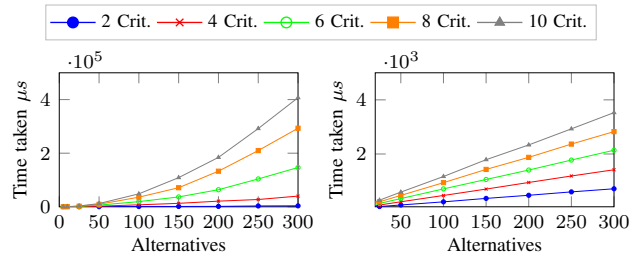
Efficient and rapid selection of suitable alternatives at runtime is crucial, especially if the graph structure offers many alternatives and several criteria must be considered simultaneously. To achieve this, we use TOPSIS as described in Section V. In our experiments, we compare the execution time of TOPSIS with finding the Pareto-optimal solution since both the ε-constraint method and MobiDiC are based on Pareto-optimal solutions. Even though MobiDiC calculates the Pareto-optimal solution once in the proposed implementation, we recalculate the solution as soon as the costs change to react to this change.

To measure the speed of the different methods, we varied the number of criteria and alternatives taken into account in two experiments. The first experiment reflects the microservice structures described in [51] and considers a higher number of criteria. The second experiment, the long-term experiment, was designed to show the progression of the execution times of the individual methods. Both experiments were conducted with JMH. In all cases, JMH was initialized with three warm-up runs for each test run to ensure accurate measurements. Measurements were recorded over a period of 10 seconds for each iteration.



(a) Average Time Pareto (b) Average Time TOPSIS

Fig. 4: Average time required to compare up to 15 alternatives (alt.) per stage and up to 40 criteria.



(a) Average Time Pareto (b) Average Time TOPSIS

Fig. 5: Average time required to compare up to 300 alternatives per stage and up to 10 criteria.

In the first experiment, the execution time of the two methods was compared with up to 40 criteria and 15 alternative ones for a task t. Each permutation of the parameters was repeated 10× over four forks as a test run, i.e., a total of 40× per permutation. In the long-term experiment, we varied the possible alternatives from 2 to 300 in steps 2, 5, 10, 25, 50, 100, 150, 200, 250, and 300. We also varied the number of criteria considered simultaneously from 2 to 10. Each permutation of the parameters was repeated as a test run 100× over four forks, i.e., a total of 400× per permutation, to obtain reliable and statistically significant results.

The use of TOPSIS leads to a remarkable improvement in the speed of the decision-making process. Figure 4a shows the average time needed to find Pareto-optimal solutions for up to 40 criteria and 15 alternatives. In contrast, Fig. 4b shows the corresponding results for our proposed TOPSIS-based approach. It can already be seen here that TOPSIS is 10× faster for the same task. Looking at the results of the long-term experiment, which can be seen in Fig. 5, it becomes clear that our proposed solution speeds up the decision-making process 100-fold. It should be noted that the scales of the diagrams are different to show the shape of the curve.

C. RQ2 - Pathfinding

Furthermore, our evaluation extends beyond selecting individual nodes and encompasses decision-making for the entire graph, as discussed in Section V. These decisions are made at runtime, taking into account external factors. To investigate this aspect, we conducted experiments with the graphs described in Table II. The increased number of alternatives within these graphs potentially leads to a higher count of Pareto-optimal paths.

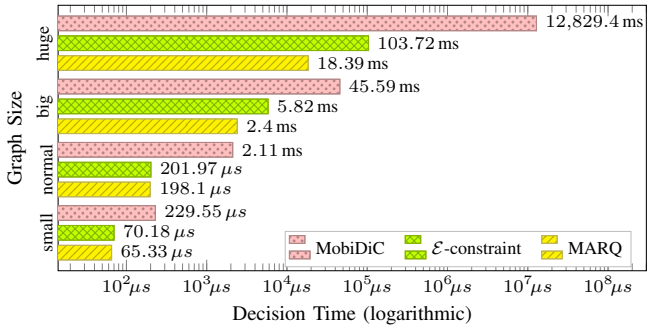


Fig. 6: Average time for finding paths.

To assess the execution time of pathfinding for each graph, we utilized JMH. The experiment configuration consisted of 40 iterations, three warm-up runs, and four forks. The costs C for the respective vertices and edges were randomly generated per run. Each method was executed once with each cost configuration.

Figure 6 presents the average time required for pathfinding on the entire graph, with a logarithmic scale for better visualization. Remarkably, our proposed approach, MARQ, consistently and efficiently identifies the paths to be utilized, even as the number of alternatives in the overall graph increases. In contrast, due to the individual Pareto-optimal pathfinding process, the ε-constraint method experiences slower processing times. As expected, the processing time increases with the growing number of variants that need to be considered for pathfinding. This particularly impacts MobiDiC's decision process. Additionally, since the ε-constraint method also requires computing the Pareto-optimal paths, a significant increase in processing time is observed with an increasing number of alternatives in this method.

The significant difference between the ε-constraint method and the MobiDiC method is worth noting, even though both methods operate on the Pareto-optimal solution and must initially compute it. This difference arises from how MobiDiC handles subgraphs. Specifically, it pertains to the data structure handling of the graphs. In contrast to MobiDiC, we have implemented an iterative approach, where we first create the possible subgraphs and the resulting permutations. However, MobiDiC treats subgraphs as "new" graphs, employing a recursive approach to iterate the entire graph.

D. RQ3 - Dynamics

After demonstrating in Section VI-C that decision-making with TOPSIS and the ε-constraint method enables rapid pathfinding, we now analyze the impact of dynamic runtime adaptation on the average QoR. Load tests are conducted using the graphs from Table II. As highlighted in the pathfinding evaluation (Section VI-C), MobiDiC takes approximately 12.8 s (see Fig. 6) to find the optimal path for the large graph. This calculation must be repeated for each dynamic

TABLE III: Measurements of the effects on the average QoR due to dynamic microservice-chain adaptations.

		TOPSIS					***ε-constraint***					***MobiDiC***				
G	t_{max}	Ø Time	Ø QoR	Success	Failed	No path	Ø Time	Ø QoR	Success	Failed	No path	Ø Time	Ø QoR	Success	Failed	No path
Small	5 s	4.803 s	100%	100%	0%	0%	4.804 s	100%	100%	0%	0%	4.266 s	87%	100%	0%	0%
	5.5 s	4.803 s	100%	100%	0%	0%	4.802 s	100%	100%	0%	0%	4.274 s	87%	100%	0%	0%
	6 s	4.802 s	100%	100%	0%	0%	4.808 s	100%	100%	0%	0%	4.301 s	87%	100%	0%	0%
Normal	4.5 s	4.301 s	100%	39%	0%	61%	4.308 s	100%	46%	7%	47%	4.471 s	93%	19%	34%	47%
	5 s	4.305 s	100%	100%	0%	0%	4.306 s	100%	100%	0%	0%	4.508 s	94%	100%	0%	0%
	5.5 s	4.307 s	100%	100%	0%	0%	4.305 s	100%	100%	0%	0%	4.506 s	94%	100%	0%	0%
Big	5.5 s	4.852 s	56%	92%	0%	8%	5.187 s	84%	42%	53%	0%	4.892 s	44%	99%	1%	0%
	6 s	4.853 s	58%	100%	0%	0%	5.323 s	98%	87%	13%	0%	4.922 s	43%	99%	1%	0%
	6.5 s	4.856 s	54%	100%	0%	0%	5.331 s	100%	100%	0%	0%	4.914 s	43%	98%	2%	0%

change, such as latency shifts. However, the execution of this graph's microservice chain finishes in less than 12.8 s, indicating that execution completes before MobiDiC finalizes the pathfinding, even without recalculating for new conditions. Given that MobiDiC's pathfinding takes longer than the chain execution, a direct comparison is not feasible. To maintain clarity and relevance, we excluded the huge graph from RQ3.

Each graph was executed 100× with all methods. We changed the costs after each execution of the methods. The executions used the hardware configuration described in Table I, and the application simulations were executed in Docker containers that ran continuously.

To evaluate the robustness of our approach, we introduced random external influences into our sample application. Each microservice node was assigned a random percentage error of up to +/-10% of the initial execution time. The simulation considers that a node with a higher QoR takes longer to execute and consumes more energy. To simulate variations in latencies between nodes, we used Pumba. We started 60% as many Pumba instances as there were active nodes. Each Pumba instance randomly selected a node and introduced a random delay so that it was 100 ms +/- 100 ms. The distribution of these delays followed a Pareto distribution. For MobiDiC, we left the time as the optimization target; for the ε-constraint method, we optimized for the QoR after finding the Pareto optimal paths, and for TOPSIS, we used the weights of time - 100%, QoR - 60%, and energy and price 25% for optimization. We chose these weights to show that the multi-criteria optimization at runtime improves the QoR, taking into account the application conditions.

Table III shows measured results for the respective methods and graphs. We have limited ourselves to the visualization of the average values (symbolized by a Ø) for time and QoR, as well as the number of successful executions, cases in which the calculation did not finish in time due to too high fluctuations, and the cases in which no valid path could be found.

The results show that TOPSIS always ensured execution within the constraints when a path for execution was available within the deadline, increasing the QoR compared to MobiDiC. Due to the selected weights, it can be seen that the QoR of TOPSIS is average when executing the "big" graph. A different choice of weighting will influence this. Due to the optimization goal of the QoR in the case of the ε-constraint method, it is possible that the calculation cannot be executed within the deadline. However, the QoR is at least as good as TOPSIS and significantly better than MobiDiC.

VII. Discussion

This paper presents MARQ, a framework that supports decision-making in mission-critical AI microservice environments through optimized pathfinding and runtime adaptation. Our findings show that MARQ, utilizing TOPSIS and the ε-constraint method, efficiently meets demanding performance targets, even under fluctuating conditions, highlighting its potential in dynamic edge computing for balancing QoR, deadlines, and energy use.

A. Interpretation and Implications

MARQ's success in controlled settings demonstrates its ability to adapt AI service chains in real-time, balancing multiple objectives in resource-limited scenarios. This adaptability is crucial for mission-critical applications like autonomous driving and edge surveillance, where meeting constraints under changing conditions is essential. However, while our tests showed consistent performance, challenges beyond controlled settings may emerge in real-world applications.

B. Threats to Validity

Internal Validity: Weights assigned to individual parameters significantly impact decision quality—both positively and negatively, depending on graph complexity and edge conditions (Section V-B). Testing focused on specific parameter settings and decision algorithms (TOPSIS and ε-constraint method), implying that other configurations could influence MARQ's performance (Section VI-D).

External Validity: Although MARQ performed reliably in a controlled environment, real-world edge computing environments are inherently more complex. Issues such as unpredictable network fluctuations, packet loss, and resource contention may limit MARQ's ability to adapt in real-time. Testing MARQ under diverse real-world conditions will provide insights into its broader applicability, especially in large-scale, multi-node settings.

Construct Validity: The primary metrics for evaluating MARQ—QoR, execution time, and energy consumption—are critical but not exhaustive. Other non-functional aspects, such as reliability under severe network degradation, were not directly considered. Future evaluations that include these factors would provide a more comprehensive assessment of MARQ's capabilities.

Framework Limitations: The scalability of MARQ using multiple QoR Managers, as described in Section III, requires

further investigation and optimization. The current static allocation of resources to QoR Managers could be extended by a dynamic method. Load balancers [41], [42], in particular, can help here. In case of resource constraints, there should be a slight overprovision of resources in production environments to buffer unexpected events.

C. Future Directions

Future work should extend the MARQ tests to operating environments with different network and resource constraints. Adding redundancies [52], warm standby systems [53], and fallback mechanisms can improve reliability under extreme conditions where MARQ currently reaches its system limits, such as abrupt or excessive network degradation. Using multiple QoR Managers and decentralized control could improve scalability in multi-node environments.

Automating graph generation, currently expert-driven, is another key direction. Reducing bias and enhancing scalability could be achieved through Software Product Lines [54]–[56], Feature Models [57]–[59], and call graph analysis [60], [61].

D. Applicability of MARQ

We focused on object detection due to its broad applicability in domains like multi-task autonomous perception (e.g., identifying agents, their actions, and locations [62]), healthcare (e.g., anomaly detection in medical imaging [63]), and robot-based search-and-rescue operations [64]. Beyond object detection, MARQ applies to any AI service encapsulated in microservices, enabling dynamic runtime adaptations. Examples include tasks in audio processing (e.g., speech-to-text [65]).

VIII. Related Work

Approximation techniques. The use of approximations to achieve improved execution speeds or energy savings has been the subject of research efforts [12], [24], [66]. These concepts and approximation techniques have been explored at various levels of computing approximations [66]. Optimizing application execution times is crucial, especially when dealing with vast amounts of data. Approximation techniques are employed to obtain faster results that may not be fully precise but are deemed acceptable for the application's requirements [27], [28], [67], [68]. Similarly, in the context of AI/ML applications at the edge, the execution time must be managed within defined limits [19], [65]. Task offloading is another use case in which much research took place. In such scenarios, the adaptation of QoR becomes relevant [21], [22], [25].

Modeling of Quality of Result. The model proposed by Pandey et al. [21] inspires the current work. Pandey et al. utilize approximations to optimize application execution time. We extend this approach to consider distributed applications, focusing on time optimization and other aspects. As our approach supports microservice chains, various external factors can impact the execution time. Therefore, we dynamically react to adaptations, aiming to achieve the application's objectives while adhering to given constraints. QLRan [25] is a work that focuses on selecting the best node for executing a particular application. However, this approach may encounter limitations when dealing with microservice chains, where an application may span across multiple nodes. In contrast, our approach, MARQ, does not solely concentrate on node selection but aims to dynamically determine the optimal paths through the microservice chain, considering various factors. MobiQoR [22] introduces QoR as a novel optimization dimension for the decision-making process of task offloading. It addresses whether a task should be offloaded to a remote server or executed locally based on energy consumption and system efficiency considerations. MobiQoR proves beneficial for making decisions regarding task offloading, especially when energy efficiency or similar system-level factors are crucial. Caches (local storage) are employed in MobiQoR to enhance the results further.

Dynamic adaptation to changing conditions becomes essential in scenarios where tasks can benefit from not only being executed on a local device. MARQ offers a viable solution to such problems by providing a framework for dynamic adaptation of QoR based on various factors and constraints.

To the best of our knowledge, the targeted adaptation of QoR based on multiple factors for mission-critical microservice AI microservice chains is a novel contribution. Existing solutions often optimize for a single parameter, such as time, while neglecting other factors like result quality, resource utilization, or latency. In distributed systems, however, resource utilization, especially CPU utilization, has a significant influence on execution time in addition to latency [51].

IX. Conclusion

We introduce MARQ, a framework developed in response to the need for dynamic adaptation of AI service chains at the edge. This need arises from the challenges of executing mission-critical AI services that require balancing performance metrics such as QoR, execution time, and energy consumption. MARQ encapsulates these services into microservices and employs real-time multi-criteria decision-making. It utilizes two decision-making approaches: the ϵ-constraint method, which optimizes one criterion while achieving Pareto optimality, and TOPSIS, which identifies the most suitable execution path by considering multiple criteria simultaneously.

Our model extends the approaches by incorporating the ability to dynamically respond to external influences such as latency and parallel processing. This enhancement enables MARQ to perform pathfinding up to $100\times$ faster than current methods, achieving a significant improvement in QoR by over 10%, effectively meeting the stringent demands of mission-critical applications at the edge.

X. Acknowledgment

This work has been funded by the German Research Foundation (DFG) within the Collaborative Research Center (CRC) 1053 MAKI and by the German Federal Ministry of Education and Research (BMBF) 01IS17050.

References

[1] H. Bagheri, M. Noor-A-Rahim, Z. Liu, *et al.*, "5G NR-V2X: Toward Connected and Cooperative Autonomous Driving," *IEEE Communications Standards Magazine*, vol. 5, no. 1, pp. 48–54, 2021. DOI: 10.1109/MCOMSTD.001.2000069.

[2] Z. Wei, B. Li, R. Zhang, X. Cheng, and L. Yang, "Many-to-Many Task Offloading in Vehicular Fog Computing: A Multi-Agent Deep Reinforcement Learning Approach," *IEEE Trans. on Mobile Comput.*, vol. 23, no. 3, pp. 2107–2122, 2023. DOI: 10.1109/TMC.2023.3250495.

[3] M. Bennis, M. Debbah, and H. V. Poor, "Ultrareliable and Low-Latency Wireless Communication: Tail, Risk, and Scale," *Proceedings of the IEEE*, vol. 106, no. 10, pp. 1834–1853, 2018. DOI: 10.1109/JPROC.2018.2867029.

[4] Y. Chiang, Y. Zhang, H. Luo, *et al.*, "Management and Orchestration of Edge Computing for IoT: A Comprehensive Survey," *IEEE Internet Things J.*, vol. 10, no. 16, pp. 14 307–14 331, 2023. DOI: 10.1109/JIOT.2023.3245611.

[5] D. Katare, D. Perino, J. Nurmi, M. Warnier, M. Janssen, and A. Y. Ding, "A Survey on Approximate Edge AI for Energy Efficient Autonomous Driving Services," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 4, pp. 2714–2754, 2023. DOI: 10.1109/COMST.2023.3302474.

[6] R. Singh and S. S. Gill, "Edge AI: A survey," *Internet of Things and Cyber-Physical Systems*, vol. 3, pp. 71–92, 2023. DOI: 10.1016/j.iotcps.2023.02.004.

[7] S. Tuli, F. Mirhakimi, S. Pallewatta, *et al.*, "AI augmented Edge and Fog computing: Trends and challenges," *Journal of Network and Computer Applications*, vol. 216, p. 103 648, 2023. DOI: 10.1016/j.jnca.2023.103648.

[8] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017. DOI: 10.1109/MC.2017.9.

[9] T. Zheng, J. Wan, J. Zhang, C. Jiang, and G. Jia, "A Survey of Computation Offloading in Edge Computing," in *International Conference on Computer, Information and Telecommunication Systems*, ser. CITS, Hangzhou, China: IEEE, 2020, pp. 1–6. DOI: 10.1109/CITS49457.2020.9232457.

[10] J. Gedeon, F. Brandherm, R. Egert, T. Grube, and M. Mühlhäuser, "What the Fog? Edge Computing Revisited: Promises, Applications and Future Challenges," *IEEE Access*, vol. 7, pp. 152 847–152 878, 2019. DOI: 10.1109/ACCESS.2019.2948399.

[11] K. Fowler, "Mission-critical and safety-critical development," *IEEE Instrumentation and Measurement Magazine*, vol. 7, no. 4, pp. 52–59, 2004. DOI: 10.1109/MIM.2004.1383466.

[12] W. Liu, F. Lombardi, and M. Shulte, "A Retrospective and Prospective View of Approximate Computing," *Proceedings of the IEEE*, vol. 108, no. 3, pp. 394–399, 2020. DOI: 10.1109/JPROC.2020.2975695.

[13] H. B. Barua and K. C. Mondal, "Approximate Computing: A Survey of Recent Trends—Bringing Greenness to Computing and Communication," *Journal of The Institution of Engineers (India): Series B*, vol. 100, no. 6, pp. 619–626, 2019. DOI: 10.1007/s40031-019-00418-8.

[14] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," en, in *Proceedings of the 50th Annual Design Automation Conference*, ser. DAC '13, Austin Texas: ACM, 2013, pp. 1–9. DOI: 10.1145/2463209.2488873.

[15] I. Roboflow, *Udacity Self Driving Car Dataset*, 2020. [Online]. Available: https://public.roboflow.com/object-detection/self-driving-car/ (visited on 01/27/2025).

[16] *OpenMMLab Detection Toolbox and Benchmark*. [Online]. Available: https://github.com/open-mmlab/mmdetection/ (visited on 01/27/2025).

[17] A. Geitgey, *Face Recognition API*. [Online]. Available: https://github.com/ageitgey/face_recognition/ (visited on 01/27/2025).

[18] *ImageMagick*. [Online]. Available: https://imagemagick.org/ (visited on 01/27/2025).

[19] H. Sharif, Y. Zhao, M. Kotsifakou, *et al.*, "ApproxTuner: A Compiler and Runtime System for Adaptive Approximations," in *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '21, New York, NY, USA: ACM, 2021, pp. 262–277. DOI: 10.1145/3437801.3446108.

[20] B. Grigorian, N. Farahpour, and G. Reinman, "BRAINIAC: Bringing reliable accuracy into neurally-implemented approximate computing," in *21st International Symposium on High Performance Computer Architecture*, ser. HPCA, Burlingame, CA, USA: IEEE, 2015, pp. 615–626. DOI: 10.1109/HPCA.2015.7056067.

[21] P. Pandey and D. Pompili, "MobiDiC: Exploiting the Untapped Potential of Mobile Distributed Computing via Approximation," EN, in *International Conference on Pervasive Computing and Communications*, ser. PerCom, vol. 38, Sydney, NSW, Australia: IEEE, 2016, pp. 1–9. DOI: 10.1109/PERCOM.2016.7456515.

[22] Y. Li, Y. Chen, T. Lan, and G. Venkataramani, "MobiQoR: Pushing the Envelope of Mobile Edge Computing Via Quality-of-Result Optimization," in *37th International Conference on Distributed Computing Systems*, ser. ICDCS, Atlanta, GA, USA: IEEE, 2017, pp. 1261–1270. DOI: 10.1109/ICDCS.2017.54.

[23] A. Younis, T. X. Tran, and D. Pompili, "Energy-Latency-Aware Task Offloading and Approximate Computing at the Mobile Edge," in *16th International Conference on Mobile Ad Hoc and Sensor Systems*, ser. MASS, Monterey, CA, US: IEEE, 2019, pp. 299–307. DOI: 10.1109/MASS.2019.00043.

[24] S. Mittal, "A Survey of Techniques for Approximate Computing," en, *ACM Comput. Surv.*, vol. 48, no. 4, pp. 1–33, 2016. DOI: 10.1145/2893356.

[25] A. Younis, B. Qiu, and D. Pompili, "QLRan: Latency-Quality Tradeoffs and Task Offloading in Multi-node Next Generation RANs," in *16th Annual Conference on Wireless On-demand Network Systems and Services Conference*, ser. WONS, Klosters, Switzerland: IEEE, 2021, pp. 1–8. DOI: 10.23919/WONS51326.2021.9415574.

[26] R. Padilla, S. L. Netto, and E. A. B. Da Silva, "A Survey on Performance Metrics for Object-Detection Algorithms," EN, in *International Conference on Systems, Signals and Image Processing*, ser. IWSSIP, Niterói, Brazil: IEEE, 2020, pp. 237–242. DOI: 10.1109/IWSSIP48289.2020.9145130.

[27] I. Goiri, R. Bianchini, S. Nagarakatte, and T. D. Nguyen, "ApproxHadoop: Bringing Approximations to MapReduce Frameworks," en, in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS*, ser. 4, vol. 50, Istanbul, Turkey: ACM, 2015, pp. 383–397. DOI: 10.1145/2775054.2694351.

[28] Z. Wen, D. L. Quoc, P. Bhatotia, R. Chen, and M. Lee, "ApproxIoT: Approximate Analytics for Edge Computing," in *38th International Conference on Distributed Computing Systems*, ser. ICDCS, Vienna, Austria: IEEE, 2018, pp. 411–421. DOI: 10.1109/ICDCS.2018.00048.

[29] G. Hu, S. Rigo, D. Zhang, and T. Nguyen, "Approximation with Error Bounds in Spark," in *27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Rennes, FR: IEEE, 2019, pp. 61–73. DOI: 10.1109/MASCOTS.2019.00017.

[30] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "YOLOv7: Trainable Bag-of-Freebies Sets New State-of-the-Art for Real-Time Object Detectors," en, in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, ser. CVPR, Vancouver, BC, Canada: IEEE, 2023, pp. 7464–7475. DOI: 10.1109/CVPR52729.2023.00721.

[31] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017. DOI: 10.1109/TPAMI.2016.2577031.

[32] N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection," en, in *Computer Society Conference on Computer Vision and Pattern Recognition*, ser. CVPR'05, vol. 1, San Diego, CA, USA: IEEE, 2005, pp. 886–893. DOI: 10.1109/CVPR.2005.177.

[33] S. Ghafouri, K. Razavi, M. Salmani, *et al.*, "IPA: Inference Pipeline Adaptation to Achieve High Accuracy and Cost-Efficiency," *Journal of Systems Research*, vol. 4, no. 1, 2024, arXiv:2308.12871 [cs]. DOI: 10.5070/SR34163500.

[34] Y. Sani, A. Mauthe, and C. Edwards, "Adaptive Bitrate Selection: A Survey," *IEEE Communications Surveys and Tutorials*, vol. 19, no. 4, pp. 2985–3014, Jun. 2017. DOI: 10.1109/COMST.2017.2725241.

[35] E. Oliveira, A. R. D. Rocha, M. Mattoso, and F. C. Delicato, "Latency and Energy-Awareness in Data Stream Processing for Edge Based IoT Systems," en, *Journal of Grid Computing*, vol. 20, no. 3, p. 27, 2022. DOI: 10.1007/s10723-022-09611-4.

[36] D. L. Quoc, R. Chen, P. Bhatotia, C. Fetzer, V. Hilt, and T. Strufe, "StreamApprox: Approximate computing for stream analytics," en, in

Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference, ser. Middleware '17, Las Vegas, Nevada: ACM, 2017, pp. 185–197. DOI: 10.1145/3135974.3135989.

[37] M. Dobber, R. Van Der Mei, and G. Koole, "A prediction method for job runtimes on shared processors: Survey, statistical analysis and new avenues," en, *Performance Evaluation*, vol. 64, no. 7-8, pp. 755–781, Aug. 2007. DOI: 10.1016/j.peva.2007.01.001.

[38] M. Amiri and L. Mohammad-Khanli, "Survey on prediction models of applications for resources provisioning in cloud," en, *Journal of Network and Computer Applications*, vol. 82, pp. 93–113, 2017. DOI: 10.1016/j.jnca.2017.01.016.

[39] Y. Gao, X. Gu, H. Zhang, H. Lin, and M. Yang, "Runtime Performance Prediction for Deep Learning Models with Graph Neural Network," in *IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice*, ser. ICSE-SEIP, Melbourne, Australia: IEEE, 2023, pp. 368–380. DOI: 10.1109/ICSE-SEIP58684.2023.00039.

[40] *OpenTelemetry*. [Online]. Available: https://opentelemetry.io/ (visited on 01/27/2025).

[41] M. H. Kashani and E. Mahdipour, "Load Balancing Algorithms in Fog Computing," *IEEE Transactions on Services Computing*, vol. 16, no. 2, pp. 1505–1521, 2023. DOI: 10.1109/TSC.2022.3174475.

[42] K. A. Nuaimi, N. Mohamed, M. A. Nuaimi, and J. Al-Jaroodi, "A Survey of Load Balancing in Cloud Computing: Challenges and Algorithms," in *Second Symposium on Network Cloud Computing and Applications*, London England, UK: IEEE, 2012, pp. 137–142. DOI: 10.1109/NCCA.2012.29.

[43] ETSI, *5G; Service requirements for enhanced V2X scenarios*, Technical Specification (TS) 22.186, Sophia Antipolis Cedex - FRANCE, Nov. 2020.

[44] C.-L. Hwang and K. Yoon, *Multiple Attribute Decision Making* (Lecture Notes in Economics and Mathematical Systems 1), en, M. Beckmann and H. P. Künzi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1981, vol. 186, ISBN: 978-3-642-48318-9. DOI: 10.1007/978-3-642-48318-9.

[45] Y. Y. Haimes and D. A. Wismer, "Integrated system modeling and optimization via quasilinearization," en, *Journal of Optimization Theory and Applications*, vol. 8, no. 2, pp. 100–109, 1971. DOI: 10.1007/BF00928470.

[46] J. Branke, K. Deb, K. Miettinen, and R. Słowiński, *Multiobjective Optimization: interactive and evolutionary approaches* (Lecture notes in computer science 5252), eng. Berlin: Springer-Verlag, 2008, ISBN: 978-3-540-88908-3.

[47] A. Ledenev, *Pumba: Chaos testing tool for Docker*. [Online]. Available: https://github.com/alexei-led/pumba/ (visited on 01/27/2025).

[48] *Network Emulation - (tc-netem)*. [Online]. Available: https://www.man7.org/linux/man-pages/man8/tc-netem.8.html (visited on 01/27/2025).

[49] J. Jenkov, *Java Microbenchmark Harness (JMH)*. [Online]. Available: https://github.com/openjdk/jmh (visited on 01/27/2025).

[50] U. Gropengießer and E. Dietz, *MARQ repository*. [Online]. Available: https://doi.org/10.5281/zenodo.14731336 (visited on 01/27/2025).

[51] S. Luo, H. Xu, C. Lu, *et al.*, "Characterizing microservice dependency and performance: Alibaba trace analysis," EN, in *Proceedings of the 2021 ACM Symposium on Cloud Computing*, ser. SoCC '21, Seattle, WA, USA: ACM, 2021, pp. 412–426. DOI: 10.1145/3472883.3487003.

[52] Y. Brun, G. Edwards, J. Y. Bang, and N. Medvidovic, "Smart Redundancy for Distributed Computation," in *31st International Conference on Distributed Computing Systems*, ser. ICDCS, Minneapolis, MN, USA: IEEE, 2011, pp. 665–676. DOI: 10.1109/ICDCS.2011.25.

[53] G. Levitin, L. Xing, and Y. Dai, "Mission Abort Policy in Heterogeneous Nonrepairable 1-Out-of-N Warm Standby Systems," *IEEE Transactions on Reliability*, vol. 67, no. 1, pp. 342–354, 2018. DOI: 10.1109/TR.2017.2740330.

[54] K. Pohl, G. Böckle, and F. Van Der Linden, *Software Product Line Engineering*, en, 1st ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, ISBN: 978-3-540-24372-4. DOI: 10.1007/3-540-28901-1.

[55] F. Van Der Linden, K. Schmid, and E. Rommes, *Software Product Lines in Action*, en, 1st ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, ISBN: 978-3-642-09061-5. DOI: 10.1007/978-3-540-71437-8.

[56] M. A. Naily, M. R. A. Setyautami, R. Muschevici, and A. Azurat, "A Framework for Modelling Variable Microservices as Software Product Lines," en, in *Software Engineering and Formal Methods*, A. Cerone and M. Roveri, Eds., Cham: Springer International Publishing, 2018, pp. 246–261. DOI: 10.1007/978-3-319-74781-1_18.

[57] D. Batory, "Feature Models, Grammars, and Propositional Formulas," en, in *Software Product Lines*, H. Obbink and K. Pohl, Eds., Berlin, Heidelberg: Springer, 2005, pp. 7–20. DOI: 10.1007/11554844_3.

[58] K. Czarnecki, S. Helsen, and U. Eisenecker, "Formalizing cardinality-based feature models and their specialization," en, *Softw Process Imprv Pract*, vol. 10, no. 1, pp. 7–29, 2005. DOI: 10.1002/spip.213.

[59] U. Gropengießer, J. Liphardt, M. Matthé, and M. Mühlhäuser, "Feature Model Slicing for Real-time Selection of Mission-critical Edge Application," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '24, New York, NY, USA: Association for Computing Machinery, 2024, pp. 2446–2447. DOI: 10.1145/3691620.3695321.

[60] G. Gharibi, R. Tripathi, and Y. Lee, "Code2Graph: Automatic generation of static call graphs for python source code," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ser. ASE '18, Montpellier, France: ACM, 2018, pp. 880–883. DOI: 10.1145/3238147.3240484.

[61] T. Cerny, A. S. Abdelfattah, V. Bushong, A. Al Maruf, and D. Taibi, "Microservice Architecture Reconstruction and Visualization Techniques: A Review," in *International Conference on Service-Oriented System Engineering*, ser. SOSE, Newark, CA, USA: IEEE, 2022, pp. 39–48. DOI: 10.1109/SOSE55356.2022.00011.

[62] A. Doula, T. Güdelhöfer, M. Mühlhäuser, and A. S. Guinea, "Conformal Prediction for Semantically-Aware Autonomous Perception in Urban Environments," en, ser. CoRL 2024, 2024. [Online]. Available: https://openreview.net/forum?id=aaY5fVFMVf (visited on 01/27/2025).

[63] H. Zhao, Y. Li, N. He, *et al.*, "Anomaly Detection for Medical Images Using Self-Supervised and Translation-Consistent Features," *IEEE Transactions on Medical Imaging*, vol. 40, no. 12, pp. 3641–3651, 2021. DOI: 10.1109/TMI.2021.3093883.

[64] Y. Masuyama, Y. Bando, K. Yatabe, Y. Sasaki, M. Onishi, and Y. Oikawa, "Self-supervised Neural Audio-Visual Sound Source Localization via Probabilistic Spatial Modeling," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, ser. IROS, Las Vegas, NV, USA: IEEE, 2020, pp. 4848–4854. DOI: 10.1109/IROS45743.2020.9340938.

[65] R. S. Kannan, L. Subramanian, A. Raju, J. Ahn, J. Mars, and L. Tang, "GrandSLAm: Guaranteeing SLAs for jobs in microservices execution frameworks," in *Proceedings of the 14th EuroSys Conference 2019*, ser. EuroSys '19, Dresden, Germany: ACM, 2019, p. 16. DOI: 10.1145/3302424.3303958.

[66] M. Shafique, R. Hafiz, S. Rehman, W. El-Harouni, and J. Henkel, "Invited - Cross-layer approximate computing: From logic to architectures," en, in *Proceedings of the 53rd Annual Design Automation Conference*, ser. DAC '16, Austin, Texas: ACM, 2016, pp. 1–6. DOI: 10.1145/2897937.2906199.

[67] H. Ahmadvand, T. Dargahi, F. Foroutan, P. Okorie, and F. Esposito, "Big Data Processing at the Edge with Data Skew Aware Resource Allocation," in *IEEE Conference on Network Function Virtualization and Software Defined Networks*, ser. NFV-SDN, Heraklion, Greece: IEEE, 2021, pp. 81–86. DOI: 10.1109/NFV-SDN53031.2021.9665051.

[68] H. Ahmadvand, M. Goudarzi, and F. Foroutan, "Gapprox: Using Gallup approach for approximation in Big Data processing," en, *J Big Data*, vol. 6, no. 1, p. 20, 2019. DOI: 10.1186/s40537-019-0185-4.