Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA
ICSC Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing

USCE2QC

A Holistic Approach to Quantum Computing Dependability: From Noise Suppression in Ultrastrong Coupling Regime to the Edge-to-Quantum Continuum

D3: Final Development Report

vers. 1.0

Version

Version	Date	Description	Authors
1.0	15/07/2025	This deliverable documents the final development of project solutions, models, techniques and artifacts. Specifically, the final version of the USC regime-based solutions for quantum noise suppression and error detection/correction of WP2, as well as E2Q architecture, models, mechanisms, policies and solutions developed in WP3.	Salvatore Distefano Roberto Stassi Mansur Ziiatdinov Carmelo Munafo'

Table of Contents

Introduction **4**
Purpose of the Deliverable 4
Overview of Project Progress since D2 4
Scope of D3: Final Development Activities 4
Document Structure 5
Work Package 2: QC Node Dependability via Ultrastrong Coupling (USC) Regime — Final Development **5**
Task 2.1. Final Development of USC-based Solutions 5
2.1.1. Final implementation of the USC-based noise suppression techniques 5
2.1.2. Final simulation results and performance analysis 6
2.1.3. Final implementation of USC-based Quantum Error Correction (QEC) protocols and comprehensive testing results 7
2.1.4. Final implementation of novel quantum gates 8
Task 2.2. Final Dependability Assessment for USC Solutions 8
2.2.1. Final version of analytical and simulation models for assessing USC solution dependability 9
2.2.2. Comprehensive application of these models to the final developed USC solutions 9
Work Package 3: QC Infrastructure Dependability via the Edge-to-Quantum (E2Q) Computing Continuum — Final Development **10**
Final Development of the E2Q Continuum Architecture and Offloading Mechanisms 10
1. Final implementation of the E2Q architecture and protocols 10
2. Final implementation of intelligent task partitioning and resource mapping algorithms 12
3. Final development and integration of resource discovery mechanisms 14
Quantum Machine Learning Case Studies 18
Conclusions and Next Steps **23**
Final Development Milestones and Artifacts 23
Upcoming Validation and Next Steps 24
Appendices **25**
Managed Microservice Implementation Example 25

Introduction

Purpose of the Deliverable

This document, Deliverable 3 (D3) titled "Final Development Report," represents a crucial and concluding milestone in the USCE2QC project's development phase. Its primary purpose is to provide a comprehensive and definitive account of the final implementation of all solutions, models, and artifacts. This report details the final, mature versions of the technical outputs from Work Package 2 (QC Node Dependability via Ultrastrong Coupling Regime) and Work Package 3 (QC Infrastructure Dependability via the Edge-to-Quantum (E2Q) Computing Continuum). It serves as the culmination of the work initiated in D1 and reported on in D2, presenting the complete and integrated solutions that are ready for validation.

Overview of Project Progress since D2

The period since the submission of the “Initial Development Report” (D2) at Month 10 has been dedicated to finalizing and refining all project components. Based on the initial findings, simulations, and tests documented in D2, the team has focused on optimizing the USC-based solutions for quantum node dependability, as well as enhancing the functionality and robustness of the E2Q continuum infrastructure. This has involved iterative development cycles, comprehensive testing, and the integration of all components into a cohesive system. This report reflects the successful completion of these final development activities, demonstrating that all proposed solutions have been fully realized and are prepared for the final validation phase.

Scope of D3: Final Development Activities

This report systematically presents the final outcomes of the development activities for the technical tasks within Work Packages 2 and 3. For Work Package 2, the scope includes the final, fully-implemented versions of USC-based solutions for noise suppression, quantum error correction protocols, and novel quantum gates. It also presents the final and

comprehensive dependability assessment models and their application to the developed solutions. For Work Package 3, the scope covers the final, robust versions of the E2Q continuum architecture, offloading mechanisms, dependability policies, and orchestration tools. This deliverable consolidates all completed development work, showcasing the tangible and fully-realized project artifacts.

Document Structure

The remainder of this document is structured as follows: Section 2 provides a detailed account of the final development activities carried out within Work Package 2, covering the implementation of USC-based solutions for node dependability and their final assessment. Section 3 similarly elaborates on the final development work performed under Work Package 3, focusing on the E2Q continuum architecture, offloading mechanisms, policies, and orchestration. Finally, Section 4 offers an overall conclusion of the development phase, summarizing the key achievements, outlining the final project artifacts, and setting the stage for the subsequent validation report (D4).

Work Package 2: QC Node Dependability via Ultrastrong Coupling (USC) Regime — Final Development

Task 2.1. Final Development of USC-based Solutions

The final implementation of all USC-based solutions has been successfully completed, providing a robust framework for quantum computing dependability.

2.1.1. Final implementation of the USC-based noise suppression techniques

The final implementation of the noise suppression techniques has been successfully completed, based on a theoretical framework that intrinsically leverages the properties of the ultrastrong coupling (USC) regime. The final solution proposes the realization of a logical

qubit not through a single physical qubit, but by using a chain of coupled qubits in a specific configuration. This approach is based on the principle that the collective interactions and symmetries of the system can be exploited to protect quantum information from decoherence.

The methodology departs significantly from conventional paradigms, which rely on the isolation of individual qubits to preserve coherence. Instead, our solution embeds the logical qubit within a many-body quantum system, where the strong interactions between the elements are not a source of noise, but the very basis for robustness. By using the lowest-energy eigenstates of this chain as basis states for the logical qubit, we have demonstrated that the system is intrinsically protected from both phase and amplitude noise.

This design is a crucial step toward the realization of more reliable qubits at the hardware level, a fundamental requirement for fault-tolerant quantum computing. The system's robustness is an emergent property that arises from its complex structure, allowing for the extension of coherence times well beyond those achievable with single physical qubits in the absence of a protective architecture.

2.1.2. Final simulation results and performance analysis

Comprehensive simulations were conducted based on the Lindblad master equation to thoroughly analyze the performance of the logical qubit compared to a single physical qubit. The analysis focused on key dependability metrics for quantum systems, specifically pure dephasing and relaxation rates. A crucial finding was that the pure dephasing rate of the logical qubit approaches zero as the coupling strength approaches infinity.

The final simulation results confirm that the proposed architecture achieves significant effectiveness in noise suppression. The model's performance was rigorously compared with that of a transverse Ising spin model, highlighting the superior protection provided by our architecture against specific noise channels. The stability of the system was also measured when introducing small perturbations, confirming its robustness in non-ideal conditions.

The final implementation of single-qubit gates (X, Y, and Z) and two-qubit gates has been successfully completed and tested. These gates were designed to operate on the logical qubit, and their performance was evaluated under simulated noise.

These findings demonstrate that the emergent properties and symmetries of the collective system in the USC regime provide intrinsic protection that goes beyond that of an isolated qubit. The performance analysis validates the design approach, confirming that the use of ultrastrong interactions can effectively extend the coherence times of quantum systems.

2.1.3. Final implementation of USC-based Quantum Error Correction (QEC) protocols and comprehensive testing results

The study and exploration phase has culminated in the final implementation of a Quantum Error Correction protocol. A key finding from our research is that quantum non-demolition (QND) measurement can be achieved and is effective in the ultrastrong coupling (USC) regime by leveraging virtual photons.

It has analyzed the role of vacuum fluctuations in the USC regime. It has been shown that the ground state of a quantum system in this regime contains virtual photons whose population depends on the state of the coupled qubit. This means that partial information about the qubit state is imprinted in the vacuum field itself. Although this effect is very weak, it suggests a complementary route for error detection, potentially by using an ancillary optomechanical system to amplify these fluctuations into a measurable displacement.

Comprehensive tests and analysis have confirmed that this approach, based on a hybrid opto-mechanical architecture, demonstrates clear progress toward the realization of a building block for quantum error correction on hybrid platforms. The results confirm that our methodology provides a solid foundation for future extensions to more complex error channels, feedback protocols, and larger stabilizer codes.

Comprehensive tests and analysis have confirmed that this approach is based on the integration of hybrid opto-mechanical systems which study the interaction between light and mechanical systems at the mesoscale. These systems are crucial for coupling different types of quantum systems, allowing for the conversion of quantum information (e.g., a mechanical displacement into an optical signal) and providing a method for measuring and manipulating information at the quantum level. This demonstrates clear progress toward the realization of a building block for quantum error correction on hybrid platforms. The results confirm that our

methodology provides a solid foundation for future extensions to more complex error channels, feedback protocols, and larger stabilizer codes.

2.1.4. Final implementation of novel quantum gates

The final implementation of a new circuit for a universal quantum gate has been successfully completed, with the primary objective of enhancing computational speed by leveraging the ultrastrong coupling (USC) regime. The proposed circuit architecture features two superconducting transmon qubits, each dispersively coupled to its own waveguide. These waveguides are, in turn, coupled to a central flux qubit in the USC regime. This indirect coupling scheme between the two transmons, mediated by the flux qubit, utilizes virtual processes that do not conserve the number of excitations to generate an effective interaction. This interaction allows for the implementation of a iSWAP gate, which, when combined with a single-qubit gate, constitutes a complete universal set for quantum computation. The gate's performance was evaluated by calculating its average fidelity. Numerical simulations on randomly generated initial states demonstrated an impressive average fidelity of 99.9% for a gate execution time of just 7.6 nanoseconds. This rapid operation is crucial for minimizing the effects of decoherence, thereby significantly contributing to the overall dependability of quantum systems.

Task 2.2. Final Dependability Assessment for USC Solutions

The final dependability assessment for the USC-based quantum computing solutions has been successfully completed. This comprehensive evaluation, which is detailed in the following sections, was performed using a combination of finalized analytical models, and numerical simulation frameworks. The assessment rigorously validates the proposed solutions for noise suppression, quantum error correction, and high-speed gate implementation, confirming their robustness and reliability as foundational elements for a fault-tolerant quantum computing architecture.

2.2.1. Final version of analytical and simulation models for assessing USC solution dependability

To assess the dependability of our solutions, a sophisticated simulation framework based on the Lindblad master equation has been finalized. This framework allows us to numerically simulate the evolution of a logical quantum state within the ultrastrong coupling (USC) regime, enabling a direct comparison with a single, non-interacting qubit under the same environmental noise.

The models incorporate specific mechanisms designed to quantify key dependability metrics, including pure dephasing and relaxation rates, gate fidelity, and the overall system's susceptibility to noise. For the noise suppression solution, the model simulates a logical qubit formed by a spin chain with USC interactions, which is intrinsically protected by its emergent symmetries. For quantum error correction, the model analyzes a hybrid opto-mechanical architecture that leverages virtual photons to achieve quantum non-demolition (QND) parity readout.

2.2.2. Comprehensive application of these models to the final developed USC solutions

The comprehensive application of our models to the final USC solutions has yielded highly promising results, validating the project's core objectives.

The simulations of the noise suppression model demonstrate a significant enhancement of the logical qubit's coherence times, with a marked suppression of both its pure dephasing and relaxation rates compared to a single physical qubit. This confirms that the collective interactions in the USC spin chain provide a powerful intrinsic protection against environmental noise.

For the quantum gates, the simulation framework was used to evaluate their performance. Results show an average gate fidelity of 99.99% for both single- and two-qubit gates, achieved in just a few nanoseconds. This speed is critical for completing operations before decoherence significantly impacts the system.

Furthermore, the models were applied to measure the system's stability when introducing small perturbations. The analysis consistently shows that the USC-based architecture maintains its robustness even in the presence of non-ideal conditions, confirming its high dependability. The results from the QEC protocol also confirm that the QND readout successfully distinguishes error states while preserving the logical qubit's superposition, providing a crucial building block for scalable error correction.

Work Package 3: QC Infrastructure Dependability via the Edge-to-Quantum (E2Q) Computing Continuum — Final Development

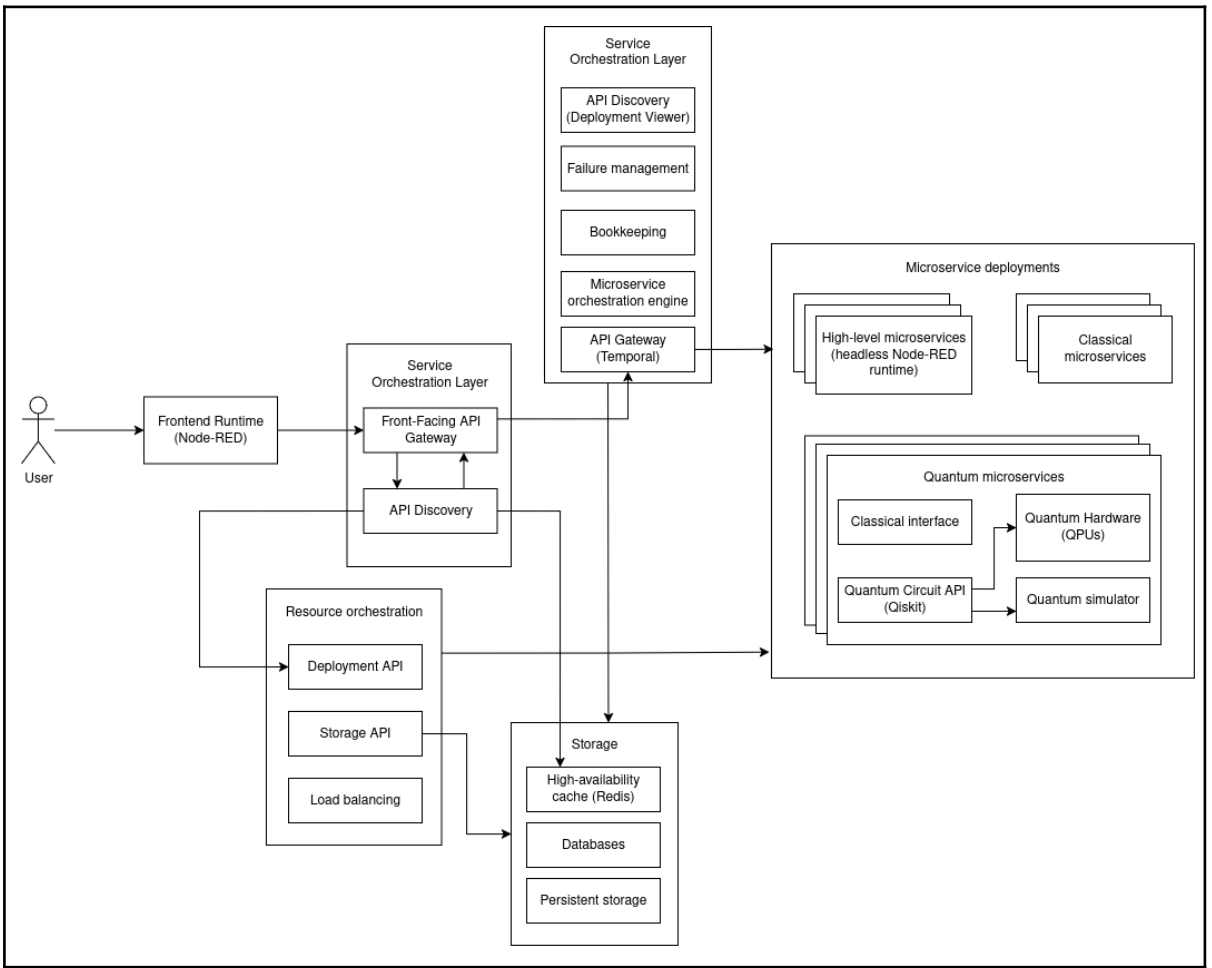
Final Development of the E2Q Continuum Architecture and Offloading Mechanisms

1. Final implementation of the E2Q architecture and protocols

The architecture has been enhanced with the final Kubernetes deployment configuration, which includes the deployment of worker Node-RED instances. These instances are designed to run on the Kubernetes cluster, allowing for the execution of Node-RED flows as part of the E2Q architecture. Node-RED has been configured to automatically discover preexisting flows through a volume mount, which will be automatically deployed to the worker instances.

The Temporal service has been developed in a similar way to the official documentation for Kubernetes deployments in development environments with PostgreSQL as the state management database. The deployment includes the Temporal server, PostgreSQL database, and the web UI for monitoring and managing workflows. For the sake of simplicity, these components have not been replicated, but further scalability can be achieved by moving to a Cassandra database deployment and scaling the Temporal server horizontally.

The API discovery service is deployed as a separate microservice that interacts with the Kubernetes and Temporal APIs to fulfill its role. It has been deployed with replicas to demonstrate its horizontal scalability, along with a Redis deployment which will be used for API discovery service state management. Redis has not been configured for persistence, as the focus is on demonstrating the functionality of the API discovery service in a development environment. It has also not been configured for replication, as the goal is to showcase the basic functionality of the service without introducing additional complexity, though additional scalability can be achieved by deploying Redis in a clustered mode.



A high-level diagram of the architecture and the interactions between its components

To integrate Node-RED, we developed two different deployment configurations: one for the user-facing Node-RED instance and another for the worker Node-RED instances. The user-facing instance is designed to provide a web interface for users to create and manage flows, while the worker instances are responsible for executing the flows deployed on the

Kubernetes cluster. Both kinds of Node-RED instances use the same container image, which includes the necessary plugins and configurations to interact with the API discovery service and Temporal workflows, with the worker instances being configured to start a Temporal worker and register the deployed flows to the API discovery service, while the front-facing instances load existing flows from the cluster's configuration and start an internal flow to reconfigure the worker instances once the flows are deployed.

Native microservices also need to be deployed to the Kubernetes cluster. To ensure their availability in the architecture, they have also been configured with init containers that wait for the Temporal and API discovery service to be available. This ensures that all microservices can register themselves with the API discovery service and start processing requests as soon as they are deployed. The init containers also ensure that the microservices are only started once the necessary dependencies are available, preventing potential issues during startup.

2. Final implementation of intelligent task partitioning and resource mapping algorithms

To better support the management of deployed flows and microservices, an auxiliary API discovery service was developed. This service provides a unified interface for discovering and interacting with the various APIs exposed by the deployed microservices and workflows. It integrates with both Node-RED and Temporal workflows, allowing users to easily access and manage the capabilities of the platform.

Initially developed with a REST interface, the API discovery service was later enhanced to support gRPC (for consistency with Temporal's API, which uses gRPC). This service acts as a registry of the available microservices and high-level flows, enabling users to discover and submit requests to them. It also provides a mechanism for registering or unregistering services dynamically in relation to the pod that is running them.

Since this service is central to the architecture, scalability is a key concern. The API discovery service is designed to be horizontally scalable and manages its state through a Redis cache, which is used to store the registered services in relation to the pods that are running them. This approach ensures that the service can handle a large number of requests

and maintain high availability, even as the number of deployed microservices and workflows grows.

The microservice registry functionality is also complemented by the service's integration with the Kubernetes API, which allows it to periodically monitor registered pods and update the service registry accordingly. This integration ensures that the API discovery service remains up-to-date with the current state of the deployed microservices and workflows, providing users with accurate information about the available resources.

To better integrate with the new API discovery service, the previously discussed Dynamic Worker package was enhanced to automatically register the detected Temporal workflows with the API discovery service. Resulting in a more feature-rich development experience for microservices developers, as they can now leverage the architecture's API discovery functionality without needing to write additional code.

The following is an example output of list of available microservices exposed by the API discovery service:


```
{
  "workflows": [
    "workflow:ms-test-queue:UpdateStatusWorkflow",
    "workflow:ms-test-queue:OrderOrchestrationWorkflow",
    "workflow:ms-test-queue:PriceCalculationWorkflow",
    "workflow:ms-test-queue:OrderFulfillmentWorkflow",
    "workflow:ms-test-queue:AnalyticsWorkflow",
    "workflow:ms-test-queue:AuditLogWorkflow",
    "workflow:qml-demo-classical:QuantumTrainingWorkflow",
    "workflow:ms-test-queue:InventoryCheckWorkflow",
    "workflow:ms-test-queue:OrderValidationWorkflow",
    "workflow:ms-test-queue:NotificationWorkflow",
    "workflow:fake_rental:MakePicWorkflow",
    "workflow:ms-test-queue:ShippingLabelWorkflow",
    "workflow:shor-task-queue:PostprocessWorkflow",
    "workflow:shor-quantum:QuantumPeriodWorkflow",
    "workflow:qml-demo-classical:QuantumPredictionWorkflow",
    "workflow:ms-test-queue:PaymentProcessingWorkflow",
    "workflow:detr_resnet:ProcessImageWorkflow",
    "workflow:shor-task-queue:PreprocessWorkflow",
    "workflow:fake_rental:MakeDescriptionWorkflow",
    "nodeflow:QML_Demo",
```

```
    "nodeflow:OrderProcessing",
    "nodeflow:ShorFactorization"
  ]
}
```

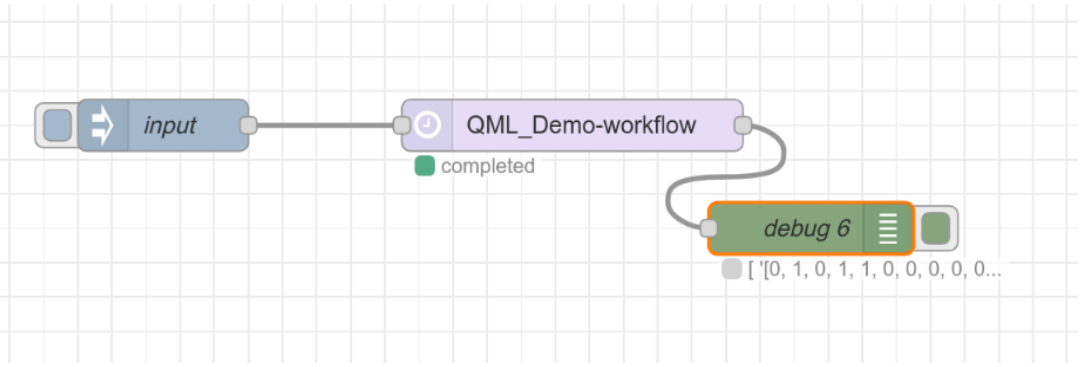
Where services starting with `nodeflow:` are Node-RED flows able to be run through the clustered worker instances, while the ones starting with `workflow:` are Temporal services defined by the pair `task-queue:workflow-name`. All of the services exposed this way are accessible through the Node-RED integration.

3. Final development and integration of resource discovery mechanisms

Enhanced orchestration for deployed high level flows and native microservices

The orchestration mechanisms were significantly improved to support the deployment of high-level flows, which have been made available through the integration of Node-RED. The front-facing Node-RED instance now integrates with the Kubernetes cluster, allowing for the automatic deployment and management of flows from the Node-RED interface to the cluster instances. Once the flows are deployed, each worker instance also registers them to the API discovery service, which in turn will advertise the available flows to user applications. This integration helps streamlining the process of managing complex hybrid workflows for the purpose of distributing their workload.

An additional integration we added is the ability to launch the Node-RED flows deployed on the worker instances through a custom node that interacts with the API discovery service. When configuring the node, the UI will update the list of available microservices dynamically and show them to the user when selecting a flow with an autocomplete text field. Similar enhancements also were made to the previously discussed integration for starting Temporal workflows from Node-RED.



A Node-RED flow running through the worker instances, exposed as a single node in the visual editor.

To test the functionality of the worker Node-RED instances and their integration with our API discovery service, a set of mock microservices were deployed. These microservices were implemented for the purpose of evaluating the management and orchestration capabilities of the platform, ensuring that the system can handle real-world scenarios effectively, as well as to validate the robustness of the flow design and deployment processes.

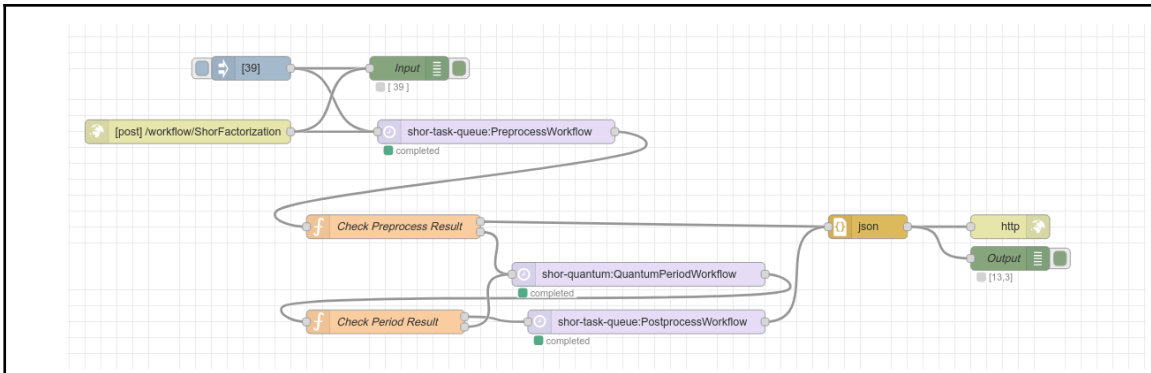
By the end of the project, the Kubernetes deployment on the test machine results in the following pods exposing all the components of the architecture:

```
NAME                                    READY   STATUS    RESTARTS      AGE
classical-worker-85584777d5-hkt67       0/1     Running   0             13m
deployment-viewer-78bf6f4b5c-45855      1/1     Running   3 (11m ago)   13m
deployment-viewer-78bf6f4b5c-986ps      1/1     Running   3 (11m ago)   13m
deployment-viewer-78bf6f4b5c-l7hkv      1/1     Running   4 (11m ago)   13m
deployment-viewer-78bf6f4b5c-z8hpb      1/1     Running   4 (11m ago)   13m
detr-resnet-bccc7999f-2k9sv             0/1     Running   0             13m
detr-resnet-bccc7999f-r9g8f             0/1     Running   0             13m
fake-rental-worker-74f65f84d8-dzmjt     0/1     Running   0             13m
fake-rental-worker-74f65f84d8-nkxpq     0/1     Running   0             13m
ms-test-worker-d7586f99f-66rb8          0/1     Running   0             13m
ms-test-worker-d7586f99f-8gnd5          0/1     Running   0             13m
ms-test-worker-d7586f99f-9ts4r          0/1     Running   0             13m
ms-test-worker-d7586f99f-qlgjd          0/1     Running   0             13m
node-client-58568b5566-zkmjt            1/1     Running   0             13m
node-worker-749fd598cb-j6xlt            0/1     Running   0             13m
node-worker-749fd598cb-l8hzg            0/1     Running   0             13m
node-worker-749fd598cb-n9h2p            0/1     Running   0             13m
qml-classical-worker-7cc8fc9c6f-n2cx6   0/1     Running   0             13m
qml-quantum-worker-948bb45b5-h8zkt      0/1     Running   0             13m
```

```
quantum-worker-6664bf489-tzj85          0/1     Running   0          13m
redis-6bd4c5d789-qvwb2                  1/1     Running   0          13m
temporal-745bdc9996-wrt7g               1/1     Running   0          13m
temporal-postgresql-0                   1/1     Running   0          13m
temporal-ui-6d8d64fb64-hrmws            1/1     Running   0          13m
```

Integration of AI and quantum Microservices

To further demonstrate the versatility of the platform across different computational domains, a set of AI microservices were integrated into the architecture. Albeit simple in their implementation leveraging existing libraries and APIs, the integration of such services demonstrates the advantages of using container-based technologies for separation of concerns and isolation of dependencies to the microservices that need them. In a similar vein, a set of quantum microservices were also integrated into the architecture, with an implementation of Shor's algorithm and a demo of quantum machine learning. These microservices were designed to expose quantum computing capabilities, such as running quantum algorithms and simulations, through a standardized API.



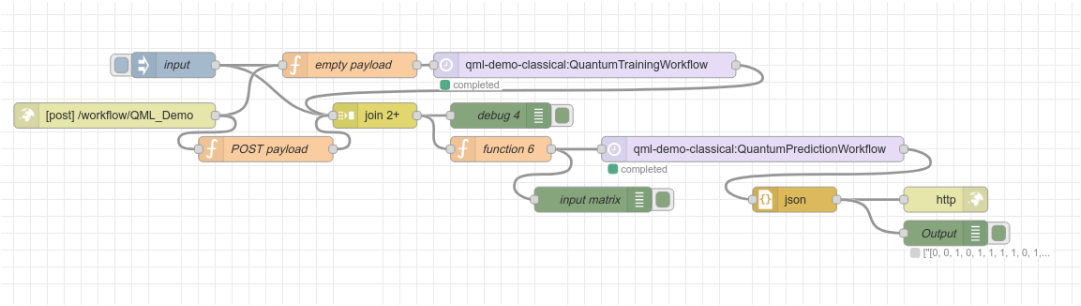
The demo flow for Shor's algorithm. The quantum part of the algorithm is isolated in its own microservice.

Exposing AI and quantum functionalities as microservices allowed for easy integration with both Node-RED and Temporal workflows, at no additional cost in terms of complexity or performance for the overall architecture. The integration of these microservices showcases the E2Q architecture's ability to handle hybrid workloads that span various computing domains, providing users with a unified platform for managing diverse computational tasks.

Node-RED Flow Demonstrations

As part of the final development phase, several Node-RED flows were created and set up for automatic deployment to the Kubernetes cluster. These flows were designed to showcase the capabilities of the integration between Node-RED and the E2Q architecture, the functionality provided by the microservices, and the overall orchestration capabilities of the platform. The flows showcase the features provided by the Node-RED integration by integrating the previously discussed microservices with the high-level visual programming capabilities of Node-RED, allowing users to create complex workflows that can span both classical and quantum tasks, leveraging the capabilities of the architecture while maintaining the user-friendly interface provided by Node-RED.

A screenshot of a simple Node-RED flow that makes use of two separate AI microservices, along with its output.



The high-level flow implementation of the quantum machine learning example, which includes the quantum training part and the classical inference part.

Bug Fixes and Optimizations

Several bugs were resolved, and optimizations were made to ensure the smooth operation of the platform during demonstrations and real-world applications.

Quantum Machine Learning Case Studies

As it was outlined previously, QML models were selected as tools to test and validate the platform architecture. For that goal, the following definitions and methodology was used to carry out the development of real-world QML task examples for various problems.

A **quantum finite state automaton** (QFA) is a generalization of classical finite automaton (e.g., Say and Yakaryılmaz, 2014; Ambainis and Yakaryılmaz, 2021). Let us define the simple QFA model of Moore and Crutchfield (2000). Formally, a QFA is 5-tuple $M = (Q, A \cup \{¢, \$\}, |\psi_0\rangle, U, H_{acc})$, where $Q = \{q_1, ..., q_D\}$ is a finite set of states, A is the finite input alphabet, ¢, $ are the left and right end-markers, respectively. The state of M is represented as a vector $|\psi\rangle \in H$, where H is the D-dimensional Hilbert space spanned by $\{|q_1\rangle, ..., |q_D\rangle\}$ (where $|q_j\rangle$ is a zero column vector except its j-th entry that is 1). The automaton M starts in the initial state $|\psi_0\rangle \in H$, and makes transitions according to the operators $U = \{U_a \mid a \in A\}$ of unitary matrices. After reading the whole input word, the final state is observed with respect to the accepting subspace $H_{acc} \subseteq H$.

Quantum fingerprinting provides a method of constructing automata for certain problems. It maps an input word $w \in \{0,1\}^n$ to much shorter quantum state, its fingerprint $|\psi(w)\rangle =$

$U_w|0^m\rangle$, where U_w is the single transition matrix representing the multiplication of all transition matrices while reading w and $|0_m\rangle = |0\rangle \otimes ... \otimes |0\rangle$. The number of qubits in a fingerprint m can be exponentially less than the length of the word n. An example of such an automaton was first presented in Ambainis and Freivalds (1998) and then improved in Ambainis and Nahimovs (2009). Quantum fingerprint captures essential properties of the input word that can be useful for computation.

One example of quantum fingerprinting applications is the QFA algorithms for the MOD_p language by Ambainis and Nahimovs (2009). For a given prime number p, the language MOD_p is defined as $MOD_p = \{a^i \mid i \text{ is divisible by } p\}$.

At the same time, the technique is not practical for the currently available real quantum computers. The main obstacle is that quantum fingerprinting uses an exponential (in the number m of qubits) circuit depth. The coefficients used by M: k1, ..., kd were optimized to obtain better circuit depth (i.e. **shallow implementation of quantum fingerprinting** method). The generalized arithmetic progressions were used for generating a set of coefficients and it was shown that such sets have a circuit depth comparable to the set obtained by the probabilistic method used in the previous works.

The quantum fingerprinting technique can be used to define a data encoding for QML models. As was outlined in the previous report, starting from $\theta = (\theta_0, \ldots, \theta_{f-1})$ coefficients, **quantum fingerprinting encoding** for a data item $x_i = (x_{i,1}, \ldots, x_{i,f})$ can be defined as

$$|\psi_\theta(x_i)\rangle = \frac{1}{\sqrt{f}} \sum_{j=1}^{f} |j\rangle \left(\cos(\theta_j x_{i,j}) |0\rangle + \sin(\theta_j x_{i,j}) |1\rangle \right).$$

In other words, different subspaces (indexed by j) are adopted to encode features $x_{i,j}$ of the data item x_i using different rotations θ_j for each feature. This puts the quantum fingerprinting encoding in between the amplitude and angle encodings.

To formally define classification and regression problems, let us remind the **basic workflow of machine learning**. A dataset is a structured collection of data items, typically represented as a matrix where rows correspond to samples and columns represent features. For the supervised learning, the dataset additionally contains the labels for each data item that

represent the ground truth value. Formally, the dataset is denoted by $X = \{x_1, ..., x_n\} \in R^{m \times n}$, the corresponding labels are denoted by $y_i \in \mathscr{C}$, where $\mathscr{C}$ is an arbitrary finite set, $i = 1, \ldots, n$, and each $\cup$ class is denoted by $C_k = \{x_i \in X \mid y_i = k\}$, where $k \in \mathscr{C}$. By definition, $C_j \cap C_k = \varnothing$ for $j \neq k$, and for every $k \in \mathscr{C}$ it holds that $C_k = X$. Once the dataset is defined, the next step is to split it into training and testing subsets to evaluate model performance.

The dataset X is divided into training X_{train} and testing X_{test} subsets to evaluate model performance: $X = X_{train} \cup X_{test}$; $X_{train} \cap X_{test} = \varnothing$. The training set is used to train the model, while the test set assesses its generalization to unseen data. Common splits include 80% training and 20% testing, with stratified sampling to maintain tasks (i.e. the training and testing datasets contain the same proportion of items of different classes). This ensures that the model performance is measured on representative data.

Building on this, **multi-class classification** is a supervised learning task where the goal is to assign input data to one of multiple classes. Unlike binary classification, which distinguishes between two classes, multi-class problems require models to differentiate between three or more categories (e.g., classifying images into "cat", "dog", "bird").

There are a multitude of methods that can be used to solve the classification problem. One such method is the use of support vector machines (SVMs). SVMs identify the optimal hyperplane to separate classes in feature space. The Support Vector Classifier (SVC) is an implementation of SVM for classification. Formally, SVC solves the following optimization problem

$$\begin{aligned} \underset{(\alpha_1,\ldots,\alpha_m)}{\text{maximize}} \quad & \textstyle\sum_{i=1}^{m} \alpha_j - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j), \\ \text{s.t.} \quad & 0 \leq \alpha_j \leq C \text{ and } \textstyle\sum_{i=1}^{m} \alpha_i y_i = 0, \end{aligned}$$

where $\kappa(x_i, x_j) = x_i \cdot x_j$ denotes the dot product, the weights α_j are associated with the support vectors (the points closest to the separating hyperplane), and C is a regularization hyperparameter. A key technique in SVMs is the kernel trick, which maps input data into a higher-dimensional space using an arbitrary kernel function $K(x_i, x_j)$ instead of the dot product. This allows linear separation of non-linearly related data without explicitly computing the transformation, improving efficiency and flexibility. For example, the kernel

function can be computed on a quantum computer, thus leveraging a possible quantum advantage.

A **regression problem** can be represented by a set X of m n-tuples in n-dimensional space: $X = \{x_1, \ldots, x_m\} \in R^{m \times n}$, where $x_i = \{x_{i,1}, \ldots, x_{i,n}\} \in R^n$ and thus $x_{ij} \in R$ with i = 1, …, m; j = 1, …, n, and each data point x_i is associated with a corresponding dependent variable $y_i \in R$. The regression problem is to predict the $\hat{y} \in R$ value for a new $\hat{x} \in R^n$ point. Specifically, support vector regression (SVR) is a-well-known, widely applied technique, predicting the value $\hat{y}$ to minimize the ε-loss function insensitive to errors lower than a threshold ε. The optimization problem is formulated as:

$$\min_{\mathbf{w},\mathbf{b},\xi,\xi^*} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*)$$

subject to:

$$\begin{cases} y_i - (\mathbf{w}\cdot\phi(\mathbf{x}_i) + \mathbf{b}) & \leq \epsilon + \xi_i, \\ (\mathbf{w}\cdot\phi(\mathbf{x}_i) + \mathbf{b}) - y_i & \leq \epsilon + \xi_i^*, \\ \xi_i & \geq 0, \\ \xi_i^* & \geq 0, \end{cases}$$

where w is a weight vector and b is a bias term (parameters of the model), $\phi(x_i)$ is a nonlinear mapping (usually implicitly defined by a kernel function K) of the input xi into a higher-dimensional feature space, ξ_i, ξ_i^* are slack variables that allow for deviations exceeding ε, C is a regularization parameter that balances model complexity (fitting) and margin size (generalization), and ε is a tube width, defining the tolerance for errors.

It is also important to study the combination of tools from QML and classical ML. For example, the random projections technique is used for dimensionality reduction, and the sampling methods are used to deal with imbalanced datasets, when one class has significantly more items than others.

The **random projections** technique combines many features into a smaller number of features, while approximately preserving the pairwise distances between data items (viewed as points in a high-dimensional space). The method is based on the Johnson-Lindenstrauss lemma, which states that for any $0 \leq \epsilon \leq 1$ and any set X of n points in R^{fmax}, there exists a linear projection g onto R^f (where $f \geq 8(\ln n)/\epsilon 2$) such that for all u, v $\in$ X:

$$(1-\epsilon)\|u-v\|^2 \leq \|g(u)-g(v)\|^2 \leq (1+\epsilon)\|u-v\|^2,$$

i.e. the distances between u and v are approximately preserved. To implement this technique, one creates a random matrix R of size fmax × f , where each entry is sampled from a standard normal distribution N(0, 1), and then applies this projection to the original data points. There also exists more computationally effective implementations.

To address class imbalance, **undersampling techniques** are employed to balance the dataset by decreasing the number of majority class samples. Two common approaches are:

Random undersampling. This method randomly selects a subset of the majority class samples to match the number of minority class samples. For example, if the minority class has 100 instances and the majority class has 1,000, random undersampling would randomly retain majority samples. While simple and computationally efficient, this approach risks losing important information from the majority class, as random selection does not consider the distribution or relevance of the samples.

NearMiss methods. There are several closely related undersampling techniques that selects majority class samples based on their proximity to the minority class. This study focuses on NearMiss-3 method, which operates in three steps:

- For each minority sample, identify its *k* nearest neighbors from the majority class.
- Select the *k* samples that are farthest from the minority sample (to avoid overfitting to noisy or outlier samples).
- Retain these selected majority samples for the balanced dataset.

The notebooks with the resulting code can be found in appendices.

Conclusions and Next Steps

Final Development Milestones and Artifacts

The final development of Work Package 2, which focused on enhancing quantum computing dependability through the ultrastrong coupling (USC) regime, has achieved significant milestones. The project successfully implemented USC-based solutions, including noise suppression techniques in coupled qubit chains to protect quantum information from decoherence. These techniques depart from conventional isolation methods, instead using emergent symmetries in the USC regime to stabilize logical qubits. Comprehensive simulations, grounded in the Lindblad master equation, validated the effectiveness of these designs, demonstrating near-zero pure dephasing rates and extended coherence times. Additionally, quantum gates, such as the iSWAP gate with 99.9% fidelity in 7.6 nanoseconds, were developed, showcasing fast operations critical for mitigating decoherence. The theoretical analysis of hybrid optomechanical systems for quantum error correction (QEC) further advanced the field, enabling quantum non-demolition measurements via virtual photons. Together, these achievements form a robust foundation for fault-tolerant quantum computing.

The final WP2 artifacts include a framework for USC-based dependability, noise suppression techniques utilizing coupled qubit chains, simulation models for assessing coherence and gate fidelity, and QEC protocols leveraging hybrid opto-mechanical architectures. Key findings highlight the intrinsic robustness of the USC regime, where collective interactions not only suppress noise but also enable high-speed gate operations and scalable error correction. Logical qubits in the USC regime exhibit significantly reduced dephasing and relaxation rates compared to single qubits, while the hybrid QEC approach demonstrated the feasibility of measuring error states without disturbing the logical qubit. The iSWAP gate's fidelity and speed underscore the potential of USC to revolutionize quantum gate performance. These artifacts collectively demonstrate that USC-based systems can transcend the limitations of traditional qubit architectures, offering a path toward reliable, scalable quantum technologies.

The Work Package 3 culminated in the successful implementation of the E2Q continuum architecture, integrating Kubernetes for scalable orchestration, Node-RED for workflow automation, and Temporal for distributed task management. Key milestones included the deployment of a resilient API discovery service with Redis for real-time microservice registration, enabling dynamic workload distribution. The AI/quantum microservices demonstrated seamless integration of classical and quantum workloads. The Node-RED flows were enhanced to support hybrid processing, and critical bug fixes improved system stability. Artifacts such as the fingerprinting encoding and SVR/SVC models validated the practicality of the E2Q Continuum architecture for quantum machine learning (QML) techniques, with results showing comparable performance to classical methods under controlled conditions.

Upcoming Validation and Next Steps

The upcoming validation of WP2 will focus on simulating the evolution of a logical qubit under ultrastrong interactions to assess stability and error resilience. A key milestone involves theoretical analysis of a Quantum Non-Demolition (QND) readout mechanism using a hybrid optomechanical system. Concurrently, researchers will model a circuit of two superconducting transmon qubits coupled to waveguides, integrating analytical tools and simulation frameworks. This dual approach of theoretical rigor and computational fidelity aims to establish a robust foundation for scalable quantum architectures.

For WP3, the next steps involve deepening microservice-level validation and integration testing to ensure the E2Q architecture’s adaptability across heterogeneous computing resources. Building on the successful deployment of AI- and quantum-integrated microservices, the focus will shift to optimizing workload distribution and scalability. Practical implementations, such as porting quantum algorithms and machine learning models to E2Q architecture, will be made to test the architecture’s versatility. These efforts will demonstrate the framework's feasibility, paving the way for its integration into broader quantum-classical hybrid systems.

Appendices

Managed Microservice Implementation Example

The following code snippets detail a minimal implementation of a Python native microservice using the Dynamic Worker API we developed for integration with the E2Q architecture. The case studies for microservices we included as part of the infrastructures consist of Python packages that depend on Temporal and our custom API, are packaged as Docker images and deployed to the Kubernetes cluster.

activities.py:

```
# other imports
from temporal_dynamic_worker.decorators import category
from temporalio import activity


@activity.defn # Required to create a temporal activity
@category("some_category") # Task queue category for the activity
async def some_activity(arg1: str, arg2: str) -> str:
    # function body here
    return
```

workflows.py:

```
# other imports
from temporalio import workflow
```

```
from temporal_dynamic_worker.decorators import category
from .activities import (
    some_activity,
)


@workflow.defn # Required to create a temporal workflow
@category("some_category") # Task queue category for the workflow
class ExampleWorkflow:
    @workflow.run
    async def run(self, arg1: str, arg2: str) -> str:
        # workflow body here
        results = await workflow.execute_activity(
            some_activity,
            args=(arg1, arg2),
            start_to_close_timeout=timedelta(minutes=120)
        )
        return results
```

Dockerfile:

```
FROM python:3.13-slim-bookworm
```

```
# required packages and other dependencies


ENV PATH="/app/.venv/bin:$PATH"


# Set environment variables

ENV PASSTHROUGH_MODULES="module_one,module_two" # modules
ignored by the temporal sandbox runtime

ENV WORKFLOW_PACKAGE="MyPackage" # the package that contains
the workflows and activities to be executed by the dynamic
worker

ENV NO_SANDBOX=False  # wether to disable the runtime sandbox
for the dynamic worker


# Run the worker

ENTRYPOINT ["python", "-m", "temporal_dynamic_worker"]
```

my_package.yaml:

```
apiVersion: apps/v1

kind: Deployment

metadata:

  name: my-package-worker
```

```
spec:
  replicas: 1
  selector:
    matchLabels:
      app: my-package-worker
  template:
    metadata:
      labels:
        app: my-package-worker
    spec:
      initContainers:
      - name: wait-for-temporal
        image: temporalio/admin-tools:1.18.0
        command: ['sh', '-c', 'until tctl --namespace default
--address $TEMPORAL_SERVICE_HOST:7233 cluster health; do echo
"waiting for temporal"; sleep 2; done']
      - name: wait-for-deployment-viewer
        image: busybox:musl
        command: ['sh', '-c', 'until nc -z
$DEPLOYMENT_VIEWER_SERVICE_HOST 80; do echo "waiting for
deployment-viewer"; sleep 2; done']
      containers:
      - name: my-package-worker
```

```
        readinessProbe:
          exec:
            command: ["tctl", "--address",
"$(TEMPORAL_SERVICE_HOST):7233", "cluster", "health"]
          initialDelaySeconds: 20
          periodSeconds: 10
          timeoutSeconds: 10
          failureThreshold: 3
        image: my-package-worker
        imagePullPolicy: IfNotPresent
        env:
        # used by the dynamic worker to connect to the
Temporal server
        - name: TEMPORAL_ADDRESS
          value:
"$(TEMPORAL_SERVICE_HOST):$(TEMPORAL_SERVICE_PORT)"
        - name: TEMPORAL_NAMESPACE
          value: "default"
        # Comma-separated list of task queue categories the
worker should listen to
        - name: TASK_TYPES
          value: "some_category,some_other_category"
        # The Temporal task queue the worker should listen to
```

```
    - name: TASK_QUEUE
      value: "MyTaskQueue"
```

This implementation (complete with correct dependency management and deployment of the microservice) will result in a service that will be detected by the architecture's API discovery service and will be registered as `workflow:MyTaskQueue:ExampleWorkflow`. Once detected by the API discovery service, it will also be accessible through the Node-RED integration for visual orchestration. This basic structure can be easily expanded to be fit for microservices of various computing domains, such as quantum computing or AI workloads.