Architectural Vision for Quantum Computing in the Edge-Cloud Continuum

Alireza Furutanpey[†][*], Johanna Barzen[‡], Marvin Bechtold[‡], Schahram Dustdar[†],
Frank Leymann[‡], Philipp Raith[†], Felix Truger[‡]
[†]Distributed Systems Group, TU Vienna first.last@dsg.tuwien.ac.at
[‡]Institute of Architecture of Application Systems, University of Stuttgart first.last@iaas.uni-stuttgart.de
[*]**Corresponding Author**

***Abstract*—Quantum processing units (QPUs) are currently exclusively available from cloud vendors. However, with recent advancements, hosting QPUs is soon possible everywhere. Existing work has yet to draw from research in edge computing to explore systems exploiting mobile QPUs, or how hybrid applications can benefit from distributed heterogeneous resources. Hence, this work presents an architecture for Quantum Computing in the edge-cloud continuum. We discuss the necessity, challenges, and solution approaches for extending existing work on classical edge computing to integrate QPUs. We describe how warm-starting allows defining workflows that exploit the hierarchical resources spread across the continuum. Then, we introduce a distributed inference engine with hybrid classical-quantum neural networks (QNNs) to aid system designers in accommodating applications with complex requirements that incur the highest degree of heterogeneity. We propose solutions focusing on classical layer partitioning and quantum circuit cutting to demonstrate the potential of utilizing classical and quantum computation across the continuum. To evaluate the importance and feasibility of our vision, we provide a proof of concept that exemplifies how extending a classical partition method to integrate quantum circuits can improve the solution quality. Specifically, we implement a split neural network with optional hybrid QNN predictors. Our results show that extending classical methods with QNNs is viable and promising for future work.**

***Index Terms*—Quantum Computing, Edge Computing, Compute Continuum, Split Computing, Circuit Cutting, Task Partitioning, DNN Partitioning, Classical-Quantum Hybrid Machine Learning, Quantum Neural Networks, Warm-Starting**

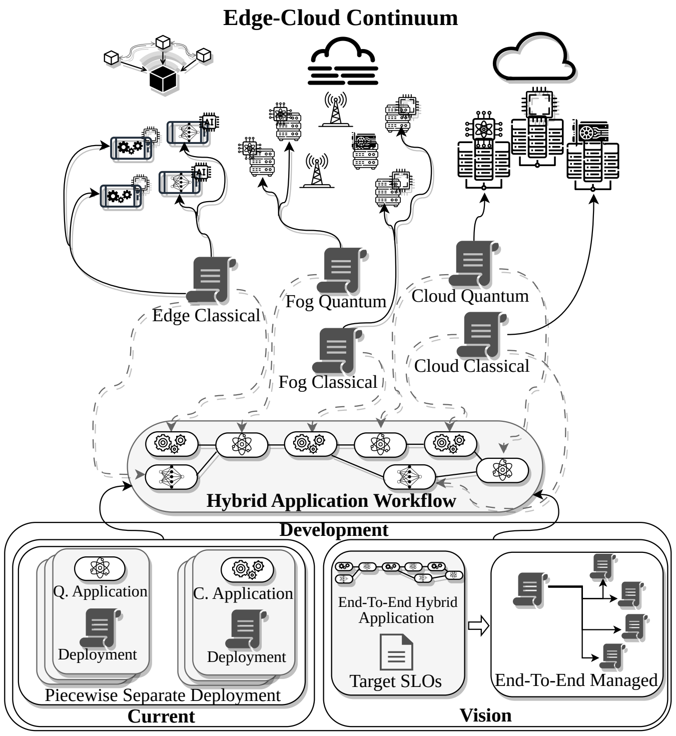

Fig. 1. A Distributed Classical-Quantum Hybrid Platform

I. Introduction

Noisy intermediate-scale quantum (NISQ) computers are error-prone, contain only a limited number of qubits, and impose restrictions on the depth of successfully executable circuits [31]. Yet, algorithms tailored towards NISQ devices started to demonstrate the viability of quantum computers in various fields, ranging from molecule simulation [27] to machine learning [11] and optimization problems [14]. Evidently, to advance research and development into practical applications of quantum algorithms, increasing the accessibility of quantum computers by introducing adequate abstractions is effective. Nevertheless, providing researchers and practitioners access to QPUs is challenging. Owing to the limited availability, complexity, and cost of Quantum Processing Units (QPUs), quantum computation for the masses may currently only be viable through cloud services that can hide the low-level machinery behind a convenient interface. However, while cloud providers can decrease the complexity and cost, we cannot exclusively rely on the efforts of hardware manufacturers to increase the accessibility of resources by simplifying the production and installation of QPUs. Instead, it is essential to draw from our experiences in classical computing on the ramifications of predominantly relying on centralized cloud platforms. Besides the privacy-related risks of entrusting third-party providers with sensitive data, the cloud computing paradigm bears numerous downsides, such as vendor lock-in and data centers posing a single point of failure vulnerable to outages. Additionally, a narrow cloud-centric view is inefficient since it leaves valuable resources close to the client (e.g., Edge, Fog) idle by indiscriminately offloading tasks to a remote server.

The edge-cloud continuum addresses the limitations of cloud computing by aggregating resources in a hierarchical distributed network ranging from constrained edge devices to cloud data centers. Unfortunately, after decades of relying on centralized architectures, the transition is slow, with semi- or fully decentralized platforms still needing to emerge [45]. Therefore, to avoid repeating the mistakes of classical computing, it is crucial to aid researchers in implementing quantum applications on distributed platforms before centralized approaches solidify.

Notably, deploying edge applications on mobile quantum devices is on the horizon with the recent advancements of diamond-based QPUs [25] that allow quantum computation at room temperature [37]. Hence, it is sensible to assume that quantum computers may soon become widely available for individuals and organizations. Analogous to how mobile Artificial Intelligence (AI) accelerators gave rise to performance-critical intelligent applications at the edge relying on Computer Vision (CV) or Natural Language Processing (NLP), the emergence of mobile QPUs will pave the way for a new class of intelligent applications that, for example, benefit from efficiently solving complex optimization tasks. Still, even with centralized approaches, the complexity of developing hybrid applications significantly obstructs advancements in quantum research. Practitioners must consider classical and hybrid components, choose the appropriate hardware, and manually manage the orchestration. To this end, this work aims to encourage quantum computation research at the edge. We describe key challenges and possible solution approaches for distributed hybrid classical-quantum architectures. Specifically, the focus is on task partitioning to illustrate the potential of drawing from an edge-cloud continuum and to explain the intricate interplay of numerous classical and quantum components the system designs stitch together into one cohesive unit. Ultimately, our vision is for a distributed hybrid platform that can automate the end-to-end process of orchestrating hybrid applications to emerge. Then, as conceptualized by Figure 1 and summarized by Table I, practitioners can dedicate their time solely to implementing the core logic of their applications and algorithms.

TABLE I
SUMMARY OF THE STATUS QUO AND OUR VISION

Category	Current Situation	Vision
Accessability	Restrictive, centralized, consolidates resources	Optionally centralized or decentralized
Complexity	Requires selection of components (e.g.,quantum computers), multiple vendors and frameworks	Heterogeneous components abstracted behind homogenous interfaces
Applications	Restricted to cloud, high network latency	Unrestriced, devices co-operation
Service Guarantees	Requires assessment end-to-end performance of manually stitched applications	SLOs, for single hybrid application. Automates orchestration resource allocation
Workflow	Must integrate individual classical and quantum service providers manually	Freely composable workflow of quantum and classical components
Resource Locality	Cloud only	Edge-Cloud Continuum

To concretize and increase the intuition of the abstract concepts of our work, we use a running example of implementing a distributed hybrid platform for Mobile Augmented Reality (MAR). Our choice is sensible, considering how conceiving methods to accommodate the demanding and complex requirements of MAR applications with edge computing generalize well to arbitrary applications and is an active area of research [61]. Moreover, a platform hosting MAR applications must reliably serve numerous intelligent tasks that can benefit from QPUs. For example, a navigation system for cyclists to safely navigate urban areas by preventing them from crashing into approaching vehicles requires object detection and tracking models from a distributed camera network [54].

We summarize our contributions as

- Highlighting the opportunities of incorporating QPUs into the edge-cloud continuum and how the distinct properties of QPUs introduce novel challenges.
- Introduce the architecture for a distributed hybrid platform with a variable composition of heterogeneous quantum and classical nodes.
- Demonstrating the feasibility of extending partitioning methods for classical Deep Neural Networks (DNN) with quantum embeddings by implementing, evaluating, and open-sourcing a proof of concept (PoC) of a splittable hybrid classical-quantum neural network[1].

Section II establishes the preliminary background and related work. Section III argues how future work on Quantum Edge Computing (QEC) can benefit from existing work on Classical Edge Computing (CEC) by drawing parallels to AI accelerators and their role in Edge Intelligence. The rest of this work is structured to progressively introduce the components of our envisioned platform from a top-down perspective. Section IV presents the core high-level serverless abstraction model for client programmers. Section V extends the model to provide first-class support for warm starting. Section VI focuses on lower-level system challenges for a distributed hybrid inference engine. Section VII describes and evaluates our PoC for a hybrid classical-quantum D(Q)NN for split inference. Section VIII concludes our work.

II. Background & Related work

Cloud-centric platforms have paved the way to make cost-efficient and large-scale applications accessible to the public. However, the emerging edge-cloud continuum accentuates the drawbacks of centralized architectures. Promising application paradigms, such as Edge Intelligence [17], heavily rely on the edge-cloud continuum and require autonomous management over the large and heterogeneous system. We argue that quantum computers can improve the quality of existing applications and pave the way for novel ones. The success of these applications is tied to available platforms that need to support developers in designing, writing, testing, deploying, and managing them. This section introduces concepts fundamental to our architectural vision and summarizes related work.

A. Orchestration

The services of centralized platforms that provide access to quantum computers can be combined with classical applications (i.e., Amazon offers event-based processing for their quantum service). Hence, practitioners and researchers must

[1]https://github.com/rezafuru/QuantenSplit

build hybrid applications by combining separate quantum and classical components. Worse, they are burdened with selecting different QPU technologies, devices, and compilers [23], [52].

1) Orchestration of Quantum Applications: Although quantum applications consist of classical and quantum components, they follow the same framework as classical computing in dividing orchestration into two distinct procedures [66]. First, workflow technologies manage control flows. Second, provisioning technologies handle the deployment of application components. Hence, we can reduce infrastructure complexity by extending existing systems to support quantum applications. Wild et al. present Tosca4Q, which extends Tosca to support workloads relying on quantum computers [69]. Weder et al. introduce Quantum Application Archives (QAAs), allowing orchestration methods to treat quantum applications as self-containing entities [66]. Later, Leymann et al. propose extending QAAs through a marketplace, with an architecture for a collaborative software platform to consider the development process [32].

2) Quantum Platforms: Several cloud offerings provide access to quantum computing as a service. However, it is still challenging to integrate managed quantum services cohesively into classical applications.

Garcia-Alonso et al. [23] present their proof-of-concept implementation of a Quantum API Gateway recommending a quantum computer target to run a given quantum application for Amazon Braket. Additionally, Beisel et al. [6] propose *Quokka*, a microservice-based framework to model and deploy quantum workflows. The authors propose a set of microservices that model the typical quantum workflow based on *Variational Quantum Algorithms (VQAs)* [12]. This workflow comprises circuit generation, execution, error mitigation, objective evaluation, and parameter optimization. The approach fully decouples each part of pre-processing, executing, and post-processing, allowing a flexible workflow definition. Further, Salm et al. [56] present a concept that automatically handles the analysis of quantum algorithms and the selection of quantum computers. Grossi et al. [28] build a prototypical platform inspired by Serverless Computing through which quantum developers can deploy their applications. They employ a scheduler that focuses on queue management and result retrieval. Leymann et al. [32] propose an architecture for a collaborative software platform for quantum applications that encompasses the development process and deployment aspect through a marketplace for quantum applications.

The systems so far have shown how platforms can enhance collaboration, improve the development of applications, and simplify deployment aspects (e.g., dynamically selecting a quantum computer). However, they only considered cloud systems and focused on software development. We see that complementary to our work as advances in these fields can enrich development aspects of our envisioned hybrid application platform. For example, an adoption of Tosca4Q [69] to model serverless applications [70] can make our envisioned serverless-based platform more accessible. Still, it requires a platform to manage hybrid applications across the edge-cloud continuum to make quantum applications accessible.

B. Serverless Edge Computing

A key problem of edge-cloud applications, such as edge intelligence, is the autonomous orchestration of edge-cloud applications [17]. Manual management is infeasible in these large-scale and geo-distributed infrastructures, so a platform that can autonomously manage application deployments is required.

Serverless Edge Computing is the natural extension of Serverless Computing that abstracts the underlying infrastructure and transparently deploys applications packaged as functions across the system [3], [45]. We argue autonomous management and simplified application development and deployment can be enablers of the emerging quantum computing paradigm. Nguyen et al. [47] present a holistic serverless platform that supports classic, quantum, and hybrid applications. Conversely, we envision a platform that spans the edge-cloud continuum and manages application deployments across heterogeneous infrastructures. The increased complexity stems from the composed applications and the sophisticated and fine-grained monitoring, as Section IV will discuss.

C. Task Partitioning

Task partitioning in classical edge computing and quantum computing are distinct research areas that address orthogonal problems. Nevertheless, they share a common idea in dividing a task into subtasks executable by isolated compute nodes.

Although we can draw from existing quantum computing partitioning methods to aid schedulers in their placement strategies, we must consider the unique properties of QPUs before we can generalize them to classical methods.

Partitioning in classical edge computing concerns distributing load for resource efficiency [34]. In quantum computing, splitting tasks between classical and quantum nodes is necessary for near-term applications to cope with the limitations of NISQ devices [31], and most common hybrid classical-quantum splitting patterns assign fixed roles to components [67]. Additionally, patterns for quantum computation are typically conceived to execute a particular class of algorithms and do not consider applications where a quantum algorithm is just one of several subtasks.

In contrast, as the limitations of quantum computation will gradually diminish, a platform should be able to accommodate new emerging patterns. For example, IBM's 433-qubit QPU announced in 2022 will soon be superseded by an 1'121-qubit system [18]. Further, for near- and long-term QPUs, the platform should dynamically adjust the workload between quantum and classical nodes according to target SLOs, internal (e.g., load), and external conditions (e.g., bandwidth).

1) Variational Quantum Algorithms: Variational Quantum Algorithm (VQA) is a generic framework for optimizing the parameters of a quantum circuit on a classical computer [12]. Depending on the target task, we can derive more specific algorithms, such as *Variational Quantum Eigensolver* for approximating the lowest eigenvalue of a matrix [62] or *Quantum Approximate Optimization Algorithms (QAOAs)* to approximate the solution of a combinatorial optimization problem [59]. Another notable instance of VQAs is *Quantum Neural Networks (QNNs)* which aim to improve the representation of classical neural networks with embeddings in the Hilbert space. Note, in literature, the distinction between VQAs, Quantum Machine Learning (QML), and QNNs is blurry; thus, for clarity, we refer to QNNs as models that are built and trained for typical ML tasks (e.g., Feature Extraction, Regression, or Classification).

2) Warm-Starting: The term *warm-starting* is ambiguous due to its widespread usage in classical and quantum computing. For example, it may refer to techniques that reduce resource usage in machine learning and optimization [2]. Contrastingly, in classical serverless computing, warm-starting typically relates to methods for preparing execution environments, such as reusing running containers [38]. This work considers warm-starting a general strategy for partially computing or preparing a quantum algorithm's output on auxiliary devices. Notably, warm-starting methods are not limited to classical-to-quantum and may be quantum-to-quantum or quantum-to-classical.

Hybrid classical-quantum systems can benefit from various warm-starting methods that utilize previously obtained solutions, approximations, or trained models [64]. For example, following the assumption that optimal variational parameters for similar problem instances solved with VQAs are in proximity, parameters can be transferred between instances as an initial point to warm-start from and improve upon [22]. Moreover, approximations that are cheaply generated by efficient classical algorithms can be utilized to initialize quantum circuits with a quantum state biased towards potential solutions rather than starting from a neutral initial state [19]. On the other hand, QNNs can benefit from pre-trained models through transfer learning, i.e., adapting a classical or hybrid model trained for a general task and training it further to tackle a similar or more precise subtask [39]. Evidently, as these warm-starting methods comprise a source algorithm from which information is drawn and a target algorithm that is enhanced with it, warm-starts may indicate potential ways of distributing both classical and quantum computational efforts in the continuum.

3) Depth and Widthwise D(Q)NN Partitioning: Depthwise refers to splitting a large DNN between vertical layers, i.e., it results in sequentially dependent partitions and does not increase parallelization. In classical computing, depthwise partitioning methods commonly aim to facilitate resource efficiency by allowing multiple devices to process a request across the continuum [20]. In quantum computing, depthwise partitioning may refer to stacking QNN circuits to mitigate cascading gate errors and accumulating noise during training [9].

Conversely, widthwise partitioning facilitates parallelization by horizontally splitting a layer or circuit. Quantum circuit cutting concerns addresses some limitations of NISQ devices by partitioning larger circuits into several smaller subcircuits [4]. Classical widthwise DNN partitioning is less common since AI accelerators parallelize the execution of layers. Still, in highly constrained environments without access to server-grade hardware, methods such as parallelizing filter computation of convolutional layers across devices are sensible [71].

We identify classical depthwise and quantum widthwise partitioning as viable methods for an essential component of our envisioned platform and will detail them in Section VI.

III. Quantum Computing and Edge Intelligence

We argue that research on Quantum Edge Computing (QEC) should directly extend Classical Edge Computing (CEC) rather than foster isolated communities. In particular, we find that future work on QEC can learn valuable lessons from existing work on Edge Intelligence (EI). Intelligent tasks commonly refer to solving problems that a program with classical control structures cannot compute tractably or with sufficient precision. EI leverages advancements in specialized hardware for constrained devices (e.g., AI accelerators) to push the computation of such tasks with modern AI methods (e.g., DNNs). Similarly, QEC can leverage the advancements in (mobile) QPUs to compute numerous tasks tractably (e.g., optimization problems) and with higher precision.

To identify where we can utilize prior experiences, this section draws parallels between the rise of energy-efficient AI accelerators and the challenges of integrating QPUs. Note that, despite the existence of a broader definition for "AI", literature in edge computing typically uses AI as synonymous with classical Machine and Deep Learning. To avoid ambiguity, we will adhere to the same convention. Moreover, for brevity, we refer to AI accelerators covering a broad class of chips suitable for mobile devices that can efficiently execute vendor-specific compiled computational graphs of DNNs as Tensor Processing Units (TPUs) [33].

A. Parallels between Mobile QPUs and TPUs

QPUs can efficiently solve some problems intractable for classical computers (e.g., integer factorization and discrete logarithms [60]). Notably, QPUs can utilize entanglement and superposition, two properties unique to quantum computing. Conversely, although adding a TPU is formally incomparable to the speed-ups and precision QPUs can provide, in practice, TPUs provide substantial speed-ups for complex tasks DNNs excel at.

For brevity, we will refer to DNNs specifically designed for resource-constrained devices (e.g., IoT devices) or DNNs compressed with quantization, pruning, or knowledge distillation methods [16], [26] as "Lightweight". Analogously, we refer to circuits designed to run on more constrained QPUs than the state-of-the-art as "shallow" or "narrow". The reference point is relative and aligns with the assumption that integrated chips in resource-constrained devices have lower capacity chips than contemporary server-grade hardware. Similarly to how TPUs are accelerators for lightweight DNNs [13] to solve CV and NLP problems, mobile QPUs (MQPUs) will allow constrained devices running shallow quantum circuits to solve optimization problems.

Conclusively, viewing QPUs as an accelerator for an orthogonal set of intelligent tasks, i.e., tasks impractical for classical handcrafted programs, is reasonable. Nevertheless, QPUs and TPUs can complement each other for non-orthogonal tasks in the near term. For example, the primary motivation of QNNs interestingly differs from the typical advantages of quantum computation. Since the training and inference of DNNs are highly parallelizable, modern classical hardware already efficiently accommodates them. Instead, QNNs can exploit properties of the Hilbert Space to find better representations [1]. Moreover, it has been shown that utilizing entanglement in training data can reduce the limits on learnability imposed by the (classical) no-free-lunch theorem [58]. Nevertheless, especially for high-dimensional data (e.g., Images), it is challenging to build and train QNNs since methods must map the high-dimensional representation to a low number of qubits. Therefore, currently, hybrid classical-quantum QNNs show more promising results. Unlike pure QNNs and other VQAs, hybrid QNNs consist of parameterized classical and quantum components, and first apply classical layers to find a suitable representation before encoding the features to a quantum state and passing them to a quantum circuit [36], [39].

B. Quantum for and on the Edge

In their seminal work, Deng et al. [17] proposed distinguishing Edge Intelligence between *AI for Edge* and *AI on Edge*. Analogous to their work, we suggest that future research in QEC should be categorized in Quantum on and for Edge.

1) Quantum on Edge (QoE): addresses the challenges of building systems for distributed applications that rely on QPUs. Research in this category can focus on integrating existing quantum systems into the continuum, such that classical resources across a hierarchical network can aid in reducing wait times and load on QPUs. Alternatively, the focus may be on generalizable approaches for extending existing classical systems to increase the solution quality with QPUs. Section V and Section VI introduce ideational methods for both subcategories to facilitate this direction.

2) Quantum for Edge (QfE): focuses on solving optimization problems in edge computing. Unlike QoE, it is not fcomplementarity to the classical AI counterpart. Rather, quantum researchers should aim to outperform AI methods. Existing classical methods could be utilized to warm-start quantum optimization. Moreover, they can serve as valuable benchmarks, marking a threshold quantum algorithms must cross before they are viable in practice. It is worth noting that quantum algorithms have been shown to outperform classical optimization methods for specific problem instances, which can be identified with a high probability prior to running the algorithm [43].

IV. An Architecture for a Distributed Hybrid Compute Platform

This section introduces the envisioned architecture illustrated in Figure 2. We first provide a high-level model of an edge-cloud continuum and relate it to our running example of a MAR platform that aims to implement our architecture. Then, we describe the individual components and how clients interact with them so that the remaining sections can reference the architecture. We do not claim that the components we in-

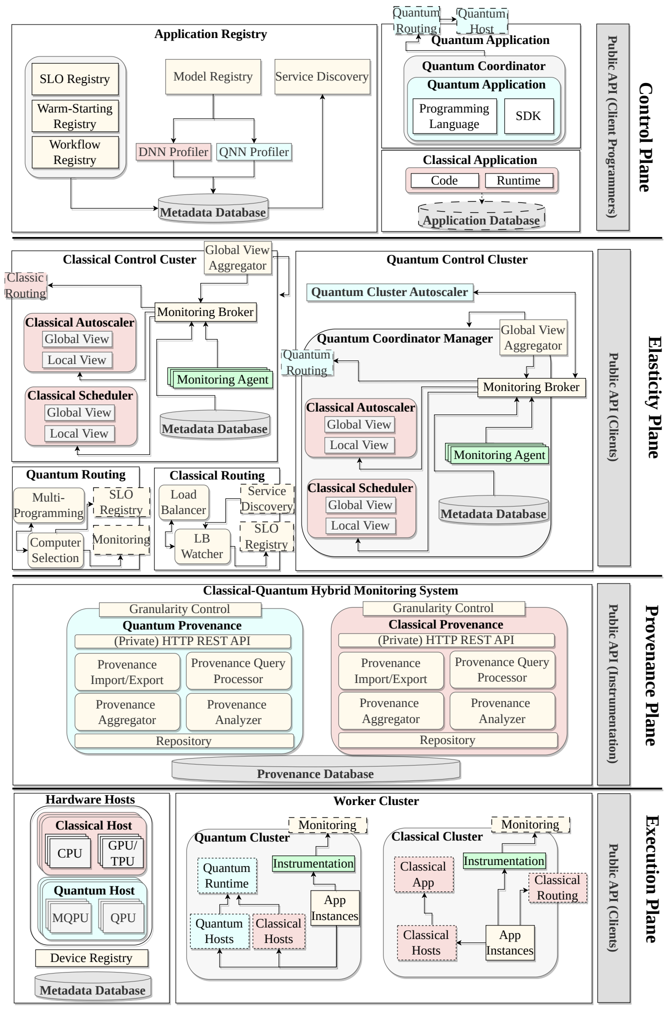

Fig. 2. Our envisioned Architecture

troduce in our architecture are sufficient for a hybrid platform. Rather, we believe it forms a foundation for future work as a starting point for a new research directive to advance quantum and edge computing.

A. Resource and Scenario Formulation

Our conceptual MAR platform distinguishes between system designers, client programmers, and clients. The system designers seek methods to build a platform that allows client programmers to deploy applications or experiments. Client programmers write and deploy MAR applications and have varying requirements. Clients are devices that access applications by issuing requests. Our objective is an architecture that maximizes the rate of advancements in quantum applications and algorithms by *unconditionally* minimizing the complexity for client programmers at the expense of system designers and platform providers.

1) Service Level Objectives and Agreements: With SLOs, clients can specify quantifiable metrics according to their application's requirements that a platform should uphold. Typically providers bundle SLOs together as Service Level Agreements (SLAs). Nevertheless, we do not require a distinction between SLOs and SLAs since the latter is simply a constructible contract of the former. Guaranteeing SLOs is difficult even in purely classical serverless (edge-)cloud systems [44], [53], [57]. The following proposes solutions for conceptualizing an architectural platform vision with the additional consideration of integrating QPUs. Section III stressed the importance of extending existing methods to accommodate QPUs instead of conceiving novel methods unaware of recent advancements in CEC. Therefore, we draw from ongoing research in classical computing from a systems perspective, intending to extend existing state-of-the-art classical methods instead of competing with them. For the remainder of this work, we refer to a component or service as *SLO-aware* if it has access to the SLO registry and implements a corresponding decision mechanism.

2) Resource Assumptions on the Edge-Cloud Continuum: Resources are spread across three domains: edge, fog, and cloud. There is no universally agreed-upon definition for the edge and fog domain. Hence, we remain as general as possible without compromising the practicability of our architecture. Defining the fog domain as clustered computational nodes significantly nearer to client devices than the closest cloud data center is sufficient for our purposes. The edge domain comprises the clients and nearby SLO-aware load balancers [8]. Table II summarizes each domain's properties and the available resources to which our MAR platform has access. In particular, for QPUs, we refer to the properties of current or expected near-term devices. While once QPUs become mature enough, they may be as accessible as classical hardware, system designers will need to address the near-term limitations of QPUs. We distinguish between mobile- and server-grade hardware. The former can only execute a subset of the latter. Capacity refers to the size of a model and how many instances we can spawn on demand. *Unlimited* means we can host as many instances of the largest off-the-shelf models as necessary. *High* implies some trade-off between the size and the number of instances of a model where it is possible to exhaust the available capacity when the number of concurrent client requests exceeds a certain threshold. *Medium* and *Low* suggest additional limitations to the hardware, e.g., how the circuit depth and the number of qubits on NISQ devices limit the size of a QNN. In our running example, the platform providers aim to place QPUs near several major cities. Therefore, they purchased numerous mobile-grade QPUs (e.g., the announced diamond-based QPUs of Quantum Brilliance [37] or SaxonQ [25]) that they expect to have sufficient capacity for most models. Since they cannot feasibly host server-grade QPUs (e.g., IBM's planned 1'121-qubit system [18]) in every supported location, the requests that require larger circuits are either forwarded to a remote cloud data center or processed with partitioning methods we will elaborate on in Section VI. Lastly, we label the capacity of edge TPUs as "very low" since they can typically only host one lightweight DNN in memory. Contrastingly, in the fog and cloud domain, we can host multiple instances of the same DNN on virtualized GPUs. A notable limitation for a resource may not be exclusive to a single domain but is a concern system designers should be particularly conscious about in that domain. Heterogeneity refers to the numerous chip architectures with varying constraints. Lightweight DNNs and Circuit Width/Depth refers to the limitation of TPUs and MQPUs to execute larger models. Interference refers to the performance degradation from serving virtualized resources to multiple clients [46].

TABLE II
SUMMARY OF RESOURCES ACROSS DOMAINS

Domain	Dominating Cost	Type	Grade	Capacity	Notable Limitation
Edge	Energy - Client may not warm start	CPU	Mobile	Low	Heterogeneity
		TPU	Mobile	Very Low	Lightweight DNNs
Fog	Load - Requests may have to be routed to the cloud.	QPU	Mobile	Low	Circuit Width/Depth
		CPU	Server	High	Outages
		GPU	Server	High	Outages
Cloud	Communication - Remote data centers incur high latency	QPU	Server	Medium	Availability
		CPU	Server	Unlimited	Interference
		GPU	Server	Unlimited	Interference

B. Architecture Planes and Components

The architecture differentiates between four planes according to their function and intended interaction with other components, clients, or client programmers. Each plane exposes private or public APIs. Private APIs are only accessible by internal components, whereas public APIs are accessible by external entities, i.e., clients and client programmers.

1) Execution Plane: Application instances run on the *execution plane*. It consists of *Public APIs*, *Hardware Hosts*, and *Worker Clusters*. The public APIs allow clients to interact with the application via access points.

The hardware hosts are subdivided according to the supported application types, i.e., into classical and quantum nodes. The quantum hosts are further subdivided according to their *grade*, i.e., QPUs are server-grade and MQPUs mobile-grade. The classical hosts additionally subdivide to consider AI workload. Each node is registered by the *device registry* that associates data describing their capacity (e.g., memory size, VRAM, qubits) and stores it into the *metadata* storage. The metadata storage is a highly-available key-value storage (e.g., ETCD used in Kubernetes) accessible by other components. Especially for the continuum, where nodes can arbitrarily join and leave the system, the metadata must be highly available and not remain stale to measure the system's overall capacity accurately.

The *Worker Clusters* consists of at least one hardware host and represents the application environments that may rely on one or multiple hardware nodes. Since quantum and classical environments are separate, it is necessary to distinguish between classical and quantum worker clusters. Naturally, quantum applications will require classical workers for auxiliary tasks (e.g., pre-processing, measurement). Once a worker completes a task, a classical worker is responsible for forwarding the result. The result is forwarded to another worker cluster for distributed applications that partition the task on multiple devices. The result is forwarded to the client for monolithic applications or the last partition of a distributed application. Instrumentation tools send telemetry data to the monitoring system by accessing private APIs of the Provenance Plane.

2) Provenance Plane: The *Provenance Plane* encapsulates a highly available distributed Provenance database through which real-time monitoring data is stored and shared. It consists of a *Quantum Provenance* and a *Classical Provenance* system, offering private APIs to access and store telemetry data. It is crucial to conceive methods that consider the intrinsic properties of QPUs, to create reliable and predictable systems for quantum workloads. Notably, error rates vary depending on a QPU's current state, i.e., exclusively collecting classically relevant data (e.g., load) for quantum and hybrid applications is insufficient for SLOs with solution quality targets. Classical monitoring for a platform deploys instrumentation tools (Execution Plane) alongside the application to collect telemetry data on workload trends and resource usage of function instances (e.g., load, CPUs, GPUs, I/O). Orthogonally, a quantum provenance system gathers information on the state of QPUs to analyze errors, such as properties of Quantum Circuits, QPUs, Compilation, and Execution, as proposed by Weder et al. [65]. A platform can utilize provenance data for error mitigation and to aid schedulers with upholding SLOs by assessing the currently expected solution quality. Platforms that support hybrid applications should integrate quantum provenance with classical instrumentation to form one cohesive monitoring system for simplified access to various heterogeneous devices.

The objective of a monitoring system is to collect the minimal data necessary for informing schedulers to uphold SLOs. Conceiving hybrid systems is non-trivial, as finding a balance is already challenging for classical edge-cloud and hybrid systems monitoring [24], [53]. A system that trivializes monitoring by aggressively collecting data may ease the task of a scheduler but can cause resource congestion across the edge-cloud continuum [24]. Conversely, collecting insufficient telemetry data will significantly reduce the system's scalability as it cannot appropriately route requests or scale resources up and down. We argue that a *granularity mechanism* capable of adjusting the frequency of quantum and classical monitoring data according to the current workload's characteristics is one of the principal challenges that future work must address before a hybrid platform can emerge. Nevertheless, monitoring itself is simply a precondition. The following describes how our architecture supports resource efficiency and elasticity based on available monitoring data.

3) Elasticity Plane: The *Elasticity plane* is the central organ of the system that decides how to allocate resources and route requests according to client SLOs, metadata, and monitoring data. The *Control clusters* are subdivided between Quantum and Classical Control Clusters. The former manages quantum application instances (e.g., scale-out quantum coordinators), while the latter manages classical application instances (e.g., horizontal scaling of applications). Each control cluster manages a set of worker clusters (see Execution Plane) that are dynamically scaled up or down according to the application instances the cluster can control. Monitoring Agents are responsible for the worker clusters and relay data to the control cluster's monitoring broker. A *Monitoring Broker* disseminates the data across the components. A *Classical Autoscaler* and *Classical Scheduler* maintain a local and global view of the control cluster's state. Classical autoscaler and schedulers exist in Quantum Control Clusters, since Quantum Coordinator Applications (See Control Plane) are classic. The *Quantum Coordinator Manager* supervises the Classical Autoscaler and Scheduler to scale and place the Quantum Coordinator Application (see Control Plane) instances.

The *Local View* contains fine-grained information about the Worker Clusters (e.g., CPU usage per second). In contrast, the *Global View* contains coarse-grained information (e.g., average CPU usage over an hour) about neighboring control clusters. The coarse-grained data consists of fine-grained local data collected by a Global View Aggregator that periodically publishes a summary, i.e., the global view consists of exchanged summaries of local views.

The control cluster's messaging topology and broker partially address the granularity control of monitoring data and permit an elastic control mechanism that can scale the entire system by adequately allocating its limited resources. The implementation of the autoscaler and scheduler is interchangeable, and system designers may experiment with various methods. The *Quantum Cluster Autoscaler* is inspired by the work of Tamiru et al. and Gandhi et al. and is an SLO-aware cluster autoscaler capable of adding and removing Quantum Hosts from worker clusters to process the incoming workload.

Classical Routing is inspired by the work by Raith et al. [53] and consists of a *Load Balancer* and a *Load Balancer (LB) Watcher*. The Load Balancer is a high-throughput and low-overhead component that re-directs incoming requests to application instances or other clusters (e.g., because no application instance is running). The Load Balancer Watcher is SLO-aware and periodically refreshes the load balancer's state to update the decision mechanism.

The *Quantum Routing* component differentiates itself from Classical Routing, as it considers additional challenges to improve the resource efficiency of quantum hosts. *Quantum Computing Selection* is particularly valuable for the edge-cloud continuum as it introduces further heterogeneity. The selection method is another freely interchangeable component. For example, system designers may opt to use the method proposed by Quetschlich et al. [52] and replace it once they find a more suitable alternative. *Multi-programming* in quantum computing has a comparable role to *virtualization* in classical computing, i.e., it allows sharing of the resources of quantum computers among multiple circuits [15]. However, unlike in classical computing, where we can readily select existing mature virtualization methods, multi-programming is more involved and should be considered together during encoding [48] and influences quantum computer selection.

4) Control Plane: The Control plane exposes an API to client programmers to deploy and manage their applications. The public API should allow client programmers to define hybrid applications as workflows without separately deploying classical and quantum parts. The platform analyzes the workflow during the registry and partitions it into classical and quantum applications. The *Application Registry* encapsulates several registries responsible for storing application services, SLO targets, and parameterized models. *The Model Registry* is similar to a model zoo (e.g., torch image models [68]). However, here the models may either be classical neural networks or quantum circuits. In addition, profilers associate static metadata to each model (e.g., number of parameters, circuit depth, layer types). Profiler metadata supplements the schedulers and autoscalers with valuable information to predict resource usage more accurately (e.g., a Swin-Tiny will incur higher usage than a Swin-Base [35]). Our architecture supports warm-starting as a first-class citizen, which Section V will elaborate on. Service Discovery enables Quantum and Classical Routing to locate running application instances.

Classical Applications consist of the application code and a runtime that executes the code upon creating an application instance. The architecture does not require any assumptions about the runtime, i.e., the runtime can range from containers to WebAssembly modules. Application may optionally provision arbitrary *Application Databases* (e.g., relational, object-based).

The *Quantum Application* is more complex as it requires the interplay between quantum and classical resources. The *Quantum Coordinator* contains the Quantum Application, implemented by the client programmers. The Quantum Application is written in a particular programming language and an SDK for quantum computing. The Quantum Coordinator coordinates the execution of a quantum application (i.e., quantum circuit). Quantum Applications requires routing capabilities to connect the classical part of the quantum application, that pre- and post-processes input and output, with the quantum part, the QPU that executes the quantum circuit. Moreover, Quantum Coordinator requires Quantum Routing capabilities to select a suitable Quantum Host or to perform Multi-Programming to increase QPU utilization efficiency.

V. Warm-Starting at the Edge

Warm-starting aligns with the objectives of a distributed hybrid platform, i.e., it facilitates drawing from resources across the continuum. Ideally, applications can pre-process input before passing it to a remote server. Warm-starting methods in the context of quantum computing are categorizable into Classical-To-Quantum (C2Q), Quantum-To-Quantum (Q2Q), or Quantum-To-Classical (Q2C) [64]. Each category has an input and an output format. For example, C2Q expects classical input, and the output format should suit a quantum algorithm.

Nevertheless, we argue that system designers must subdivide the categories further to include neural input and output formats, such as Neural-To-Quantum (N2Q) or Classic-To-Neural (C2N), for two reasons. First, neural methods rely on AI accelerators that may not be present or have alleviated energy consumption, i.e., it is indispensable for a scheduler to know hardware properties to hit SLO targets. Second, although the output of a classical DNN is classical, the network weights may be tuned to extracted features tailored for a particular class of algorithms.

A. Current and Future Role of Warm-Starting

Currently, a common motivation for warm-starting is to reduce the dependency on QPU time due to cost and limited availability [64]. However, we stress that the importance of warm starting lies in improving the solution quality by combining classical and quantum algorithms. Moreover, once QPUs mature to a point where we can entirely forgo classical computation, the research focus can shift to Q2Q warm-starting, where smaller client devices can partially onload quantum algorithms for resource efficiency. For example, warm-starting can be a means to embed performance guarantees of classical algorithms into quantum algorithms and can reduce the amount of training data required for (Q)ML.

A downside of warm-starting is that it increases the applications' complexity as it introduces more parts that must be managed, i.e., it is crucial to introduce a convenient interface that supports warm-starting as a first-class citizen further to shift complexity from client programmers to platform providers. Notably, warm-starting is chainable. Hence, we can naturally integrate warm-starting methods in the edge-cloud continuum if a platform exposes an interface that resembles the hierarchical properties regarding device capacities in the network.

The following describes the requirements and proposed solution approaches for an aware hierarchical warm-starting programming model. Hierarchical refers to how the warm-starting methods are composable in a *pipeline* that resembles their resource usage requirements. The proposed interface does not rely on any assumption regarding the availability and limitations of QPUs, i.e., it treats each method in the pipeline as exchangeable building blocks. The current progress of available QPUs is reflected in the client programmers by informing them about the feasibility of their planned warm-starting pipeline. For example, the platform could disable support for Q2Q warm-starting at the network's edge until MQPUs find widespread adoption in end devices, such as smartphones.

B. Hierarchical Warm-Starting

In our running example, the MAR platform aims to improve resource efficiency with hierarchical warm-starting. The load heavily fluctuates for city-scale applications according to date and time. By chaining warm-starting hierarchically, idle computational resources of edge and fog nodes can be utilized, e.g., to reduce offloading to the cloud.

Warm-starting is a broad term encompassing numerous classes of methods [64]. The challenge is to conceive an interface flexible enough to remain convenient without exposing low-level details, such as manually selecting devices and fallback mechanisms, to the clients. An interface would be maximally flexible if it forces client programmers to define every single step of the execution, where to deploy which part at which node, and to manually configure the quantum executions (e.g., device, compiler) for every method in the pipeline. An interface that does not restrict the configuration space is especially undesirable when considering the heterogeneity of the continuum. Specifically, it would not be sufficient to provide a single configuration for a method, and there is no guarantee for the availability of a particular device configuration at the edge or fog. Conversely, constraining client programmers exclusively to a list of pre-implemented solutions is counterproductive as it hinders innovation, i.e., they should at least be able to (optionally) provide their warm-starting method.

To summarize, the responsibility of a platform is to build the infrastructure and provide adequate abstractions to access the resources. A dedicated interface for warm-starting should allow clients to define composable workflows to process a warm-starting pipeline, i.e., client programmers can register pre-processing steps for warm-starts through the control plane from our architecture in Figure 2 that may run on CPUs, TPUs, or QPUs deployed at the client device or fog nodes.

Consider Figure 3 for the following example. Three clients execute the same application using the public API of the control plane. The client programmers defined a warm-starting pipeline for their applications. Hence, the pipeline and its methods are placed in the warm-starting registry, and the system decides where to position the models in the continuum based on the profiler metadata. However, the clients request varying target qualities; hence, the platform applies different intensities of warm-starting before sending the task to a quantum cloud vendor. Intensity refers to the expected solution quality of a quantum algorithm with an input processed by a warm-starting method. In Figure 3, client C1 cannot achieve

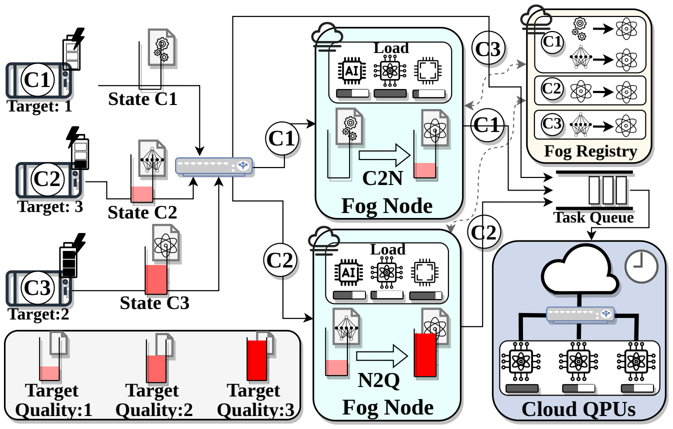

Fig. 3. High-level Sequence of Hierarchical Warm-Starting

any pre-processing in the edge due to energy constraints (indicated by a low battery symbol) and therefore forwards the input in its original classical representation. Since C1 registered at least one pre-processing step at the fog, the load balancer routes the request to a fog node, applying the step that maps to a C2Q warm-start. Conversely, if C1 had the resources to pre-process its input for a target neural network, the fog node would have taken over processing its output further to warm-start the quantum task, resulting in a chain of C2N and N2Q warm-start. Client C2 can perform resource-conscious pre-processing for a C2N warm-start. Still, since it does not achieve the required target quality, the request is routed to a fog node with available QPUs to apply further pre-processing for an N2Q warm-start. Client C3's request is directly routed to the cloud by the load balancer, since C3 had enough onboard resources for pre-processing the task to the target quality without relying on fog nodes.

VI. A Distributed Inference Engine for Hybrid Classical-Quantum Neural Networks

While the last two sections introduced high-level concepts of the architecture, this section focuses on lower-level system designs for platform designers and how client-programmers may implement a distributed hybrid application. We extend our running example and discuss how the MAR platform may support an inference engine with hybrid classical-quantum neural networks.

(D)QNNs are uniquely qualified as a representative application beyond DNN inference to cover how a distributed platform can adapt as the limitations of QPUs are gradually lifted for three reasons. First, a hybrid QNN is composable of parameterized classical and quantum nodes. Contrastingly, other VQAs typically operate non-parameterized classical components exclusively for pre-/post-processing and to optimize the parameters. For a hybrid QNN, there are additional components with no statically predefined role assignments, i.e., we can represent how systems adapt as they progressively replace classical with quantum nodes proportional to availability and advancements in QPUs. Second, we can split a neural network horizontally by layer or cut individual layers vertically and view each partition as an isolated computational graph. Then, a simple inference request (e.g., image classification) can emulate the behavior of a complex task with numerous classical and quantum subtasks that a system must coordinate to compute a single solution. Third, DNN partitioning and collaborative inference are well-established research areas in CEC that consider the heterogeneity of classical hardware and the resource asymmetry between edge, fog, and cloud nodes [41]. Hence, we can directly extend existing work to determine whether QPUs may improve the quality of classical methods and are not restricted to orthogonal problems infeasible for classical computers.

A. System Challenges

We consider the domain properties summarized in Table II to describe the challenges system designers may face when implementing our architecture.

Unlike regular business logic, intelligent tasks (e.g., image classification, object detection) rely on specialized hardware, i.e., regardless of whether we include QPUs, the platform must treat inference requests as a workload with distinct characteristics.

1) Volatility: The scheduler must dynamically adapt to two sources of volatility. First, outages in the fog domain are frequent. Moreover, unlike in the cloud, classical fog resources cannot seamlessly scale horizontally, i.e., requests may have to be routed to the cloud. Second, fog and cloud QPUs may be scarce, and depending on their current state, they may not hit the target solution quality SLO.

2) Device Heterogeneity: While the challenges of heterogeneity of classical components are only tangentially related to the integration of QPUs, minimal consideration regarding the numerous accelerators is necessary. Compilers map classical DNN to computational graphs, and vendors have varying support for operations limiting the available layer types and activation functions [33].

3) Task Chaining and Bandwidth Consumption: To fully draw from the resources on the continuum, we require methods that onload some computation on client-side accelerators. However, mobile devices can typically only host a single network in memory, and swapping out DNN weights from storage incurs significant overhead. Hence, latency-sensitive applications sending subsequent inference requests for different tasks must offload, leaving valuable resources idle. Additionally, when numerous clients compete for limited bandwidth by streaming high-dimensional image data, the limited bandwidth will inevitably lead to erratic response delays.

4) Optional Quantum Embeddings: Although the availability of QPUs is steadily increasing, clients cannot expect the same graceful scaling of classical resources in the cloud currently, i.e., depending on the load, it may not be possible to hit latency SLOs with quantum layers. Therefore, the inference engine should be flexible enough to skip computing quantum embeddings for near- and intermediate-term devices.

5) Utilizing Mobile Quantum Devices: Diamond quantum accelerators are expected to be mature enough soon for commercial use [25], [37]. However, regardless of how MQPUs improve, analogous to classical hardware, we assume that server-grade hardware will consistently outperform their mobile counterparts. The challenge is to conceive methods that can leverage the advantages of MQPUs, ideally without sacrificing solution quality.

B. Solution Approach

The simple solution for accommodating multiple tasks and configurable solution qualities considers a separate DNN for each variation. This is difficult to maintain and inflexible, forcing client programmers to implement and re-deploy entire architectures for each task. Worse, it does not address challenge 3), i.e., applications relying on multiple tasks cannot draw from client resources without a significant latency penalty. Instead, we describe how partitioning methods lead to composable architectures that naturally define small deployable applications.

1) Classical Split Computing: Depthwise split computing addresses challenges 2)-4) and groups layers into partitions. Each partition is a feature extractor, and sequentially applying them is a particular form of hierarchical warm-starting we introduced in Section V. Platforms can include pre-trained DNNs in the warm-starting registry of the Control Plane from. Additionally, client programmers may register modules according to their requirements. For example, edge devices can optionally perform preliminary feature compression, and fog nodes can apply a small- or medium-sized feature extractor according to solution quality and latency targets.

To address challenge 2), we require an encoder suitable for constrained end devices (e.g., Smartphones) composed of operations widely supported by the various vendors of AI accelerators. To address challenge 3), the encoder should perform initial feature extraction and find a minimal representation for a sufficient statistic on several downstream tasks to reduce bandwidth consumption. Then, the server can select an interchangeable DNN for additional feature extraction according to the configured latency and accuracy SLO. Lastly, a QNN layer should embed the features to potentially improve the representation before passing it to the classifier. However, to address challenge 4), i.e., to cope with the limited availability of QPUs, the QNN should be optional.

2) Quantum Circuit Cutting: Quantum circuit cutting addresses challenge 5). The idea is to cut large circuits that require many qubits widthwise into smaller subcircuits requiring fewer qubits [4] by strategically cutting circuit wires [10], [49] and gates [4], [42]. Figure 4 illustrates an example in which one wire and two gates are cut. Wire cutting separates circuit wires through multiple measurements with different observables O_i and subsequent initializations of the qubit to state $|\psi_i\rangle$, while gate cutting substitutes two-qubit gates with varying combinations of single-qubit gates. The stacked subcircuits in Figure 4 indicate the generation of multiple variations for each subcircuit.

The widthwise partitioning enables each subcircuit instance to be executed individually on smaller quantum devices, which may be more readily available. Following the execution of these subcircuits, a classical post-processing procedure is applied to recombine the results obtained from the individual subcircuits, ultimately reconstructing the output of the original circuit as a linear combination of the subcircuit results.

This approach facilitates the distribution of quantum circuit computations across multiple QPUs without necessitating quantum communication. As a result, quantum circuit cutting offers the opportunity to harness the power of several smaller MQPUs at the edge, enabling the computation of larger quantum circuits. Moreover, it promotes the more flexible placement of quantum circuits across resources of the compute continuum. Additionally, once MQPUs are wildly available, we can leverage them to parallelize the execution of subcircuits. This parallelization can alleviate the overhead associated with each cut, significantly improving computational efficiency and scalability.

3) Inference Flow: To provide intuition on how our proposed approaches can serve client requests with varying requirements, consider the flow of inference requests in Figure 5.

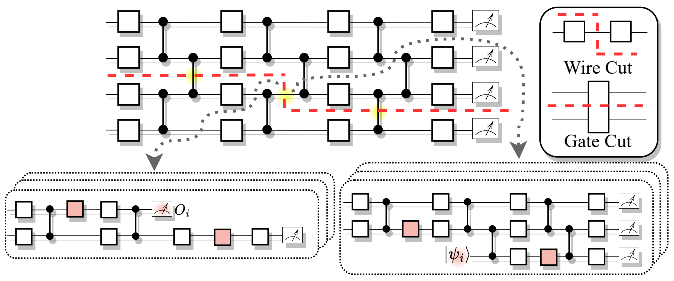

Fig. 4. Circuit Cutting Basics

For simplicity, we restrict the example to image classification. However, the image classification tasks may be different. For example, one client classifies artwork in a museum to retrieve a description using a virtual tour guide. Simultaneously, another client is interested in retrieving descriptions of the local fauna. Clients with AI accelerators apply neural compression methods for preliminary feature extraction. Solid black lines represent the flow of a request within a domain. The dashed red lines represent inter-domain data transfers. The dashed gray lines represent the telemetry and provenance data collected adjacent to inference requests by a runtime spanning all domains. The runtime is detailed by the Elasticity Plane of our architecture in Figure 2 and collects telemetry data to periodically update the SLO-aware load balancer to perform informed decisions based on the state of each participating node and client configurations. Dashed black lines represent an alternative path, i.e., one request flows only to one of the choices.

An edge load balancer routes the request to the load balancer of a fog cluster. Depending on load and client requirements, the request is routed to a Fog GPU node or the cloud. After feature extraction, a load balancer decides whether the classical embedding should be passed to a QNN before classification. The QNN may be executed on a Quantum Fog Node or sent to a remote cloud provider (e.g., due to privacy or availability). However, based on our assumption, the MQPUs are more constrained than the server-grade QPUs from cloud providers. Hence, circuit cutting methods can aid MQPUs in achieving a target solution quality. A request ends after the classification label is sent as a response to the client.

Section VII will detail the DNN architecture and demonstrate the viability of our proposed approach. Addressing challenges 1), 5), and introducing runtime components (e.g., SLO-aware schedulers, deployment mechanisms) are other promising research directives but out of scope for this work. Still, the following describes how a platform may realistically prepare and deploy the neural network components.

4) Application Preparation and Deployment: The individual operations of a DNN form a computational graph. Moreover, partitioning methods naturally demarcate a monolithic DNN into connectable vertices. The vertices represent coarse-grained classical layers or QNN circuits, and one or multiple consecutive vertices form one *depthwise partitioned deployment unit* (e.g., a container). Alternatively, we can further partition a vertice to create one *widthwise partitioned deployment unit* (e.g., with circuit cutting). Notably, depthwise methods define isolated compute nodes, which we can transparently combine with widthwise methods, i.e., from an outside view, an adequate abstraction can present a cut circuit as a single coarse-grained layer.

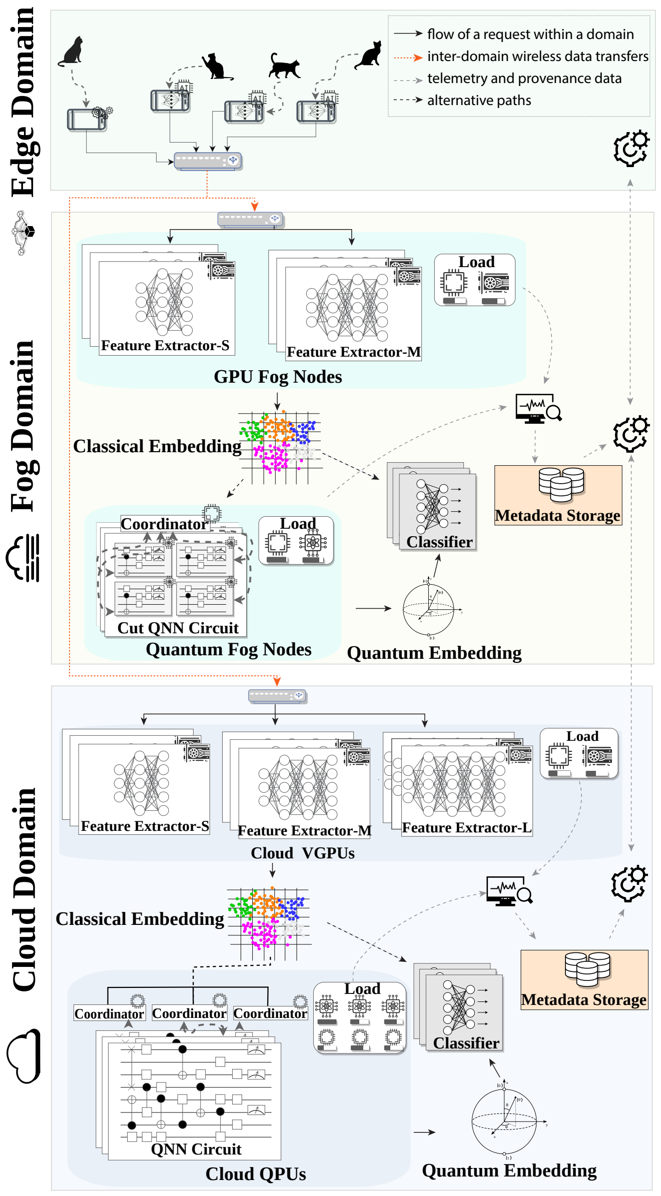

Fig. 5. Inference Engine Request Flow

The client programmers may provide hints to the platform via annotations, but the application should be deployable as a single (monolithic) workflow. It is the responsibility of the Control Plane of our architecture to create the deployment units before spawning Quantum and Classical Application instances. From the point of view of the client programmers, they have deployed a single application. However, the runtime system should be aware that the application is split into multiple parts. Figure 6 illustrates an example with a computational graph of coarse-grained layers. A coarse-grained

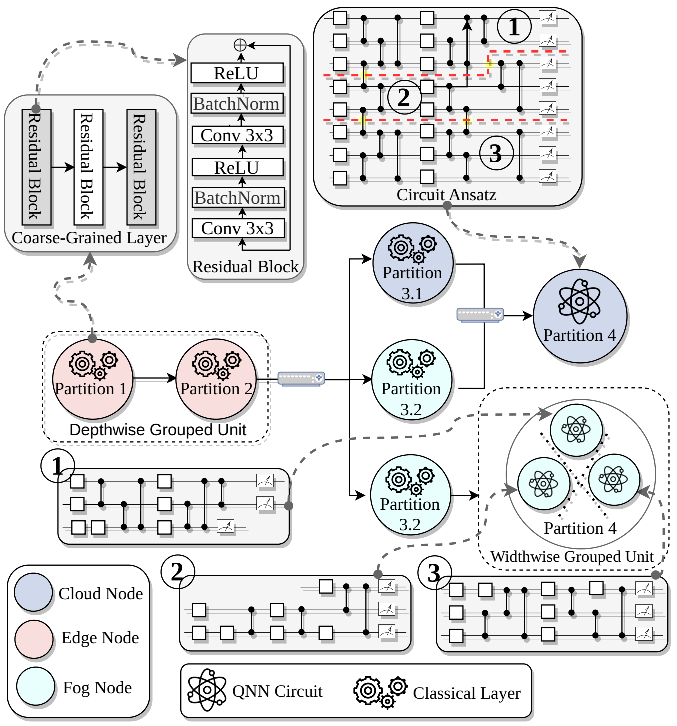

Fig. 6. Coarse Grained Deployment Units

layer consists (recursively) of finer-grained layers. The nodes are enumerated to indicate the processing sequence, and a subindex indicates a branching path. A node with the same index and subindex implies the same partition deployed on different nodes. Partitions 1-2 are grouped depthwise and will be deployed as one unit on edge devices. Partition 3.2 is deployed on cloud and fog nodes, while Partition 3.1 is a different model deployed exclusively on cloud nodes. For example, Partitions 3.1 and 3.2 could be Feature Extractor L and M from Figure 5. An SLO-aware load balancer routes the output of Partition 2 to a variation of Partition 3. Lastly, the output of Partition 3 is passed on to one of the instances of Partition 4. Partition 4 is deployed on fog and cloud nodes. However, it is a QNN circuit that a cloud QPU can execute but must be partitioned widthwise (i.e., cut into subcircuits as described in Section VI-B) for the constrained MQPUs at the fog nodes.

VII. Split Inference with Classical-Quantum Hybrid Predictors

Arguably, demonstrating that we can extend classical DNN partitioning methods to classical-quantum DQNN is a necessary precondition for a distributed hybrid inference engine. Hence, to show the viability of our visions, this section addresses challenges 2)-4) from Section VI (and partially 1) with a partitionable neural network architecture where the runtime of the Elasticity Plane can freely decide using a hybrid or a classical predictor.

To encourage follow-up work, we open-source a repository that provides a convenient framework to add configurable experiments with new circuit or model implementations.

A. Problem Formulation

As advocated for in Section III, we extend a method originally conceived for CEC to QEC instead of disregarding existing work. Specifically, we introduce *QuantenSplit*, a simple modification of the split inference method *FrankenSplit* [21] with depthwise DNN partitioning, to support quantum embeddings with QNNs for image classification.

We choose FrankenSplit since it is not limited to a single head-tail pair, so client devices do not have to swap out head weights whenever two subsequent requests require a different head network. Moreover, FrankenSplit focuses the local resources on bandwidth reduction with a variational feature compression model, i.e., an extension to support QNNs addresses challenges 3) and 4) from Section VI-A. The method draws from the Information Bottleneck (IB) [63] principle to achieve considerably higher compression rates than handcrafted codecs without sacrificing predictive strengths.

We omit the formal, conceptual details and instead refer to the original work [21]. Here, it is sufficient to consider FrankenSplit as a framework to train a variational autoencoder that is particularly selective about the signals it discards during compression. Notably, attaching different (split) neural networks for multiple downstream tasks to the autoencoder is possible. In other words, it permits a platform to deploy a universal encoder to all participant clients agnostic towards their particular inference requests.

To determine whether FrankenSplit is generalizable to hybrid classical-quantum QNN predictors, we answer the following question: *Do the highly compressed features of the classical universal encoder generalize to downstream tasks with QNNs?*. With this, we aim to show the usefulness of our envisioned platform from Section IV and to provide evidence for the viability of our distributed inference engine discussed in Section VI.

B. Methodology

To answer the above question, we must show that we can attach hybrid and classical predictors to a single autoencoder-backbone pair. We consider four datasets resulting in eight predictors, i.e., for each dataset, we attach one hybrid and one classical classification model to the base network.

1) Compression and Feature Extraction: Figure 7 illustrates the architecture of the modular neural network. The backbone is a pre-trained split classical neural network that further extracts features. It is possible to attach several backbones of varying sizes (or architectures) such that the network is deployable, as depicted in Figure 5. Nevertheless, we only require attaching multiple (hybrid) classification models to a single pre-trained backbone for our purposes. The encoder only relies on widely-supported operations to deal with device heterogeneity in the edge domain. The number in the bracket denotes the stage depth. The decoder blocks smooths out

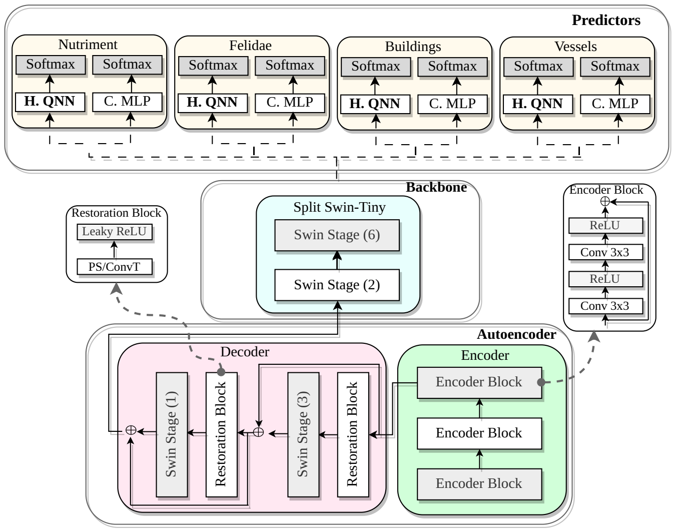

Fig. 7. QuantenSplit Network Architecture

the quantized features and transforms features to suit the backbone. The split Swin-Tiny backbone is described in [35], except the first two stages are discarded. The difference from the original work is that we attach hybrid classification models in addition to the classical ones.

2) Classification Models: A classical classifier is a simple two-layer Multi-Layer Perceptron (MLP). Figure 8 illustrates the Ansatz of the QNN with four qubits and layers. A Hybrid

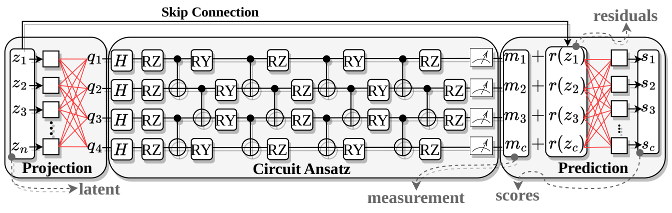

Fig. 8. Hybrid QNN with Four Qubits and Layers

QNN model takes as input the n-dimensional real-valued feature vector $\mathbb{Z}^n$ and classically projects it to a vector with dimensions equal to the number of qubits. Then, it passes the features as input to the Ansatz. Regardless of circuit depth, it first applies a Hadamard H Gate and a parameterized Z-rotation RZ to embed features in the quantum node. Next, it applies a repeating sequence of trainable variational layers. A layer consists of pairwise (shifted) C-NOT gates followed by alternating parameterized Y- or Z-rotations, i.e., a layer with Y-rotation is followed by a layer with Z-rotations. The number of layers is a hyperparameter given by the depth of the circuit. Lastly, it passes the measurements to a fully connected layer to output the class scores. The skip connection is inspired by classical Residual Neural Networks [29]. Adding the skip connection is configurable as a hyperparameter.

C. Training and Implementation

We first separately train the autoencoder on the 1.28 ImageNet [55] training samples according to the baseline objective function of FrankenSplit. Then, we freeze the parameters and train each attached classification model sequentially using the cross entropy loss function. We use Adam optimization [30] for the autoencoder and all classification models. Applying widthwise partitioning methods, such as circuit cutting, is out of the scope of this work. A detailed description of the training parameters can be seen in the configuration files of the accompanying repository.

We did not perform exhaustive hyperparamter tuning or experiments regarding optimizers. We implement our models using PyTorch [51] and CompressAI [5]. The backbones and pre-trained weights are from PyTorch Image Models [68]. For numerical simulations of the quantum circuits, we use PennyLane [7]. To ensure reproducibility and facilitate follow-up work, we extend torchdistill [40].

D. Evaluation

Our experiments are conducted on small-scale simulations with considerably fewer classes than the original work. Therefore, to draw meaningful insights from our results, the dimensions of the classical models are set equal to the number of qubits of their hybrid counterparts., i.e., the MLP first projects the high-dimensional backbone features to a low-dimensional classical embedding. For example, in an experiment with four qubits, we compare a hybrid predictor with a classical baseline predictor where the first layer of the MLP projects the backbone features to a four-dimensional representation. To emulate the scenario of Section VI with clients requesting

TABLE III
TRAINING AND TEST SUBSETS OF ILSVRC

Task	Classes	Training Samples	Test Samples
Nutriment	10	13’000	500
Felidae	13	16’900	650
Buildings	14	18’200	700
Vessels	23	29’900	1150

inference from diverse environments, we create four thematically unrelated datasets from the 1000 labels from the ILSVRC classification task. Table III summarizes each dataset representing a different location requiring separate predictors. The accompanying repository contains a script and instructions to recreate the datasets.

We run experiments with 4, 6, 8, and 10 qubits with a classical predictor as the baseline. The depth of the circuit is fixed at 8. We performed additional experiments with varying depth sizes and found that increasing the depth yields diminishing accuracy improvement. Table IV summarizes our results. The Plots in Figure 9 and Figure 10 show how a hybrid model without and with the skip connection compares to their classical counterpart for each dataset. Relative to the classical

TABLE IV
TOP-1 (ERR)OR OF (C)LASSIC, (H)YBRID, HYBRID WITH (RES)IDUALS

	Qubits	Top-1 Err. C. (%)	Top-1 Err. H. (%)	Top-1 Err. H. Res. (%)
Nutriment	4	13.00	37.13	12.11
	6	11.73	25.20	10.40
	8	11.30	16.90	10.07
	10	10.69	14.80	9.58
Felidae	4	19.82	31.57	19.05
	6	17.60	29.07	16.77
	8	16.77	18.77	15.85
	10	16.56	18.31	15.31
Buildings	4	9.29	31.57	8.57
	6	7.26	14.86	6.86
	8	6.14	10.57	5.71
	10	5.74	9.00	5.27
Vessels	4	32.43	62.00	30.69
	6	27.82	48.52	26.00
	8	25.48	31.56	24.96
	10	24.26	25.87	23.91

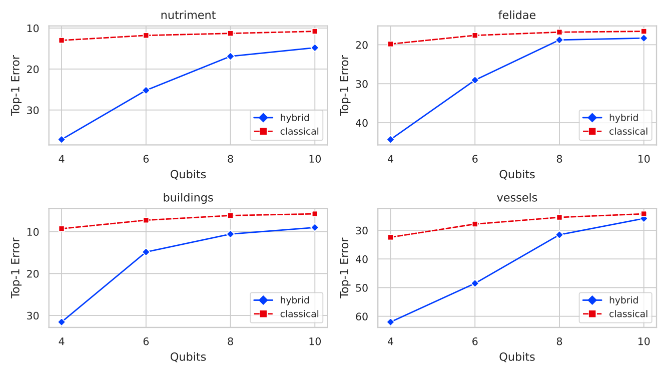

Fig. 9. Hybrid QNN without Skip Connection

baselines, Hybrid QNNs *without* skip connections gradually approach comparable, albeit still worse, Top-1 error as we increase the number of qubits. For 2 and 4 qubits, the Top-1 error is roughly 20-30% worse while the difference narrows to 2-5%. Interestingly, Hybrid QNNs *with* skip connections consistently outperform the classical baselines across all numbers of qubits. Although the motivation of skip connections in

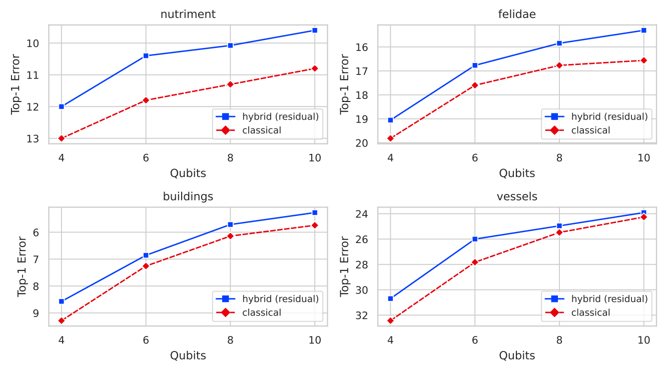

Fig. 10. Hybrid QNN with Skip Connection

classical residual networks is to mitigate accuracy saturation for very deep neural networks, they essentially learn a residual function. Considering the autoencoder-backbone pair already heavily processes the input data, we hypothesized that a QNN could find more suitable representations for the classical features for some instances. In contrast, a QNN could decrease the performance for samples already sufficiently processed for classification. The QNN narrowing the performance gap with increasing qubits is consistent with our assumptions. The model without a skip connection cannot find a representation as good as the initial input for a low number of qubits. Conversely, the QNN with a skip connection can learn the residual function and sees a performance gain for the samples, where a quantum embedding is useful.

A skip connection was the first intuitive attempt to provide empirical evidence, and the initial results seem promising. Nevertheless, we remind the reader that QuantenSplit only serves as a PoC to determine whether our vision is viable. The evaluation strategy is not extensive enough, and thus, our results should not be considered conclusive. Moreover, even with the skip connection, the hybrid model only marginally outperforms the classical model across all tasks simulations with a *noise free* device. We did not experiment with optimization algorithms more appropriate for QNNs and did not spend considerable effort conceiving a suitable circuit design. Further, the backbone is classical, i.e., the extracted features may be biased favorably towards classical predictors. Future work can significantly improve our results by experimenting with more sophisticated approaches for mapping low-dimensional qubits to a high-dimensional feature space [50]. Additionally, once training large QNN extractors is feasible, it would be interesting to determine whether we can train the classical compression model to find more suitable representations for quantum embeddings.

VIII. CONCLUSION

This work presented our vision of integrating quantum computing into the edge-cloud continuum. We summarized existing literature in quantum and classical computing relevant to our work and subsequently described the importance of extending existing Classical Systems for the edge. Then, we introduced an architecture for a hybrid classical-quantum platform and identified the critical challenges of integrating QPUs. We focused on facilitating research efforts in quantum applications with warm-starting and AI inference for distributed intelligent tasks. Lastly, we extended a classical split inference method to support Hybrid QNNs optionally. Our results provide empirical evidence for the viability of our vision and suggest that our ideas are interesting research directives.

ACKNOWLEDGMENT

We would like to thank Fabian Bühler, Benjamin Weder, Vladimir Yussupov and Florian Kowarsch for their valuable input. This work was partially funded by the BMWK projects *PlanQK* (01MK20005N), *EniQmA* (01MQ22007B), and *SeQuenC* (01MQ22009B).

References

[1] Amira Abbas, David Sutter, Christa Zoufal, Aurélien Lucchi, Alessio Figalli, and Stefan Woerner. The power of quantum neural networks. *Nature Computational Science*, 1(6):403–409, 2021.

[2] Jordan Ash and Ryan P Adams. On warm-starting neural network training. *Advances in neural information processing systems*, 33:3884–3894, 2020.

[3] Mohammad S Aslanpour, Adel N Toosi, Claudio Cicconetti, Bahman Javadi, Peter Sbarski, Davide Taibi, Marcos Assuncao, Sukhpal Singh Gill, Raj Gaire, and Schahram Dustdar. Serverless edge computing: vision and challenges. In *2021 Australasian Computer Science Week Multiconference*, pages 1–10, 2021.

[4] Marvin Bechtold, Johanna Barzen, Frank Leymann, Alexander Mandl, Julian Obst, Felix Truger, and Benjamin Weder. Investigating the effect of circuit cutting in QAOA for the MaxCut problem on NISQ devices. *arXiv preprint arXiv:2302.01792*, 2023.

[5] Jean Bégaint, Fabien Racapé, Simon Feltman, and Akshay Pushparaja. Compressai: a pytorch library and evaluation platform for end-to-end compression research. *arXiv preprint arXiv:2011.03029*, 2020.

[6] Martin Beisel, Johanna Barzen, Simon Garhofer, Frank Leymann, Felix Truger, Benjamin Weder, and Vladimir Yussupov. Quokka: A service ecosystem for workflow-based execution of variational quantum algorithms. In *Service-Oriented Computing–ICSOC 2022 Workshops: ASOCA, AI-PA, FMCIoT, WESOACS 2022, Sevilla, Spain, November 29–December 2, 2022 Proceedings*, pages 369–373. Springer, 2023.

[7] Ville Bergholm, Josh Izaac, Maria Schuld, Christian Gogolin, Shahnawaz Ahmed, Vishnu Ajith, M Sohaib Alam, Guillermo Alonso-Linaje, B AkashNarayanan, Ali Asadi, et al. Pennylane: Automatic differentiation of hybrid quantum-classical computations. *arXiv preprint arXiv:1811.04968*, 2018.

[8] Kirill L Bogdanov, Waleed Reda, Gerald Q Maguire Jr, Dejan Kostić, and Marco Canini. Fast and accurate load balancing for geo-distributed storage systems. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 386–400, 2018.

[9] Dmytro Bondarenko and Polina Feldmann. Quantum autoencoders to denoise quantum data. *Physical review letters*, 124(13):130502, 2020.

[10] Lukas Brenner, Christophe Piveteau, and David Sutter. Optimal wire cutting with classical communication. February 2023.

[11] M Cerezo, Guillaume Verdon, Hsin-Yuan Huang, Lukasz Cincio, and Patrick J Coles. Challenges and opportunities in quantum machine learning. *Nature Computational Science*, 2(9):567–576, 2022.

[12] Marco Cerezo, Andrew Arrasmith, Ryan Babbush, Simon C Benjamin, Suguru Endo, Keisuke Fujii, Jarrod R McClean, Kosuke Mitarai, Xiao Yuan, Lukasz Cincio, et al. Variational quantum algorithms. *Nature Reviews Physics*, 3(9):625–644, 2021.

[13] Yinpeng Chen, Xiyang Dai, Dongdong Chen, Mengchen Liu, Xiaoyi Dong, Lu Yuan, and Zicheng Liu. Mobile-former: Bridging mobilenet and transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5270–5279, 2022.

[14] Jaeho Choi and Joongheon Kim. A tutorial on quantum approximate optimization algorithm (qaoa): Fundamentals and applications. In *2019 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 138–142. IEEE, 2019.

[15] Poulami Das, Swamit S Tannu, Prashant J Nair, and Moinuddin Qureshi. A case for multi-programming quantum computers. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 291–303, 2019.

[16] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE*, 108(4):485–532, 2020.

[17] Shuiguang Deng, Hailiang Zhao, Weijia Fang, Jianwei Yin, Schahram Dustdar, and Albert Y Zomaya. Edge intelligence: The confluence of edge computing and artificial intelligence. *IEEE Internet of Things Journal*, 7(8):7457–7469, 2020.

[18] Jay DGambetta. IBM's roadmap for scaling quantum technology. https://research.ibm.com/blog/ibm-quantum-roadmap. (accessed: 05.05.2023).

[19] Daniel J Egger, Jakub Mareček, and Stefan Woerner. Warm-starting quantum optimization. *Quantum*, 5:479, 2021.

[20] Alireza Furutanpey and Schahram Dustdar. Adaptive and collaborative inference: Towards a no-compromise framework for distributed intelligent systems. 2022.

[21] Alireza Furutanpey, Philipp Raith, and Schahram Dustdar. Frankensplit: Saliency guided neural feature compression with shallow variational bottleneck injection. *arXiv preprint arXiv:2302.10681*, 2023.

[22] Alexey Galda, Xiaoyuan Liu, Danylo Lykov, Yuri Alexeev, and Ilya Safro. Transferability of optimal qaoa parameters between random graphs. In *2021 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 171–180. IEEE, 2021.

[23] Jose Garcia-Alonso, Javier Rojo, David Valencia, Enrique Moguel, Javier Berrocal, and Juan Manuel Murillo. Quantum software as a service through a quantum api gateway. *IEEE Internet Computing*, 26(1):34–41, 2021.

[24] Zacharias Georgiou, Moysis Symeonides, Demetris Trihinas, George Pallis, and Marios D Dikaiakos. Streamsight: A query-driven framework for streaming analytics in edge computing. In *2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC)*, pages 143–152. IEEE, 2018.

[25] DSaxonQ GmbH. SaxonQ Technology qubits in diamond. https://saxonq.com/technology. (accessed: 05.05.2023).

[26] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129:1789–1819, 2021.

[27] Harper R Grimsley, Sophia E Economou, Edwin Barnes, and Nicholas J Mayhall. An adaptive variational algorithm for exact molecular simulations on a quantum computer. *Nature communications*, 10(1):3007, 2019.

[28] Michele Grossi, Luca Crippa, Antonello Aita, Giacomo Bartoli, Vito Sammarco, Eleonora Picca, N Said, Filippo Tramonto, and Federico Mattei. A serverless cloud integration for quantum computing. *arXiv preprint arXiv:2107.02007*, 2021.

[29] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[30] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[31] Frank Leymann and Johanna Barzen. The bitter truth about gate-based quantum algorithms in the nisq era. *Quantum Science and Technology*, 5(4):044007, 2020.

[32] Frank Leymann, Johanna Barzen, Michael Falkenthal, Daniel Vietz, Benjamin Weder, and Karoline Wild. Quantum in the cloud: application potentials and research opportunities. *arXiv preprint arXiv:2003.06256*, 2020.

[33] Wenbin Li and Matthieu Liewig. A survey of ai accelerators for edge environment. In *Trends and Innovations in Information Systems and Technologies: Volume 2 8*, pages 35–44. Springer, 2020.

[34] Li Lin, Xiaofei Liao, Hai Jin, and Peng Li. Computation offloading toward edge computing. *Proceedings of the IEEE*, 107(8):1584–1607, 2019.

[35] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021.

[36] Seth Lloyd, Maria Schuld, Aroosa Ijaz, Josh Izaac, and Nathan Killoran. Quantum embeddings for machine learning. *arXiv preprint arXiv:2001.03622*, 2020.

[37] Quantum Brilliance Pty Ltd. Quantum Brilliance Hardware technology applications. https://quantumbrilliance.com/quantum-technology-application, 2023. accessed: 05.05.2023.

[38] Johannes Manner, Martin Endreß, Tobias Heckel, and Guido Wirtz. Cold start influencing factors in function as a service. In *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, pages 181–188. IEEE, 2018.

[39] Andrea Mari, Thomas R Bromley, Josh Izaac, Maria Schuld, and Nathan Killoran. Transfer learning in hybrid classical-quantum neural networks. *Quantum*, 4:340, 2020.

[40] Yoshitomo Matsubara. torchdistill: A modular, configuration-driven framework for knowledge distillation. In *Reproducible Research in Pattern Recognition: Third International Workshop, RRPR 2021, Virtual Event, January 11, 2021, Revised Selected Papers*, pages 24–44. Springer, 2021.

[41] Yoshitomo Matsubara, Marco Levorato, and Francesco Restuccia. Split computing and early exiting for deep learning applications: Survey and research challenges. *ACM Computing Surveys*, 55(5):1–30, 2022.

[42] Kosuke Mitarai and Keisuke Fujii. Constructing a virtual two-qubit gate by sampling single-qubit operations. *New Journal of Physics*, 23(2):023021, 2 2021.

[43] Charles Moussa, Henri Calandra, and Vedran Dunjko. To quantum or not to quantum: towards algorithm selection in near-term quantum optimization. *Quantum Science and Technology*, 5(4):044009, oct 2020.

[44] Stefan Nastic, Andrea Morichetta, Thomas Pusztai, Schahram Dustdar, Xiaoning Ding, Deepak Vij, and Ying Xiong. Sloc: Service level objectives for next generation cloud computing. *IEEE Internet Computing*, 24(3):39–50, 2020.

[45] Stefan Nastic, Philipp Raith, Alireza Furutanpey, Thomas Pusztai, and Schahram Dustdar. A serverless computing fabric for edge & cloud. In *2022 IEEE 4th International Conference on Cognitive Machine Intelligence (CogMI)*, pages 1–12. IEEE, 2022.

[46] Ripal Nathuji, Aman Kansal, and Alireza Ghaffarkhah. Q-clouds: managing performance interference effects for qos-aware clouds. In *Proceedings of the 5th European conference on Computer systems*, pages 237–250, 2010.

[47] Hoa T Nguyen, Muhammad Usman, and Rajkumar Buyya. Qfaas: A serverless function-as-a-service framework for quantum computing. *arXiv preprint arXiv:2205.14845*, 2022.

[48] Yasuhiro Ohkura, Takahiko Satoh, and Rodney Van Meter. Simultaneous execution of quantum circuits on current and near-future nisq systems. *IEEE Transactions on Quantum Engineering*, 3:1–10, 2022.

[49] Tianyi Peng, Aram Harrow, Maris Ozols, and Xiaodi Wu. Simulating large quantum circuits on a small quantum computer. *Physical Review Letters*, 125:150504, 3 2019.

[50] Evan Peters, João Caldeira, Alan Ho, Stefan Leichenauer, Masoud Mohseni, Hartmut Neven, Panagiotis Spentzouris, Doug Strain, and Gabriel N Perdue. Machine learning of high dimensional data on a noisy quantum processor. *npj Quantum Information*, 7(1):161, 2021.

[51] Automatic Differentiation In Pytorch. Pytorch, 2018.

[52] Nils Quetschlich, Lukas Burgholzer, and Robert Wille. Predicting good quantum circuit compilation options. *arXiv preprint arXiv:2210.08027*, 2022.

[53] Philipp Raith, Thomas Rausch, Schahram Dustdar, Fabiana Rossi, Valeria Cardellini, and Rajiv Ranjan. Mobility-aware serverless function adaptations across the edge-cloud continuum. In *2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC)*, pages 123–132, 2022.

[54] Thomas Rausch, Waldemar Hummer, Christian Stippel, Silvio Vasiljevic, Carmine Elvezio, Schahram Dustdar, and Katharina Krösl. Towards a platform for smart city-scale cognitive assistance applications. In *2021 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, pages 330–335. IEEE, 2021.

[55] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[56] Marie Salm, Johanna Barzen, Uwe Breitenbücher, Frank Leymann, Benjamin Weder, and Karoline Wild. The nisq analyzer: automating the selection of quantum computers for quantum algorithms. In *Service-Oriented Computing: 14th Symposium and Summer School on Service-Oriented Computing, SummerSOC 2020, Crete, Greece, September 13-19, 2020 14*, pages 66–85. Springer, 2020.

[57] Wonik Seo, Sanghoon Cha, Yeonjae Kim, Jaehyuk Huh, and Jongse Park. Slo-aware inference scheduler for heterogeneous processors in edge platforms. *ACM Transactions on Architecture and Code Optimization (TACO)*, 18(4):1–26, 2021.

[58] Kunal Sharma, M. Cerezo, Zoë Holmes, Lukasz Cincio, Andrew Sornborger, and Patrick J. Coles. Reformulation of the no-free-lunch theorem for entangled datasets. *Phys. Rev. Lett.*, 128:070501, Feb 2022.

[59] Ruslan Shaydulin and Yuri Alexeev. Evaluating quantum approximate optimization algorithm: A case study. In *2019 tenth international green and sustainable computing conference (IGSC)*, pages 1–6. IEEE, 2019.

[60] P.W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994.

[61] Yushan Siriwardhana, Pawani Porambage, Madhusanka Liyanage, and Mika Ylianttila. A survey on mobile augmented reality with 5g mobile edge computing: architectures, applications, and technical aspects. *IEEE Communications Surveys & Tutorials*, 23(2):1160–1192, 2021.

[62] Jules Tilly, Hongxiang Chen, Shuxiang Cao, Dario Picozzi, Kanav Setia, Ying Li, Edward Grant, Leonard Wossnig, Ivan Rungger, George H Booth, et al. The variational quantum eigensolver: a review of methods and best practices. *Physics Reports*, 986:1–128, 2022.

[63] Naftali Tishby, Fernando C. Pereira, and William Bialek. The information bottleneck method, 2000.

[64] Felix Truger, Johanna Barzen, Marvin Bechtold, Martin Beisel, Frank Leymann, Alexander Mandl, and Vladimir Yussupov. Warm-starting and quantum computing: A systematic mapping study. *arXiv preprint arXiv:2303.06133*, 2023.

[65] Benjamin Weder, Johanna Barzen, Frank Leymann, Marie Salm, and Karoline Wild. Qprov: A provenance system for quantum computing. *IET Quantum Communication*, 2(4):171–181, 2021.

[66] Benjamin Weder, Johanna Barzen, Frank Leymann, and Michael Zimmermann. Hybrid Quantum Applications Need Two Orchestrations in Superposition: A Software Architecture Perspective. In *Proceedings of the 18^IEEE International Conference on Web Services (ICWS 2021)*, pages 1–13. IEEE, 2021.

[67] Manuela Weigold, Johanna Barzen, Frank Leymann, and Daniel Vietz. Patterns for hybrid quantum algorithms. In *Service-Oriented Computing: 15th Symposium and Summer School, SummerSOC 2021, Virtual Event, September 13–17, 2021, Proceedings 15*, pages 34–51. Springer, 2021.

[68] Ross Wightman. Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019.

[69] Karoline Wild, Uwe Breitenbücher, Lukas Harzenetter, Frank Leymann, Daniel Vietz, and Michael Zimmermann. Tosca4qc: two modeling styles for tosca to automate the deployment and orchestration of quantum applications. In *2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC)*, pages 125–134. IEEE, 2020.

[70] Michael Wurster, Uwe Breitenbücher, Kálmán Képes, Frank Leymann, and Vladimir Yussupov. Modeling and automated deployment of serverless applications using tosca. In *2018 IEEE 11th conference on service-oriented computing and applications (SOCA)*, pages 73–80. IEEE, 2018.

[71] Zhuoran Zhao, Kamyar Mirzazad Barijough, and Andreas Gerstlauer. Deepthings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2348–2359, 2018.