Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

QUASAR: A Dynamic Software Product Line Architecture for Runtime Self-Adaptation in Hybrid Quantum–Classical Software Systems

Samuel Sepúlveda

samuel.sepulveda@ufrontera.cl

Universidad de La Frontera

Ricardo Pérez-Castillo

University of Castilla-La Mancha

Jesús Tapia-Martin

Universidad de La Frontera

Sebastián Candia

Universidad de La Frontera

Claudio Navarro

Universidad de La Frontera

Ania Cravero

Universidad de La Frontera

Research Article

Keywords:

Posted Date: November 17th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11565999/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

QUASAR: A Dynamic Software Product Line Architecture for Runtime Self-Adaptation in Hybrid Quantum–Classical Software Systems

Samuel Sepúlveda[1*], Ricardo Pérez-Castillo[2], Jesús Tapia-Martin[1], Sebastián Candia[1], Claudio Navarro[1], Ania Cravero[1]

[1*]Departamento de Ciencias de la Computación, QuantumLab-DCI, Universidad de La Frontera, Francisco Salazar 01145, Temuco, 4780000, La Araucanía, Chile.

[2]Faculty of Social Sciences & IT, University of Castilla-La Mancha, Avenida Real Fábrica de Sedas, Talavera de la Reina, 45600, Spain.

*Corresponding author(s). E-mail(s): samuel.sepulveda@ufrontera.cl;
Contributing authors: ricardo.pdelcastillo@uclm.es; j.tapia05@ufromail.cl; s.candia05@ufromail.cl; claudio.navarro@ufrontera.cl; ania.cravero@ufrontera.cl;

Abstract

Context: Hybrid quantum–classical software systems face extensive variability and dynamically changing operational conditions, including shifts in backend availability, queue latency, noise, cost, and provider constraints. These changes complicate the selection and maintenance of suitable configurations during operation.

Problem: Software Product Line approaches have demonstrated their usefulness for addressing quantum-aware variability, particularly by capturing configuration alternatives and dependencies. However, most existing approaches manage variability primarily at design time, offering limited support for assessing and enacting runtime reconfigurations in response to changing operational conditions.

Proposal: This paper proposes QUASAR (QUantum Adaptive Software ARchitecture), a Dynamic Software Product Line architecture for runtime self-adaptation in hybrid quantum–classical software systems. QUASAR integrates four artifacts into a MAPE-K feedback loop: a Runtime Decision Catalog, Runtime Information Catalog, Decision–Information Matrix, and Runtime Variability

Model. An LLM-supported reasoner generates candidate adaptations, while deterministic guardrails validate variability constraints, execution compatibility, and applicable policies before enactment.
Results: A proof-of-concept examines backend rebinding, execution-parameter adjustment, and hybrid quantum–classical capability activation or bypass. Across 42 controlled executions, 10 (23.8%) contained LLM-generated candidates violating implemented variability constraints; all were blocked before enactment. The prototype records auditable adaptation traces and reports 1.5–3 s adaptation latency, consistent with coarse-grained software orchestration.
Implications: QUASAR provides preliminary evidence that a bounded subset of the proposed architecture can combine model-based variability management, LLM-supported planning, and deterministic validation for runtime adaptation in hybrid quantum–classical software systems.

Keywords: Hybrid Quantum–Classical Software Systems, Quantum Software Engineering, Dynamic Software Product Lines, Runtime Self-Adaptation, Variability Management, Large Language Models.

1 Introduction

Quantum computing (QC) is increasingly shaping new forms of software-intensive systems in which quantum resources are integrated with classical components to address computationally demanding problems (Nielsen and Chuang 2010; Preskill 2018; Piattini et al. 2020; Zhao 2020; Kim et al. 2023; Murillo et al. 2025). In the near term, such systems are expected to remain predominantly hybrid, since quantum processors must be orchestrated together with classical services, data processing components, optimization routines, and execution infrastructures (Weder et al. 2021b, 2022; Leymann et al. 2020). This situation raises software engineering challenges that go beyond the implementation of quantum circuits, including architectural design, integration, variability management, execution control, and runtime operation (Gheorghe-Pop et al. 2020; Serrano et al. 2022; De Stefano et al. 2022; Murillo et al. 2025).

Hybrid quantum–classical software systems (HSS) involve several design and configuration decisions (Weder et al. 2021b; Serrano et al. 2022; Sepúlveda et al. 2025a). Engineers must decide whether functionality should be implemented classically, quantumly, or through a hybrid workflow; which quantum algorithm family should be used; which programming framework and execution backend are suitable; and which hardware-related constraints must be considered (Weder et al. 2022; Leymann et al. 2020; Gheorghe-Pop et al. 2020). These decisions introduce a high degree of variability that must be systematically managed throughout the development lifecycle (Clements and Northrop 2001; Pohl et al. 2005; Benavides et al. 2010; Sepúlveda et al. 2025b).

In previous work, we proposed SPL-HQC, a Software Product Line (SPL) approach to develop HSS (Sepúlveda et al. 2024; Sepúlveda et al. 2025b). This proposal includes a feature model (FM) and an associated process to represent and configure relevant variation points (e.g., functionality, quantum algorithms, programming languages, integration mechanisms, and hardware-based constraints). This approach contributes to

managing variability at design time and supports the derivation of HSS configurations in a more systematic and reusable manner (Clements and Northrop 2001; Pohl et al. 2005; Benavides et al. 2010).

However, HSS also faces runtime variability that cannot be fully resolved at design time since optimal decisions depend on conditions known only at runtime (Hallsteinsen et al. 2008; Aguayo and Sepúlveda 2022), especially in Noisy Intermediate-Scale Quantum (NISQ) settings (Preskill 2018). For example, the behavior of the quantum backends can change due to queue times, noise levels, calibration conditions, execution latency, cost, availability, and hardware-specific limitations (Gheorghe-Pop et al. 2020; Leymann et al. 2020; Hopf et al. 2026). Moreover, the suitability of a quantum, simulated, or classical alternative may depend on the size of the problem instance, the service-level objectives, the cost constraints, and the required fidelity or precision (Murali et al. 2019, 2022; Weder et al. 2021b; Shinde and Szefer 2026; Chundury et al. 2026). As a result, a configuration that is valid and suitable at design time can become suboptimal or infeasible during execution (Hallsteinsen et al. 2008; Aguayo and Sepúlveda 2022; Kephart and Chess 2003). This possibility creates the need to explicitly identify which decisions can be revisited at runtime, what observable information is required to support them, and which constraints must be checked before executing a reconfiguration.

Dynamic Software Product Lines (DSPLs) provide a natural foundation to address this challenge, as they support the reconfiguration of software variants at runtime (Hallsteinsen et al. 2008; Aguayo and Sepúlveda 2022). DSPL can be used in combination with a self-adaptive loop based on MAPE-K (Monitor-Analyze-Plan-Execute over a shared Knowledge base) (IBM Corporation 2006). Thus, DSPL concepts can be used to monitor execution conditions, analyze deviations from expected quality goals, plan alternative configurations, execute reconfigurations, and maintain runtime knowledge of the system and its environment (Kephart and Chess 2003; Hachicha et al. 2016; Oh et al. 2023). In the context of HSS, this means treating variability models not only as design artifacts but also as runtime specifications that guide adaptation (Hallsteinsen et al. 2008; Benavides et al. 2010; Aguayo and Sepúlveda 2022).

Recent advances in Large Language Models (LLMs) open up an additional opportunity for runtime adaptation (Wang et al. 2024). LLMs can help interpret heterogeneous contextual information, reason about trade-offs, and generate human-readable justifications for configuration decisions (Donakanti et al. 2024; Alves et al. 2026). However, a robust adaptation architecture should combine LLM-supported contextual reasoning with deterministic validation mechanisms derived from variability models and traceability links (Benavides et al. 2010; Czarnecki and Wasowski 2007; Kephart and Chess 2003).

This paper proposes QUASAR (QUantum Adaptive Software ARchitecture), a DSPL architecture for model-based runtime self-adaptation in HSS. It uses an explicit runtime variability model to delimit reconfiguration alternatives and deterministic constraint validation to check candidate adaptations before enactment. These artifacts are integrated with a MAPE-K loop, an LLM-supported contextual reasoner, deterministic guardrails, and Quantum DevOps-oriented operational integration.

The main contributions of this paper are as follows.

- An extension of SPL-HQC toward a DSPL-based runtime variability management approach for HSS;
- A set of decision-oriented artifacts, including a Runtime Decision Catalog, a Runtime Information Catalog, and a Decision–Information Matrix, that relate adaptation decisions to observable runtime information;
- A MAPE-K-based architecture that integrates runtime variability management with LLM-supported contextual reasoning for adaptive configuration support;
- An LLM-supported reasoning process with deterministic constraint validation;
- A preliminary evaluation based on selected runtime adaptation scenarios, examining feasibility, constraint validation, and adaptation traceability.

The remainder of the paper is structured as follows. Section 2 presents the background. Section 3 reviews related work and synthesizes research gaps. Section 4 introduces the QUASAR proposal and its artifacts. Section 5 presents the motivating and running scenario. Section 6 describes the prototype implementation. Section 7 reports the preliminary evaluation. The threats to validity are presented in Section 8. Section 9 discusses the findings, implications, and limitations. Finally, Section 10 concludes the paper and outlines future work.

2 Background

This section briefly introduces the conceptual foundations required to understand QUASAR: hybrid quantum–classical software systems, Software Product Lines and variability modeling, quantum variability, Dynamic Software Product Lines, self-adaptation and MAPE-K, LLM-supported reasoning, and Quantum DevOps.

Throughout this section, a simplified route-optimization application serves as a running illustration. Classical services may invoke an optional quantum optimization capability, whose circuits can execute on a QPU or a simulator. The example progressively illustrates hybrid execution, product-line variability, quantum-aware constraints, runtime rebinding, MAPE-K control, and LLM-supported reasoning subject to deterministic validation; it is conceptual and does not represent the evaluated prototype.

2.1 Hybrid Quantum–Classical Software Systems

At the software-architecture level, HSS involves classical application logic, classical–quantum integration mechanisms, and quantum execution backends (Leymann et al. 2020; Weder et al. 2022). Classical components prepare inputs and process results, while integration mechanisms coordinate requests to backends that execute quantum circuits on QPUs or simulators (Leymann 2019; Weder et al. 2022; Gheorghe-Pop et al. 2020). Figure 1 illustrates these functional roles and their interactions. The colors distinguish functional roles rather than physical deployment boundaries.

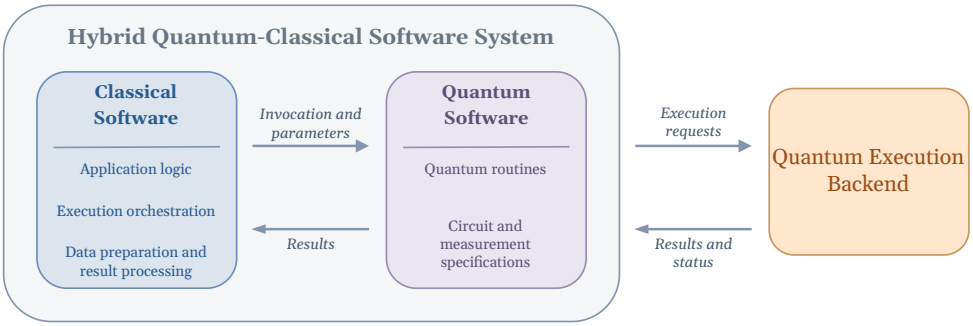


Fig. 1 High-level organization of an HSS and its functional components.

2.2 Software Product Lines and Variability Modeling

A Software Product Line (SPL) is a family of related software systems that share a managed set of common and variable features and are developed from a common set of reusable software assets (Clements and Northrop 2001; Pohl et al. 2005).

Variability models make explicit the configuration choices that distinguish product variants, while FMs represent these choices through mandatory, optional, alternative, and grouped features (Kang et al. 1990; Czarnecki and Wasowski 2007). Cross-tree constraints such as *requires* and *excludes* restrict the set of admissible configurations and provide a basis for automated reasoning, including consistency checking and valid product derivation (Czarnecki and Wasowski 2007; Benavides et al. 2010).

Continuing the route-optimization illustration, Figure 2 shows how an FM represents commonality, variability, and configuration constraints. The model is specified in the Universal Variability Language (UVL) (Benavides et al. 2025) and visualized using the FlamaPy IDE (Benitez et al. 2025). ClassicalServices is mandatory, whereas QuantumOptimization is optional. When selected, it requires exactly one ExecutionMode: QPU or Simulator. Two cross-tree constraints complement these hierarchical relationships. Assuming cloud-based access to the QPU, QPU requires CloudAccess; CloudAccess excludes LocalOnly, which denotes a configuration restricted to local execution. Together, these relationships delimit the admissible variants of the illustrative product line.

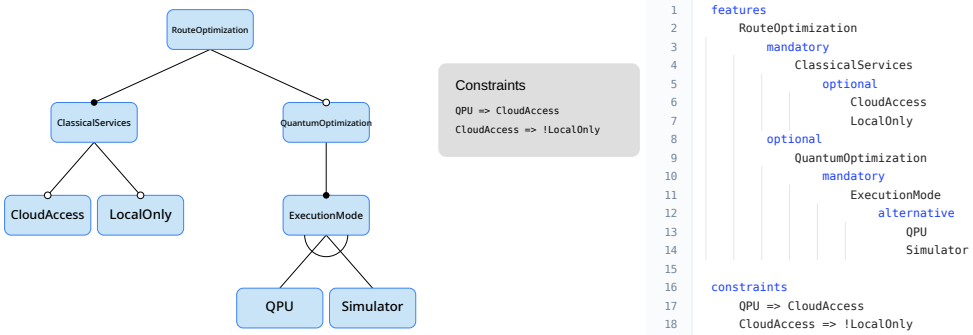


Fig. 2 Illustrative feature model for a route-optimization product line.

2.3 Quantum-Aware Variability Modeling in HSS

Quantum-aware variability modeling represents the configuration dependencies introduced by integrating classical and quantum software. Quantum Software Engineering (QSE) and development-lifecycle perspectives highlight the need to coordinate algorithmic, technological, and integration decisions across these elements (Weder et al. 2022; Serrano et al. 2022). A variability model makes the corresponding alternatives and constraints explicit; selecting a programming framework, execution mode, or backend must remain consistent with the other configuration choices. SPL-HQC applies this perspective through a FM that captures HSS variation points and their dependencies, supporting the systematic derivation of admissible configurations (Sepúlveda et al. 2025b).

An important distinction is that configuration validity does not necessarily imply operational suitability. Modeled dependencies determine whether a combination is admissible, whereas its suitability for a particular execution also depends on hardware limitations and noise-related conditions in NISQ environments (Preskill 2018), cloud-resource characteristics (Leymann et al. 2020), and quantum–classical orchestration requirements (Weder et al. 2021b). Quantum-aware variability modeling thus provides a configuration foundation that can be complemented with runtime evidence to assess whether selected decisions require adaptation during operation.

Continuing the route-optimization illustration, Table 1 distinguishes feature-model validity from execution feasibility. A configuration selecting QuantumOptimization, QPU, and CloudAccess, with LocalOnly unselected, satisfies the illustrative FM. Nevertheless, a circuit requiring eight qubits cannot execute on a selected QPU backend providing only five. This resource constraint complements the boolean feature relationships and must be checked against the concrete circuit and backend.

Table 1: Illustrative configuration-validity and execution-feasibility checks. Numerical values are hypothetical.

Check	Condition	Outcome
Requires	QPU implies CloudAccess; both selected	Satisfied
Excludes	CloudAccess excludes LocalOnly; LocalOnly deselected	Satisfied
Qubit capacity	Required qubits $\leq$ available qubits: $8 \leq 5$	Not satisfied

Passing the qubit-capacity check is necessary but not sufficient for execution feasibility; connectivity, supported operations, and other platform constraints may also need to be considered.

2.4 Dynamic Software Product Lines and Runtime Adaptation

DSPLs extend product-line engineering by supporting the binding and rebinding of selected variation points during system operation (Hallsteinsen et al. 2008; Aguayo and Sepúlveda 2022). A binding associates a variation point with a concrete alternative; runtime reconfiguration revises selected bindings to obtain another configuration within the modeled product-line space. Not every variation point needs to become dynamic:

decisions fixed during design or deployment may coexist with runtime-reconfigurable ones.

DSPL-based adaptation relates the active configuration, the context motivating adaptation assessment, and the variability constraints defining admissible alternatives. Variability management in self-adaptive DSPLs provides representations and mechanisms for maintaining these relationships during operation (Aguayo and Sepúlveda 2022). This perspective distinguishes configuration validity from reconfiguration enactment: validity concerns whether the resulting combination satisfies the modeled constraints, whereas enactment concerns applying the selected changes to the running system. An example of a DSPL for modeling self-adaptive systems in the underwater robotics domain is provided by (Päßler et al. 2025).

For HSS, this perspective provides a basis for assessing whether selected configuration choices require adaptation when execution resources, quality conditions, costs, or application contexts change during operation.

For example, consider an HSS with a runtime-reconfigurable execution mode offering QPU and simulator alternatives. A policy may prefer simulator execution when the estimated QPU queue time exceeds an illustrative threshold of 120 s. Listing 1 shows how this preference is subject to configuration constraints.

Listing 1 Illustrative guarded runtime rebinding.

```
if estimatedQueueTime > 120 s then
    candidate := Simulator

    if admittedByVariabilityModel(candidate)
       and compatibleWithExecutionStack(candidate)
       and supportsResourceRequirements(candidate) then
        rebind executionMode to candidate
    else
        reject candidate
    end if
end if
```

The queue-time condition triggers an assessment of the current binding, whereas the subsequent checks determine whether the proposed alternative is admissible. The example distinguishes adaptation preferences from constraints defining valid reconfigurations (Aguayo and Sepúlveda 2022).

2.5 Self-Adaptive Systems and the MAPE-K Loop

Self-adaptive systems modify their behavior at runtime in response to changes in their environment, internal state, requirements, or quality goals. MAPE-K is a widely used feedback-loop model that structures adaptation into Monitor, Analyze, Plan, and Execute phases operating over shared Knowledge (Kephart and Chess 2003; Cheng et al. 2009). Monitor observes the system and environment, Analyze determines whether adaptation is required, Plan determines an adaptation, and Execute applies it, while Knowledge maintains the models, context, policies, constraints, and historical evidence used by the loop.

In variability-intensive self-adaptive systems, MAPE-K can be combined with runtime variability models so that adaptation involves not only changing parameters but also rebinding variation points or reconfiguring software components while preserving configuration consistency.

For HSS, this feedback-loop perspective is applicable when operational evidence can affect the suitability of an active software configuration, providing a generic control structure for monitoring such changes and coordinating runtime reconfiguration.

Figure 3 organizes the illustrative runtime rebinding into MAPE-K responsibilities. Monitor observes the QPU queue time, Analyze detects a policy-threshold violation, Plan proposes and validates simulator execution, and Execute applies the rebinding before subsequent circuit execution. Shared Knowledge supports all phases through configuration, variability, policy, and runtime evidence.

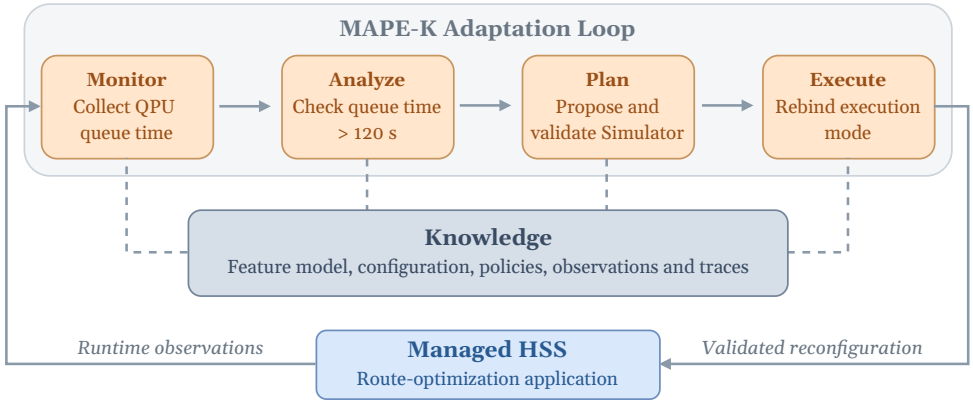


Fig. 3 MAPE-K loop for illustrative execution-mode rebinding.

2.6 LLM-Supported Reasoning, Deterministic Validation, and Quantum DevOps

LLMs have been explored as reasoning components in software-intensive systems when decision-making requires interpreting heterogeneous information, generating explanations, or supporting human-readable interaction with technical artifacts (Wang et al. 2024; Donakanti et al. 2024; Alves et al. 2026). In runtime adaptation, they may assist in interpreting contextual evidence, reasoning about alternatives, and producing candidate adaptations or explanatory rationales.

LLMs, however, are probabilistic and should not be treated as formal configuration reasoners. Their outputs may violate structural dependencies, configuration constraints, or domain policies (Wang et al. 2024; Xiao et al. 2026). In variability-intensive systems, FMs and deterministic constraint analysis provide mechanisms for checking whether a candidate configuration belongs to the admissible configuration space before enactment (Benavides et al. 2010; Czarnecki and Wasowski 2007).

Quantum DevOps extends operational software-engineering concerns to quantum software by emphasizing deployment, monitoring, experimentation, execution management, and operational feedback, particularly under NISQ conditions (Gheorghe-Pop et al. 2020). For HSS, these mechanisms provide an operational basis for observing and managing coordinated classical and quantum executions.

Figure 4 illustrates the separation between LLM-supported candidate generation and deterministic validation. For example, a candidate selecting Simulator and LocalOnly while retaining CloudAccess is rejected because it violates the exclusion defined in the illustrative FM. The LLM rationale does not override this constraint.

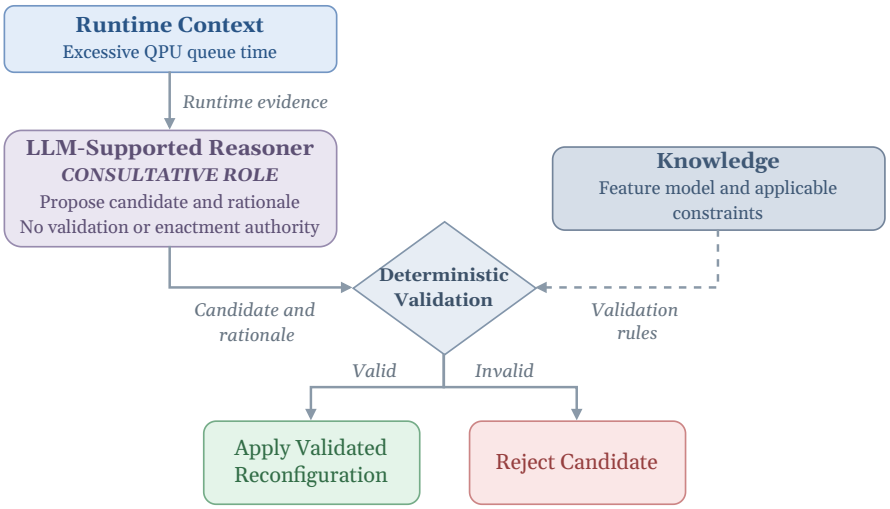


Fig. 4 Separation of LLM-supported candidate generation and deterministic validation.

3 Related Work and Research Gaps

This section positions QUASAR with respect to related work on runtime adaptation in HSS across three complementary perspectives: runtime variability and adaptation mechanisms; model-based variability, observability, and traceability; and AI/LLM-supported reasoning and deterministic validation. The resulting synthesis identifies the research gaps and design requirements. The complete structured search, evidence-extraction process, and extended evidence synthesis are documented in the companion QUASAR Literature Evidence Report (Sepúlveda et al. 2026a).

3.1 Review Scope and Evidence Synthesis

To position QUASAR, we conducted a structured literature-based gap analysis rather than a full systematic literature review. The analysis covered HSS runtime adaptation, SPL/DSPL, variability management, Quantum DevOps, runtime observability and traceability, and AI/LLM-supported software decision-making. Studies addressing

runtime factors, adaptation mechanisms, monitoring and execution evidence, variability and validation, or AI-supported decision-making were considered, whereas work restricted to quantum algorithms, hardware physics, or low-level circuit optimization without software-architecture or runtime-management implications was excluded.

The synthesis examined which runtime factors and adaptation mechanisms are reported for HSS, how adaptation decisions are represented and validated, and how AI/LLM mechanisms support adaptive decision-making. We extracted the corresponding runtime factors, mechanisms, validation approaches, and AI roles to derive the design requirements.

3.2 Runtime Variability and Adaptation in HSS

Related work identifies three recurring sources of runtime variability in HSS. Quantum-execution conditions include noise, calibration state, topology, connectivity, and execution-quality indicators (Preskill 2018; Murali et al. 2019); cloud-resource conditions include backend availability, queue latency, provider load, execution time, cost, quotas, and provider constraints (Ravi et al. 2021; Giortamis et al. 2025; Cranganore et al. 2024; Ye et al. 2026); and hybrid-workflow conditions include quantum–classical coordination, data movement, asynchronous execution, workflow state, and end-to-end quality-of-service concerns (Weder et al. 2021b; Burgholzer et al. 2026; Seitz et al. 2024).

Existing approaches respond at multiple levels. Infrastructure-oriented mechanisms select among QPUs, simulators, or providers (Ravi et al. 2021; Giortamis et al. 2025; Ye et al. 2026); execution-level mechanisms adapt mappings, compilation, error-mitigation techniques, partitioning, or execution parameters (Murali et al. 2019; Burgholzer et al. 2026; Seitz et al. 2024; Jaleel et al. 2026); and workflow-level approaches replace or reassign quantum and classical stages or activate alternative execution paths (Giortamis et al. 2025; Seitz et al. 2024; Sepúlveda et al. 2025a). Selection is commonly driven by heuristics, optimization functions, or learned policies.

Table 2 summarizes the adaptation factors and mechanisms explicitly addressed by the reviewed approaches.

Table 2: Conditions and mechanisms addressed by related work on resource-aware execution, orchestration, and adaptation in HSS.

Ref.	Conditions or requirements	Mechanism and scope
Murali et al. (2019)	Calibration-dependent gate and readout errors, coherence, and device connectivity.	Noise-aware qubit placement, routing, and scheduling during compilation.
Ravi et al. (2021)	Queue delays, predicted execution fidelity, and resource availability.	Adaptive job scheduling and quantum-device selection to balance execution quality and waiting time.

Continued on next page

Table 2 – continued

Ref.	Conditions or requirements	Mechanism and scope
Giortamis et al. (2025)	Fidelity requirements, completion time, resource heterogeneity, and workload distribution.	Resource planning and scheduling across quantum and classical resources, balancing fidelity and completion time.
Cranganore et al. (2024)	Task requirements, resource availability, and available classical or quantum implementations.	Proposed workflow-management architecture for selecting task implementations and mapping tasks to resources.
Ye et al. (2026)	Circuit requirements, qubit capacity, estimated fidelity, execution time, and cost.	Feasibility filtering and hardware recommendation before backend-specific execution.
Weder et al. (2021b)	Workflow dependencies, quantum–classical coordination, and deployment requirements.	Combined workflow orchestration and infrastructure provisioning using BPMN and TOSCA.
Burgholzer et al. (2026)	Device capabilities, calibration information, and heterogeneous execution resources.	Integrated resource management, scheduling, and hardware-aware compilation in an HPC–quantum software stack.
Seitz et al. (2024)	Circuit size, device capacity, noise, and resource utilization.	Quantum-task scheduling and circuit cutting to accommodate execution-resource constraints.
Jaleel et al. (2026)	Circuit depth, backend noise, expected latency, and qubit connectivity.	Runtime allocation of computational stages to quantum hardware or classical simulation using a multifactor cost model.
Sepúlveda et al. (2025a)	Quantum hardware fidelity, queue latency, and cloud costs.	DSPL-based proposal integrating MAPE-K and solver-based multi-objective optimization for runtime selection of algorithm and backend configurations.

Taken together, these studies show that execution suitability depends on both quantum-device conditions and the coordination of classical and quantum resources. They also distinguish resource-aware execution mechanisms from explicit runtime variability management. The DSPL proposal by (Sepúlveda et al. 2025a) provides a preliminary connection between these perspectives. Building on this direction, QUASAR makes the relationships among modeled alternatives, runtime information, and configuration constraints explicit and separates candidate generation from deterministic validation before enactment.

3.3 Model-Based Variability, Observability, and Traceability

Our previous SPL-HQC work supports the derivation of valid HSS configurations through explicit variability modeling and a development process spanning computing-independent, platform-independent, and platform-specific abstraction levels (Sepúlveda et al. 2024; Sepúlveda et al. 2025b). Its focus is primarily on design-time configuration rather than systematic runtime rebinding.

A subsequent DSPL-oriented extension explored how HSS configurations and monitored quality attributes can participate in runtime adaptation through a MAPE-K-oriented architecture (Sepúlveda et al. 2025a). More broadly, research on self-adaptive DSPLs has combined runtime variability models with context-aware FMs and constraint-based reasoning to preserve configuration consistency under changing conditions (Aguayo and Sepúlveda 2022). Formal modeling and verification of feedback loops further emphasize the need for adaptation decisions to remain consistent and inspectable before execution (Hachicha et al. 2016). These works provide a model-based foundation for runtime reconfiguration, but the reviewed approaches do not yet integrate decision-specific runtime evidence and adaptation provenance as explicit architectural concerns for HSS.

From a variability-modeling perspective, the distinction is whether adaptation alternatives and constraints are represented explicitly or remain embedded in operational decision logic. The scheduling and workflow approaches discussed in Section 3.2 mainly illustrate the latter (Ravi et al. 2021; Weder et al. 2021a; Giortamis et al. 2025; Jaleel et al. 2026; Burgholzer et al. 2026; Seitz et al. 2024).

Runtime observability provides complementary evidence to inform adaptation decisions within the modeled configuration space. Quantum DevOps and related cloud, workflow, compilation, and metadata-oriented approaches expose backend status, queue conditions, calibration data, execution parameters, workflow state, compilation decisions, and execution outcomes (Gheorghe-Pop et al. 2020; Ravi et al. 2021; Giortamis et al. 2025; Ye et al. 2026; Weder et al. 2021a,b; Cranganore et al. 2024; Murali et al. 2019; Burgholzer et al. 2026; Huenchuleo et al. 2026). However, this evidence is commonly consumed locally by scheduling or execution mechanisms, rather than being represented as reusable runtime knowledge explicitly associated with configuration decisions.

Traceability remains similarly fragmented. SPL and model-based approaches provide links among requirements, features, configuration decisions, and software artifacts, while QSE lifecycle and workflow approaches provide partial links from architectural abstractions toward executable resources (Clements and Northrop 2001; Pohl et al. 2005; Piattini et al. 2020; Serrano et al. 2022; Weder et al. 2021a,b, 2022). Runtime schedulers and metadata-oriented approaches further relate operational information to execution decisions and outcomes (Ravi et al. 2021; Giortamis et al. 2025; Ye et al. 2026; Jaleel et al. 2026; Cranganore et al. 2024; Huenchuleo et al. 2026). However, these relationships remain largely fragmented across development and runtime operations. Overall, existing work provides complementary support for explicit variability, runtime observability, and partial traceability, but these concerns remain only weakly integrated across design-time variability knowledge, runtime evidence, adaptation decisions, configuration validation, and adaptation provenance.

3.4 AI/LLM-Supported Reasoning and Deterministic Validation

Runtime decision-making in quantum and hybrid execution environments has been supported through reinforcement learning, heuristics, and optimization mechanisms embedded in schedulers, resource managers, and workflow mechanisms (Ravi et al. 2021; Giortamis et al. 2025; Ye et al. 2026; Seitz et al. 2024; Jaleel et al. 2026; Cranganore et al. 2024; Luo et al. 2025).

LLMs provide a different form of support through their ability to interpret heterogeneous information, generate explanations, and assist in reasoning over alternative decisions (Wang et al. 2024). These capabilities make them potentially useful for interpreting runtime evidence and explaining trade-offs among alternative adaptations. An LLM-assisted self-adaptation architecture has been proposed in which an LLM analyzes contextual information and suggests adaptation actions, offering an alternative to traditional rule-based or machine-learning approaches and potentially supporting responses to changes not anticipated at design time (Donakanti et al. 2024). According to (Alves et al. 2026), the limitations present in traditional self-adaptive systems are further explored, as they are based on predefined rules and policies established at design time, which makes self-adaptation to unanticipated situations difficult. To address this, an extension of MAPE-K that incorporates an LLM to support adaptation in such situations is proposed.

However, the reviewed literature provides limited evidence of LLMs being systematically integrated as runtime adaptation components for HSS. Their role in this context, therefore, remains exploratory rather than that of an established configuration mechanism.

Related software-engineering work also supports separating LLM reasoning from authoritative control. Xiao et al. argues that concentrating interpretation, recommendation, and control responsibilities in an LLM can reduce transparency and maintainability, motivating explicit engineering control over core decision logic (Xiao et al. 2026). Complementarily, feature-model analysis and DSPL research provide deterministic mechanisms for checking structural consistency and runtime configuration validity (Czarnecki and Wasowski 2007; Benavides et al. 2010; Aguayo and Sepúlveda 2022; Hachicha et al. 2016). Together, these strands support an architectural separation between contextual AI-supported reasoning and authoritative configuration validation.

The relevant gap is therefore not the absence of AI-supported runtime decision mechanisms, but the limited support for integrating contextual LLM reasoning with explicit model-based validation and auditable adaptation decisions in HSS.

3.5 Research Gaps and Design Requirements

Across the reviewed literature, the main limitation is integration: runtime evidence, adaptation choices, variability management, AI-supported reasoning, deterministic validation, and adaptation provenance are generally addressed as separate concerns rather than as parts of a connected runtime architecture. Table 3 synthesizes the resulting research gaps, design requirements, and corresponding QUASAR responses.

The study-level evidence traceability and the detailed mapping from the reviewed literature to these six gaps and design requirements are provided in the companion

QUASAR Literature Evidence Report (Sepúlveda et al. 2026a). These requirements define the architectural rationale operationalized by the QUASAR architecture presented in Section 4.

Table 3: Research gaps and design requirements derived for QUASAR.

Gap	Design requirement	QUASAR response
Adaptation actions remain implicit in implementation-specific decision logic.	Represent runtime-controllable adaptation choices explicitly.	Runtime Decision Catalog.
Runtime evidence remains embedded in scheduling, execution, workflow, or optimization mechanisms.	Externalize observable runtime conditions as reusable information assets.	Runtime Information Catalog.
Runtime decisions and the evidence supporting them are weakly connected.	Relate each adaptation decision to the runtime information relevant to its analysis and reconsideration.	Decision–Information Matrix.
Design-time HSS variability is not systematically operationalized as an explicit runtime reconfiguration space.	Identify runtime-rebindable variation points and preserve their admissible alternatives and configuration constraints.	Runtime Variability Model.
The separation between AI-supported adaptation proposals and authoritative enactment control remains insufficiently explicit in the reviewed HSS approaches.	Assign contextual reasoning a consultative role and require independent validation before candidate enactment.	LLM-supported contextual reasoner and deterministic constraint-based guardrails.
Adaptation provenance remains fragmented across runtime observation, decision, validation, and enactment.	Preserve auditable links among observations, affected decisions, candidates, validation results, and enactment outcomes.	MAPE-K Knowledge and adaptation traces, supported by Quantum DevOps operational integration.

4 Proposed Approach: QUASAR

This section presents QUASAR, a DSPL architecture for model-based runtime self-adaptation in HSS. QUASAR coordinates variability knowledge, runtime evidence, adaptation decisions, contextual reasoning, deterministic validation, execution, and adaptation provenance within a model-based feedback architecture. Building on the SPL-HQC approach initially introduced at Q-SE 2024 and subsequently developed in Information and Software Technology (Sepúlveda et al. 2024; Sepúlveda et al. 2025b), QUASAR operationalizes the design requirements derived in Section 3.5. Its scope encompasses a broader range of runtime variability decisions than those instantiated in the scenarios evaluated in this paper.

4.1 Overview of QUASAR

A central design decision in QUASAR is to distinguish variation points according to their binding time: design-time fixed (`DT`), deployment-bound (`DEP`), and runtime-controllable (`RT`). `DT` decisions define stable aspects of the HSS and its admissible configuration space; `DEP` decisions are fixed at deployment; and `RT` decisions remain eligible for runtime operation. Runtime-observed information (`OBS`) provides evidence for evaluating `RT` decisions without constituting a configuration choice or constraint. Figure 5 presents the high-level architecture that coordinates these decisions with runtime evidence and adaptation control. Solid arrows represent the operational adaptation flow, while dashed arrows indicate supporting information exchanges and interactions with Knowledge.

Only variation points explicitly classified as `RT` may be assessed for runtime adaptation. Candidate configuration changes must satisfy the variability constraints inherited from SPL-HQC, along with applicable architectural, platform, and policy restrictions.

In QUASAR, runtime adaptation is an evidence-driven process in which `RT` decisions are assessed to determine whether a configuration change is warranted; thus, the current configuration may remain unchanged. When an admissible change is enacted, it constitutes a runtime reconfiguration, whereas rebinding refers specifically to replacing a previously selected alternative at a variation point.

The Managed HSS contains the classical and quantum software elements that are subject to observation or adaptation. A MAPE-K loop observes this system, identifies affected runtime decisions, plans candidate adaptations, and enacts only validated reconfigurations.

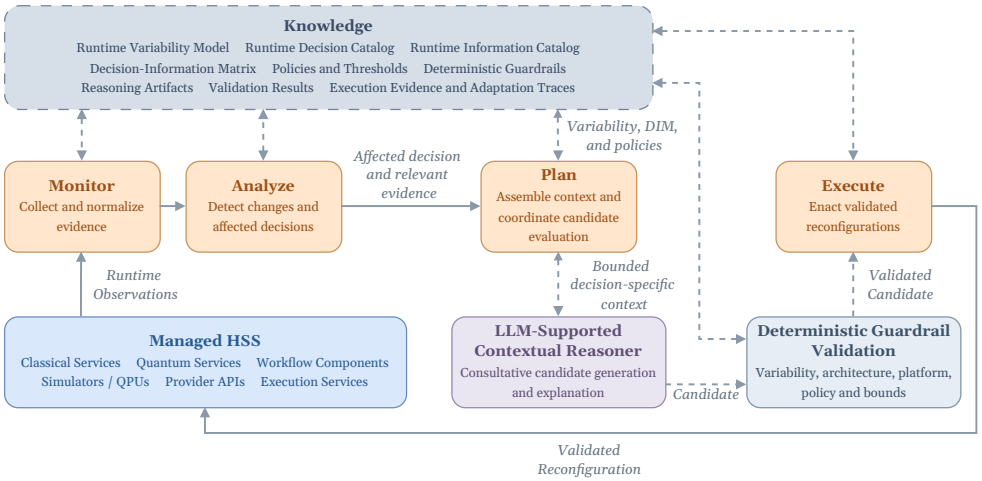


Fig. 5 High-level architecture of QUASAR.

Knowledge provides the shared architectural assets required across the loop, including runtime variability, decisions and information, policies, guardrails, reasoning

and validation artifacts, execution evidence, and adaptation traces. It represents an architectural responsibility rather than necessarily a single physical repository.

Before runtime, SPL-HQC establishes the admissible variability space, while QUASAR specifies runtime decisions, information dependencies, applicable policies, and validation constraints. During operation, Monitor contributes runtime observations, and Analyze identifies affected `RT` decisions. Plan coordinates LLM-supported candidate generation, while deterministic guardrails independently assess candidate admissibility before Execute. Candidate rationales, validation outcomes, enacted configurations, and execution evidence are retained in Knowledge as adaptation provenance, without redefining the design-time variability space.

Operational integration can draw on Quantum DevOps practices to connect deployment, monitoring, and execution with the collection of adaptation evidence (Gheorghe-Pop et al. 2020).

In the current architecture, LLM support is scoped to the Plan. Monitor collects and normalizes runtime observations, while Analyze evaluates these observations against applicable conditions to identify affected runtime decisions. Plan then assembles a bounded reasoning context from the relevant observations and Knowledge assets, including the current configuration, admissible alternatives, DIM-defined information, and applicable constraints and policies. This context is supplied to the LLM-supported reasoner to generate a candidate adaptation and its rationale. Contextual interpretation by the reasoner supports planning; it does not replace monitoring or adaptation-trigger detection.

A key architectural boundary separates candidate reasoning from configuration validity. The LLM-supported reasoner may interpret decision-specific evidence, propose adaptations, and explain its rationale, but it has neither validation nor execution authority. Deterministic guardrails retain authority over candidate admissibility before Execute.

4.2 Runtime Decision Catalog

The Runtime Decision Catalog explicitly represents the HSS configuration decisions associated with variation points classified as `RT`. Each decision identifies a configurable aspect that may be reconsidered during operation, independently of the software mechanisms used to propose, validate, or enact an adaptation.

The current architectural catalog comprises six decisions: backend selection (D1), execution mode/simulator fallback (D2), error-mitigation strategy (D3), shot adjustment (D4), workflow fallback (D5), and HQC capability activation/bypass (D6) (Table 4). These concerns derive from the synthesis in Section 3 and our previous DSPL-oriented work (Sepúlveda et al. 2025a).

The representative adaptation triggers in Table 4 may arise from runtime-observed information, explicit policies, predefined constraints, or their evaluation in combination. They indicate when an `RT` decision should be assessed for potential adaptation; they do not redefine `OBS` information or the decision–information dependencies specified by the Runtime Information Catalog and the Decision–Information Matrix.

Table 4: Runtime Decision Catalog in QUASAR. All decisions (D1–D6) correspond to runtime-controllable (`RT`) variation points. D1–D4 primarily concern quantum execution; D5–D6 concern hybrid workflow orchestration.

ID	Runtime decision	Adaptation trigger	Adaptation candidate	Affected HSS concern
D1	Backend selection	Backend unavailability, excessive queue, degraded execution conditions, or provider-related constraints	Keep current backend, rebind to another admissible backend, change provider, or defer execution	Quantum execution service
D2	Execution mode / simulator fallback	QPU unavailability, excessive latency or cost, execution failure, or policy condition	QPU execution, cloud simulator, local simulator, or deferred execution	Execution layer
D3	Error-mitigation strategy	Degraded noise-related indicators or execution-quality evidence	Keep current strategy, activate an admissible mitigation technique, change mitigation strategy, or disable mitigation when admissible	Quantum execution pipeline
D4	Shot adjustment	Noise-related execution conditions, uncertainty or quality requirements, cost constraints, or execution limits	Reduce, keep, or increase the number of shots within the allowed bounds	Execution parameters
D5	Workflow fallback	Task failure, timeout, unavailable quantum resource, or workflow-level constraint violation	Retry, activate simulator fallback, use an alternative workflow fragment, or invoke an admissible classical fallback	Hybrid workflow
D6	HQC capability activation / bypass	Application-context change, problem characteristics, domain policy, or conditions making quantum participation unsuitable	Keep the current HQC state, activate the HQC capability, or bypass quantum execution	HQC capability / hybrid workflow

The catalog is a representative, non-exhaustive set of runtime decisions. D1–D4 primarily concern quantum execution, whereas D5–D6 address hybrid workflow orchestration, including admissible classical fallback and HQC capability activation or bypass. The current scope does not explicitly model problem partitioning, quantum–classical

workload redistribution, or changes to variational objectives, weights, or ansatz structures. These extensions are left for future work. Only D1, D4, and D6 are instantiated and evaluated independently in Sections 5–7.

4.3 Runtime Information Catalog

The Runtime Information Catalog externalizes the observable runtime evidence used to trigger adaptations and define certain adaptation candidates. Rather than embedding observations in provider-specific or planning logic, QUASAR represents them as explicit information assets that may trigger, inform, or document decision adaptations.

Whereas the Runtime Decision Catalog specifies what may change, the Runtime Information Catalog specifies what may be observed. Its entries correspond to `OBS` information and are not configuration choices, explicit policies, or constraints.

The catalog comprises eight categories spanning infrastructure, execution, workflow, outcome, and application-context evidence (Table 5). Workflow state (I6) and application context and problem characteristics (I8) explicitly support hybrid orchestration, while other categories may inform both quantum-execution and hybrid-workflow decisions. These categories are not intended as an exhaustive information model for adaptive hybrid algorithms.

Table 5: Runtime Information Catalog in QUASAR.

ID	Information category	Representative runtime information
I1	Backend availability and status	Availability, operational status, advertised backend capabilities, and supported execution modes
I2	Queue and latency	Queue length, estimated waiting time, job status, observed latency, and timeout events
I3	Calibration and noise	Calibration timestamp or age, gate/readout errors, coherence-related indicators, and topology and connectivity metadata
I4	Cost and quota	Execution cost, quota consumption, remaining credits, and current resource-usage or billing information
I5	Circuit and execution properties	Circuit depth and width, gate count, number of shots, optimization level, and transpilation metadata
I6	Workflow state	Task status, failures, retry count, timeout events, dependency state, and workflow progress
I7	Execution outcome and quality evidence	Execution success or failure, result distribution, observed latency, quality proxies, and post-execution statistics
I8	Application context and problem characteristics	Problem size or complexity, application context, workload characteristics, task-specific conditions, and other contextual indicators relevant to the applicability of the HQC capability

Observed information is acquired during operation and made available through Knowledge. Explicit policies, thresholds, and validation rules remain separate from

these observations, while their decision-specific relevance is defined by the Decision–Information Matrix. Runtime evidence may, therefore, trigger or inform the assessment of an `RT` decision, but it does not determine candidate admissibility.

4.4 Decision–Information Matrix

The Decision–Information Matrix (DIM) relates each runtime decision to the information relevant to assessing its potential adaptation, thereby defining a decision-specific information context rather than exposing every decision to the complete monitoring stream.

An entry is required (R), optional/supporting (O), or not used by default (–); these relations define an architectural information profile rather than a requirement to collect every observation in every cycle.

The matrix represents only decision–observation dependencies. Policies, thresholds, variability constraints, and validation rules are part of Knowledge but remain distinct from `OBS` information and are not encoded as information categories. Table 6 specifies these relations for D1–D6 and I1–I8.

Table 6: Decision–Information Matrix.

Runtime decision	**I1**	**I2**	**I3**	**I4**	**I5**	**I6**	**I7**	**I8**
D1 Backend selection	R	R	O	O	O	–	O	–
D2 Execution mode / simulator fallback	R	O	O	O	R	O	O	–
D3 Error-mitigation strategy	O	–	R	O	R	–	O	–
D4 Shot adjustment	–	O	R	O	R	–	O	–
D5 Workflow fallback	O	O	–	O	–	R	O	–
D6 HQC capability activation / bypass	–	–	–	O	O	O	O	R

R: required; **O**: optional/supporting; –: not used by default.
Information categories.
I1: Backend availability/status; **I2**: Queue/latency; **I3**: Calibration/noise; **I4**: Cost/quota; **I5**: Circuit/execution properties; **I6**: Workflow state; **I7**: Execution outcome/quality; **I8**: Application context/problem characteristics.

For the evaluated D4 case, noise-related evidence (I3) and the current execution configuration (I5) are required. I5 includes the current shot setting as observed configuration information; D4 remains the decision to adjust the shot count within its admissible bounds.

For D6, application context and problem characteristics (I8) are required observations. The domain policy defining HQC applicability is maintained in Knowledge, separately from these observations. Thus, D6–I8 expresses an information dependency rather than a policy rule.

At runtime, the DIM bounds the information context supplied for an affected decision; it does not transfer validation authority to the reasoning mechanism. The same relation can be retained in the adaptation trace to associate the affected decision with the evidence available when it was assessed for adaptation.

4.5 Runtime Variability Model and SPL-derived Traceability

The Runtime Variability Model (RVM) provides a runtime-enabled view of the HSS variability space established through SPL-HQC rather than an independent configuration model. It preserves the alternatives and constraints relevant during operation while identifying the variation points eligible for runtime adaptation.

Only `RT` variation points enter the Runtime Decision Catalog, whereas `OBS` information remains separate from configuration choices and constraints. In this context, the current space of `RT` decisions is determined by D1–D6.

Figure 6 distinguishes the design-time foundation from runtime adaptation provenance. QUASAR preserves available requirements–variability–architecture relationships and complements them with the runtime chain from observations and affected decisions to candidate validation, as well as the respective enactment or rejection actions.

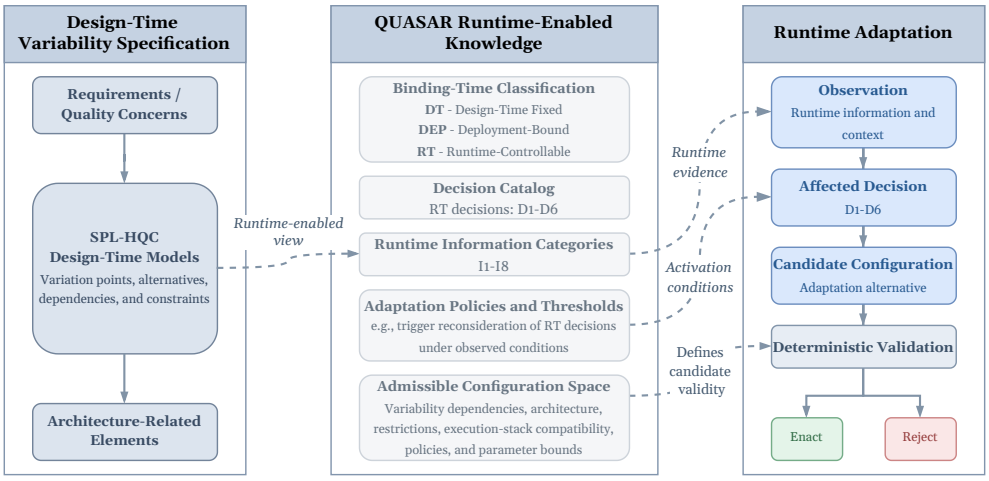


Fig. 6 Runtime Variability Model and traceability in QUASAR.

The empirical evaluation presented in Section 7 primarily examines the runtime adaptation provenance chain: Observation → Affected Decision → Candidate → Validation → Enactment/Reject. It does not empirically validate the complete requirements-to-runtime traceability chain.

4.6 LLM-Supported Reasoning and Deterministic Guardrails

QUASAR addresses the decision-control requirement identified in Section 3.5 by separating consultative candidate generation from authoritative validation. During Plan, the LLM-supported reasoner interprets decision-specific evidence and may propose a candidate adaptation and rationale. Candidate admissibility is determined through deterministic checks independently of the LLM's recommendation, and only validated candidates may proceed to Execute.

Plan constructs the reasoning context for the affected decision by combining the relevant runtime evidence identified through the DIM with the current configuration, admissible alternatives, and applicable constraints and policies from Knowledge. The LLM receives this bounded context as input; restricting its reasoning scope does not guarantee candidate validity.

The candidate rationale is retained as adaptation evidence rather than proof of validity. Candidate admissibility is determined independently through deterministic guardrails grounded in the RVM and complemented by architecture, execution-stack, policy, and parameter constraints. Table 7 defines the six validation concern classes.

Table 7: Deterministic guardrail classes defined in QUASAR.

ID	Guardrail	Validation purpose	Rule source	Illustrative D/I links
G1	Feature-model validity	Checks whether the candidate preserves mandatory features, dependencies, exclusions, and admissible feature combinations.	Runtime Variability Model	D1–D6
G2	Execution-stack compatibility	Verifies compatibility among the candidate backend and the associated algorithm, framework, SDK, language, provider, and execution service, as applicable.	Variability and platform constraints	D1; I1, I5
G3	Execution-mode feasibility	Checks whether the proposed execution mode is admitted and feasible for the current configuration.	Variability, platform, and execution constraints	D2; I1, I5
G4	Policy and operational compliance	Checks whether a candidate satisfies applicable operational and domain policies that can be evaluated before enactment, including modeled cost, quota, latency, quality, or context conditions when relevant.	Operational and domain policies	D1; I2 / D6; I8
G5	Architecture and workflow consistency	Verifies that the candidate preserves required architectural dependencies, integration constraints, and the predefined workflow or fallback paths admitted by the deployed configuration.	Architecture and workflow rules	D5; I6

Continued on next page

Table 7 – continued

ID	Guardrail	Validation purpose	Rule source	Illustrative D/I links
G6	Parameter bounds	Checks whether candidate values for runtime-controllable execution parameters, such as shots, timeout, optimization level, or retry limits, remain within their admitted bounds before enactment.	Parameter and execution rules	D4; I5

At design time, engineers may define application- and decision-specific admissibility rules within the guardrail classes G1–G6. Such rules may constrain candidate configurations through quality thresholds, cost limits, latency bounds, or other domain policies, provided the relevant conditions can be evaluated before enactment. These rules complement adaptation triggers: triggers determine when an `RT` decision requires assessment, whereas guardrails determine whether a proposed candidate is admissible. The D/I associations in Table 7 are illustrative; the DIM specifies decision–observation dependencies, while candidate validation also depends on the applicable constraints and policies maintained in Knowledge.

G1–G6 denote validation concern classes rather than six independent or mutually exclusive validator modules. A concrete rule may contribute to more than one class; for example, a backend–algorithm constraint may be represented structurally under G1 while expressing the execution-stack concern captured by G2.

Only the guardrails applicable to the affected decision are evaluated. Candidates satisfying the mandatory checks may proceed to Execute; invalid candidates are rejected, while insufficient required evidence prevents an executable adaptation unless an explicitly modeled fallback applies.

The guarded planning process follows five steps:

1. runtime observations identify an affected `RT` decision;
2. the DIM defines its decision-specific information context;
3. the LLM-supported reasoner proposes a candidate adaptation and rationale;
4. applicable deterministic guardrails validate candidate admissibility; and
5. validated candidates proceed to execution, whereas invalid candidates are rejected.

Accepted and rejected candidates and their validation outcomes are retained in Knowledge through the adaptation provenance chain: Observation → Affected Decision → Candidate → Validation → Enactment/Reject. Validated adaptations are enacted through the managed HSS interfaces, with execution outcomes recorded as adaptation evidence.

5 Running Scenario

This section defines the running scenario and its adaptation cases; presenting the empirical assessment of QUASAR. We instantiate the architecture in a bounded running scenario that provides the basis for the prototype implementation and evaluation.

5.1 Scenario Context and Adaptation Need

The running scenario considers an HSS in which classical services coordinate an optimization workflow that may invoke an HQC capability depending on the problem characteristics and current operational conditions. The HQC capability uses a configurable execution backend and runtime execution parameters within a predefined hybrid workflow.

At deployment time, the system starts from an initially valid configuration that determines the HQC activation state, execution backend, and relevant execution parameters. During operation, changes in backend conditions, execution-quality indicators, or application context may render that configuration operationally unsuitable.

The scenario, therefore, illustrates bounded runtime adaptation: only explicitly runtime-controllable decisions are eligible for adaptation, and candidate changes must satisfy the modeled structural and policy constraints before enactment. Sections 5.2 and 5.3 instantiate this bounded adaptation space.

5.2 Initial HSS Configuration and Adaptation Space

The initial HSS configuration is derived from the design-time variability space defined in Section 4. For this scenario, only `DT` and `RT` configuration elements are instantiated as active concerns; `DEP` remains part of the general QUASAR binding model but is not exercised independently in C1–C3.

The three adaptation cases share a common scenario-level configuration template but address different runtime conditions and adaptation decisions. C1 concerns backend selection, C2 addresses execution-parameter adjustment, and C3 considers HQC capability activation or bypass. This common template organizes the adaptation cases without implying identical experimental conditions or equivalent computational workloads across their prototype realizations.

At the scenario-design level, application functionality, the optimization goal, and the overall hybrid workflow structure are treated as fixed adaptation boundaries. However, preservation of the same optimization workload across heterogeneous backend implementations is not established by the C1 prototype. The HQC activation state, execution backend, and number of shots are runtime-controllable and may be assessed for adaptation and, when appropriate, changed within their admitted alternatives and constraints.

Table 8 summarizes the initial configuration at the scenario level and identifies the binding classes and adaptation space shared by C1–C3. The cases differ in their triggering evidence and affected runtime decisions; they do not necessarily require different initial configurations. This scenario-level description does not constitute a complete specification of the experimental settings reported in Sections 6 and 7.

Table 8: Initial HSS configuration and runtime roles for C1–C3. Deployment-bound (`DEP`) decisions are not represented separately in this scenario.

Element	Initial state	Class / runtime role	Runtime treatment
Application functionality and optimization goal	Fixed application functionality with a hybrid optimization task	`DT`	Fixed at the scenario-design level; preservation across C1 adapters is not established.
Hybrid workflow structure	Classical services with a predefined optional HQC capability	`DT`	Fixed; no runtime workflow synthesis or restructuring.
HQC capability	Initially HQC-enabled	`RT`	May remain enabled or be bypassed according to context and domain policy.
Execution backend	Primary compatible backend initially selected	`RT`	May be rebound to another admissible backend.
Number of shots	Initial configured value	`RT`	May be adjusted within the admitted parameter bounds.
Domain and service-level policies	Configured policies, thresholds, and parameter bounds	Policies and constraints from Knowledge	Constrain analysis and candidate admissibility; not adaptation targets.
Runtime operational context	Backend, execution, application-context, and outcome evidence	`OBS`	Provides evidence for assessing `RT` decisions; not a configuration choice.

The instantiated adaptation space is therefore restricted to HQC activation, backend selection, and shot adjustment. These three runtime-controllable concerns motivate the infrastructure-, execution-, and context-driven cases introduced next.

5.3 Three Representative Adaptation Cases

Three cases illustrate distinct runtime adaptations within a predefined hybrid workflow: C1 may rebind the execution backend (D1) following infrastructure degradation, C2 may adjust the shot count (D4) in response to quantum-execution conditions, and C3 may enable or bypass the HQC capability (D6) according to application context and policy. Table 9 summarizes their runtime evidence, applicable constraints, affected decisions, and candidate adaptations.

Table 9: Representative runtime adaptation cases considered in the running scenario.

ID	Runtime situation	Runtime-observed information	Policies and constraints (Knowledge)	Affected decision	Candidate adaptations
C1	Infrastructure degradation	Backend latency, backend status, and operational availability	Latency threshold and applicable execution-stack compatibility constraints	Execution backend (D1)	Keep the current backend or rebind to another admissible backend adapter.
C2	Quantum-execution degradation	Simulated noise or execution-quality indicators, current execution configuration, and available execution outcomes	Execution-quality threshold and admitted parameter bounds	Number of shots (D4)	Keep the current value or adjust the number of shots within the admitted bounds.
C3	Context- or policy-driven change	Problem complexity, application context, workload characteristics, and other contextual indicators relevant to HQC applicability	Domain policy defining whether HQC participation is appropriate under the observed context	HQC capability (D6)	Keep the current HQC state or switch between HQC-enabled and bypassed execution within the predefined configuration space.

5.3.1 C1 – Infrastructure degradation.

C1 represents an initially valid backend that becomes operationally unsuitable, for example, because the observed latency or backend status violates the applicable operational condition. The affected decision is backend selection (D1), which may retain the current binding or rebind the HQC execution abstraction to another admissible backend adapter.

5.3.2 C2 – Quantum-execution degradation

C2 represents parameter-level adaptation triggered by simulated noise or execution-quality evidence (I3) together with the current execution configuration (I5). The affected decision is shot adjustment (D4), whose candidate values remain constrained by the configured quality condition and accepted parameter bounds; execution outcomes (I7) may provide supporting evidence.

5.3.3 C3 – Context- or policy-driven change

C3 represents capability-level adaptation in which the application or problem context (I8) is evaluated against a domain policy defining HQC applicability. This policy is maintained in Knowledge and is distinct from the observed context. The affected

decision is HQC activation or bypass (D6), and the candidate may retain or switch the HQC state only within the predefined runtime configuration and workflow.

Unlike C1 and C2, which primarily concern quantum execution, C3 addresses whether the HQC capability participates in the predefined hybrid workflow. It therefore illustrates a limited form of hybrid workflow adaptation without involving runtime workflow synthesis or restructuring.

Together, C1–C3 exercise infrastructure, parameter, and capability-level adaptation within a predefined hybrid workflow. C1–C3 are architectural analysis cases rather than experimental groups; the original prototype campaign used four operational scenario groups that do not map one-to-one to these cases. Detailed C1–C3 mappings, including their runtime evidence, relevant knowledge, affected decisions, and technical realization, are provided in the QUASAR Architecture and Technical Companion (Sepúlveda et al. 2026b).

6 Prototype Implementation

The prototype provides a controlled proof-of-concept (PoC) realization of QUASAR for the bounded adaptation space defined by C1–C3, consistent with the concept implementation as a research method described by (Ramesh et al. 2004). It operationalizes backend selection, execution-parameter adjustment, and HQC activation or bypass through MAPE-K-based control, LLM-supported candidate reasoning, deterministic validation, HQC execution abstractions, and adaptation tracing. The implementation focuses on a representative subset of QUASAR's architectural capabilities, providing an initial assessment of their feasibility in a controlled experimental setting. The complete architecture-to-prototype mapping, implementation structures, technology stack, and technical realization details are documented in the QUASAR Architecture and Technical Companion (Sepúlveda et al. 2026b).

6.1 Implementation Scope and Technology Stack

The prototype combines a classical application layer, an adaptation-control layer, and an HQC execution layer. It uses Python/Flask and React/Vite for the application interface, Docker-managed simulated classical services, and Google Gemini 2.5 Flash, accessed through a remote API, for LLM-supported contextual reasoning. Quantum execution relies on Qiskit Aer and Cirq Simulator, which instantiate distinct SDK-specific adapters. Using both simulation stacks allows C1 to exercise software-level backend rebinding across heterogeneous SDK interfaces rather than demonstrating semantically equivalent workload migration or comparative backend performance. All experiments were conducted under controlled simulation conditions, enabling the repeatable injection of latency- and noise-related conditions without reproducing physical-QPU or quantum-cloud operation.

FMweb-K is a framework for variability management in Dynamic Software Product Lines (DSPLs) through feature models and MAPE-K-based self-adaptation (Aguayo et al. 2025). The QUASAR prototype reuses structures from FMweb-K-Quantum, an HSS-oriented version available in the repository from which the QUASAR replication repository was forked. These structures represent a reduced operational subset of the

Runtime Variability Model, while Python-based control and logging modules maintain configuration, policy, runtime-state, and adaptation-trace information.

6.2 Prototype Architecture and Mapping to QUASAR

As shown in Figure 7, the prototype is organized into three logical layers. The managed HSS/application layer contains the classical application and services; the adaptation-control layer realizes the MAPE-K workflow, runtime Knowledge, LLM-supported candidate reasoning, and deterministic validation; and the HQC execution layer encapsulates SDK-specific execution through a common backend abstraction. This separation isolates application functionality, adaptation control, and quantum execution concerns. The dashed connections in Figure 7 indicate interactions with the Knowledge assets used during adaptation.

The implementation is intentionally partial to assess the feasibility of the QUASAR runtime adaptation workflow through C1–C3 (D1, D4, and D6) rather than implementing its complete architectural scope. Within this bounded scope, core control and execution responsibilities have direct software counterparts, whereas runtime adaptation artifacts and Knowledge are distributed across configuration structures, policies, control logic, runtime state, and logs. Table 10 summarizes this mapping; the complete element-level correspondence is provided by (Candia et al. 2026).

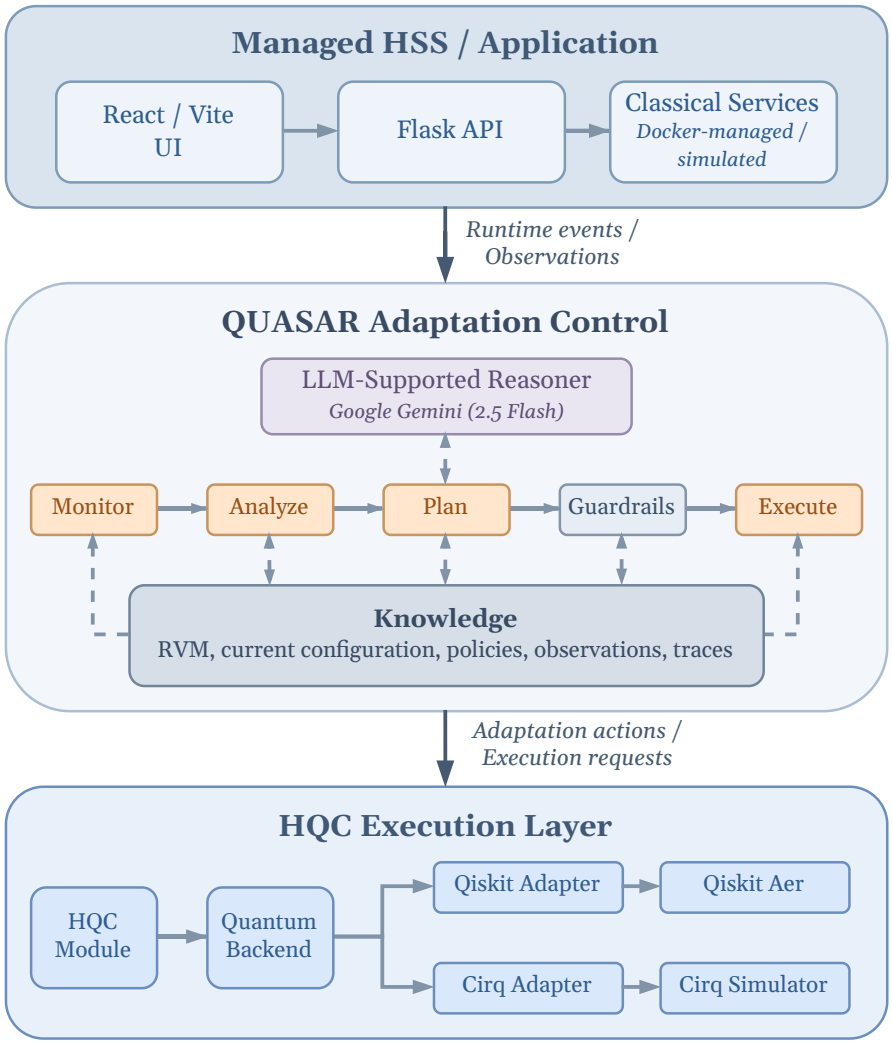


Fig. 7 Prototype architecture and mapping to QUASAR.

Table 10: Mapping between QUASAR architectural elements and their realization in the prototype.

QUASAR element	**Prototype realization**	**Implementation coverage**
MAPE-K	Monitor–Analyze–Plan–Execute logic	Direct for C1–C3
Runtime adaptation artifacts	Variability/configuration, policies, and decision/information associations	Partial, distributed
LLM-supported reasoning	LLM-supported contextual reasoner	Direct; candidate generation only

Continued on next page

Table 10 – continued

QUASAR element	Prototype realization	Implementation coverage
Deterministic validation	Instantiated variability, compatibility, policy and parameter checks	Partial guardrail subset
HQC execution abstraction	HQCModule, QuantumBackend, and Qiskit/Cirq adapters	Direct
Knowledge and provenance	Runtime state, observations, validation outcomes and logs	Partial/distributed

This mapping makes explicit that QUASAR artifacts are not necessarily implemented as one-to-one software modules. Some architectural concepts, such as the MAPE-K controller, the LLM reasoner, and the HQC abstraction, have identifiable software counterparts, whereas decision and information catalogs, variability constraints, policies, and Knowledge are distributed across configuration structures and runtime control logic. This distinction preserves traceability to the architectural proposal without overstating the implementation coverage of the PoC.

6.3 Runtime Variability, Policies, and Knowledge Representation

The prototype instantiates a reduced operational subset of the Runtime Variability Model defined in Section 4.5, whose variability semantics and admissible configuration space originate from SPL-HQC. The implemented subset covers three `RT` decisions exercised by C1–C3: backend selection (D1), shot count adjustment (D4), and HQC activation or bypass (D6). This representation operationalizes the runtime-variability requirement from Section 3.5 for the three implemented adaptation concerns, delimiting their runtime adaptations, admissible alternatives, and associated constraints.

The overall workflow structure remains fixed, while application-goal preservation is treated as a scenario-design assumption. Adaptation is defined by the corresponding variability constraints, along with configured latency and execution-quality thresholds, parameter bounds, and domain policies. These elements remain distinct from runtime observations.

Runtime Knowledge is distributed across model/configuration structures, MAPE-K state, scenario data, and execution logs rather than being implemented as a single repository. It comprises the current configuration, relevant variability and policy constraints, runtime observations, candidate and validation information, and resulting adaptation records. These assets provide the state consumed and updated by the runtime workflow described next.

Design-time variability constraints and adaptation policies are specified by engineers, whereas runtime observations, candidate adaptations, validation outcomes, and execution evidence are maintained by the corresponding MAPE-K activities during operation.

6.4 MAPE-K Runtime Workflow and Guarded LLM Reasoning

For C1–C3, Plan assembles a case-specific context containing runtime observations, the current configuration, applicable policies and constraints, and admissible alternatives. This context is supplied to the Google Gemini 2.5 Flash-based reasoner, which produces a candidate adaptation and contextual rationale. The LLM output is treated as a proposal. Before enactment, deterministic checks evaluate the candidate against the instantiated constraints, independently of the LLM-generated rationale. Invalid candidates are rejected, whereas accepted configuration changes are enacted through the managed HSS or HQC execution layer.

In the prototype, guardrail checks are embedded in the adaptation logic rather than implemented as six independent validator modules. The instantiated checks include variability constraints (G1), partial backend/stack compatibility for C1 (G2), HQC applicability policies for C3 (G4), and shot-count bounds for C2 (G6). Section 7.4 distinguishes their implementation from their empirical coverage.

6.5 HQC Execution and Adaptation Case Realization

Validated configuration changes are materialized through an HQC abstraction layer that isolates the adaptation-control logic from SDK-specific execution. As shown in Figure 8, the HQCModule selects an implementation of the common QuantumBackend abstraction according to the validated runtime configuration. The current prototype provides QiskitAdapter and CirqAdapter, which delegate execution to Qiskit Aer and Cirq Simulator, respectively.

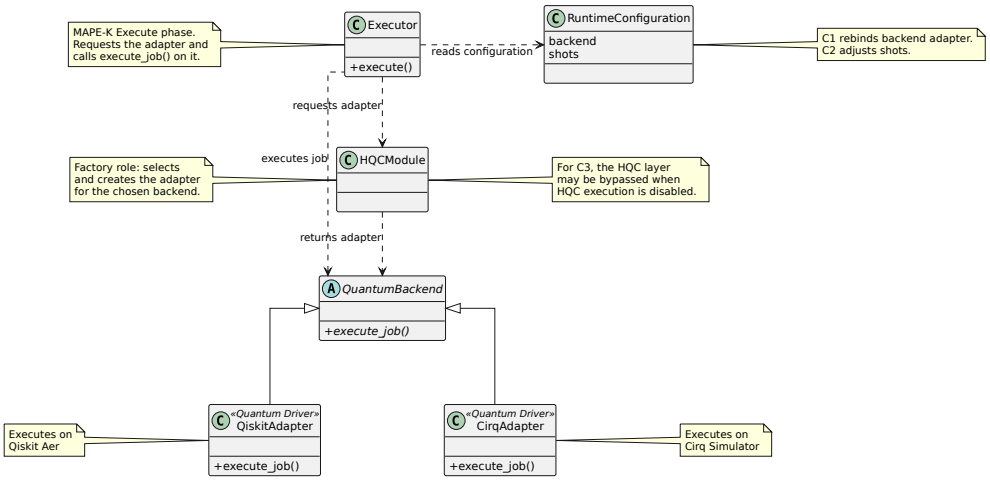


Fig. 8 Class-level view of the HQC execution abstraction in the QUASAR prototype.

Execute passes the validated backend and execution parameters to this layer, and the available results and runtime evidence are returned to the adaptation control. C1 exercises adapter rebinding, C2 updates the shot configuration while retaining the selected backend, and C3 may bypass the HQC layer when quantum execution is disabled.

The two adapters encapsulate heterogeneous optimization implementations: the Qiskit realization uses a TSP (Traveling Salesperson Problem)-oriented QAOA workload, whereas the Cirq realization uses a Max-Cut-oriented VQE workload. Consequently, C1 provides evidence of software-level backend rebinding and execution redirection, but not of semantic-preserving migration of an identical workload across SDKs or physical quantum providers.

For C1 (D1), the prototype evaluates backend status and latency against the configured operational conditions. Following deterministic compatibility validation, HQCModule selects the accepted QiskitAdapter or CirqAdapter. For C2 (D4), simulated noise or execution-quality information and the current execution configuration inform a candidate shot-count adjustment, which is applied only when the configured parameter bounds are satisfied. For C3 (D6), application context is evaluated against the HQC applicability policy to retain, enable,

or bypass quantum execution within the predefined workflow. These cases exercise distinct runtime decisions separately rather than a combined multi-decision adaptation problem.

6.6 Adaptation Trace and Audit Evidence

Each execution records adaptation provenance using the chain: Observation → Affected Decision → Candidate → Validation → Enactment/Reject. The available trace includes the relevant initial configuration, runtime observations, candidate rationale, validation outcome, enacted configuration when applicable, timing evidence when available, and the resulting execution state.

These records are distributed across runtime state, configuration information, execution metrics, and audit logs. Importantly, the LLM-generated candidate and the configuration ultimately enacted are represented as distinct trace elements, allowing rejected candidates to remain observable as guardrail interventions rather than disappearing from the execution history.

The adaptation traces provide the empirical interface between implementation and the evaluation in Section 7, supporting the reconstruction of the bounded adaptation path and the analysis of validation and runtime overhead. The complete trace schema, case-specific reconstructions, and field-level evaluation mapping are provided in the companion technical artifact (Candia et al. 2026).

7 Evaluation and Results

The evaluation examines whether the PoC implementation described in Section 6 provides sufficient evidence to operationalize the bounded QUASAR adaptation space defined by C1–C3. The objective is not to establish industrial effectiveness, superiority over alternative adaptation approaches, or performance under real quantum hardware conditions.

The evaluation uses the execution and audit evidence produced by the prototype. The original experimental campaign was organized using four operational scenario groups—Normal Operation, High Demand, Environmental Crisis, and Infrastructure Failure—whereas Section 5.3 organizes the QUASAR running scenario around the architectural cases C1–C3. Consequently, the evaluation distinguishes representative case-specific evidence from aggregate prototype-level results rather than retrospectively assigning every original execution to C1–C3.

7.1 Evaluation Questions

To assess the implemented subset of QUASAR, three evaluation questions (EQ) examine complementary dimensions of the proposed contributions: runtime adaptation feasibility (EQ1), deterministic guardrail enforcement (EQ2), and operational overhead and decision traceability (EQ3). Together, they provide preliminary evidence of the feasibility of the integrated approach, rather than a comprehensive validation of the reference architecture.

EQ1 – Adaptation feasibility. Can the prototype process the runtime conditions represented by C1–C3 through the implemented QUASAR workflow and enact the corresponding bounded reconfiguration when an admissible adaptation is available?

EQ2 – Guardrail effectiveness. To what extent do deterministic guardrails prevent invalid LLM-generated candidate configurations from reaching execution?

EQ3 – Runtime overhead and traceability. What runtime overhead is introduced by the implemented adaptation mechanism, and to what extent can an adaptation decision be reconstructed from the recorded runtime evidence?

7.2 Evaluation Protocol and Metrics

The evaluation uses the prototype adaptation traces described in Section 6.6. Case-specific evidence is examined at the adaptation-cycle level, whereas aggregate validation outcomes are analyzed at the execution level. Table 11 relates the evaluation questions to the available evidence and corresponding metrics or observations.

Table 11: Evaluation questions, evidence, and metrics.

ID	Aspect	Evidence	Metric / observation
EQ1	Adaptation feasibility	Runtime condition, affected variation point, validation outcome, and resulting configuration	Completion of the bounded adaptation path and observed enactment or configuration preservation
EQ2	Guardrail effectiveness	LLM candidates, deterministic validation outcomes, identified invalid candidates, and execution records	Observed blocked/non-blocked execution outcomes; Invalid Candidate Blocking Rate (ICBR); Invalid Configurations Enacted (ICE)
EQ3	Runtime overhead	Adaptation-cycle and LLM-related timing evidence	Adaptation latency and reported reasoning overhead
EQ3	Traceability and auditability	Observation, candidate, validation, enactment, and final-state evidence	Availability of the evidence chain required to reconstruct an adaptation cycle

Adaptation feasibility (EQ1) is assessed descriptively through the observed relationship between runtime conditions, affected decisions, validation outcomes, and resulting configurations. This evidence supports the assessment of bounded adaptation behavior in C1–C3 without establishing a quantitative adaptation success rate or improvements in application-level utility.

To specifically characterize whether candidates identified as invalid crossed the validation boundary, we derive the Invalid Candidate Blocking Rate (ICBR):

$$ICBR = \frac{N_{\text{invalid candidates blocked}}}{N_{\text{invalid candidates}}} \tag{1}$$

We also report the number of Invalid Configurations Enacted (ICE):

$$ICE = N_{\text{invalid configurations enacted}} \tag{2}$$

ICBR and ICE are validation-boundary metrics derived in this study from the available audit evidence. ICBR measures the proportion of identified invalid candidates prevented from reaching execution, whereas ICE records whether any candidate identified as invalid was nevertheless enacted.

For adaptation cycles resulting in an enacted reconfiguration, adaptation latency is defined as:

$$T_{\text{adapt}} = t_{\text{enactment}} - t_{\text{trigger}} \tag{3}$$

The prototype does not expose equally fine-grained timestamps for every internal MAPE-K operation; therefore, timing evidence is interpreted descriptively rather than through inferential statistical analysis.

7.3 Case-Based Results

This subsection presents the observed results for C1–C3, considering the evidence available for each case and their differing experimental coverage.

7.3.1 C1 – Infrastructure degradation and backend rebinding.

C1 is supported by controlled executions in which backend degradation is represented through excessive simulated latency. The prototype uses a latency threshold of 60 s for the corresponding infrastructure-failure condition. One documented adaptation trace reports a simulated latency of 173 s, after which the system identifies the current backend as unsuitable and redirects execution from the initial Qiskit profile to the alternative Cirq profile.

The original campaign further reports that when injected waiting times violated this latency condition, migration toward the secondary backend was performed for the evaluated infrastructure-degradation executions.

C1 provides preliminary feasibility evidence for software-level backend-adapter rebinding in response to simulated infrastructure degradation. The prototype redirects execution from the initially selected adapter to an alternative configured adapter when the latency condition is violated. However, the Qiskit and Cirq adapters encapsulate different optimization workloads (TSP/QAOA and Max-Cut/VQE, respectively). Therefore, this result demonstrates execution redirection rather than functionally equivalent workload migration or comparative backend performance.

7.3.2 C2 – Quantum-execution degradation and parameter adjustment

C2 is supported by the mechanism-oriented validation in which simulated noise or execution-quality degradation provides the I3 runtime evidence, and the current shot configuration provides the corresponding I5 execution information.

Together, these inputs provide the context for assessing whether shot adjustment (D4) is warranted before candidate generation and deterministic bounds validation. The prototype completes the MAPE-K cycle by collecting execution information, identifying the adaptation condition, generating a candidate parameter adjustment, validating the resulting configuration, and applying the updated parameter through the HQC execution layer.

This result provides evidence for bounded parameter-level reconfiguration while preserving the application functionality, optimization concerns, and overall workflow structure. It does not establish that a particular increase or decrease in the number of shots necessarily improves quantum-result quality. The evidence supports the ability to reconsider and adjust the shot count under a simulated execution condition; demonstrating a causal improvement in solution quality would require an independent experiment controlling noise models, repeated executions, and shot values.

7.3.3 C3 – Context- and policy-driven HQC activation.

C3 is supported by executions in which problem complexity and domain-level conditions affect whether the HQC capability should participate in the workflow. In the original High Demand scenario, variations in problem complexity are used to determine whether the activation of the HQC module is justified; the reported policy uses a complexity condition to differentiate between HQC-enabled and purely classical execution. Domain restrictions are also exercised independently in the Environmental Crisis scenario, where contextual conditions constrain which services may remain active.

For QUASAR, the relevant result is that runtime context can affect a capability-level configuration decision rather than only the selection of a quantum backend. The adaptation remains bounded because the HQC-enabled and bypassed alternatives belong to the deployed configuration space; the system does not synthesize a new workflow structure at runtime.

Together, these cases provide heterogeneous but complementary feasibility evidence for infrastructure-, parameter-, and capability-level adaptation. They should not be interpreted as symmetric experiments or as evidence of semantic backend equivalence or improved quantum-result quality. Scenario-specific evidence for C1–C3 is reported separately from the aggregate experimental outcomes, which primarily inform the assessment of guardrail enforcement and operational overhead.

Answer to EQ1.

The results across C1–C3 provide feasibility evidence for runtime adaptation within the implemented variability subset, supporting the partial operationalization of the runtime-variability requirement.

7.4 Aggregate Results and Cross-Case Analysis

The C1–C3 results provide case-specific feasibility evidence, while the aggregate outcomes of the original experimental campaign are used to assess candidate validation and guardrail intervention.

The guardrail classes defined in Section 4.6 are not uniformly instantiated or empirically exercised by the current PoC. Table 12 summarizes the relationship between the architectural guardrail taxonomy, its prototype realization, the C1–C3 adaptation cases, and the evidence available for assessing implementation coverage. The complete guardrail implementation and evidence mapping, including the Specified–Implemented–Evaluated distinction for G1–G6, is provided in the QUASAR Architecture and Technical Companion (Sepúlveda et al. 2026b).

Table 12: Guardrail coverage in the current PoC.

ID	Prototype status	Empirical coverage
G1	Instantiated variability/configuration constraints	Exercised for the implemented subset; non-exhaustive
G2	Partial execution-stack compatibility	Primarily exercised through C1/D1
G3	Architecturally defined	Not independently exercised
G4	Partial policy and operational validation	Strongest candidate-level evidence in C3
G5	Architecture/workflow consistency mainly preserved by construction	Indirectly exercised
G6	Deterministic shot bounds	Directly exercised through C2/D4

The original evaluation reports 32 non-blocked outcomes and 10 blocked outcomes across 42 controlled executions. Because the available evidence does not explicitly establish that every execution generated exactly one LLM candidate, these values are interpreted at the execution-outcome level rather than as candidate acceptance and rejection rates: 32 of the 42 executions (76.2%) proceeded without a guardrail block, whereas 10 executions (23.8%) contained an LLM-generated candidate configuration that violated implemented variability constraints and was blocked by deterministic validation.

The audit analysis is particularly important for interpreting these values. The 10 blocked events correspond to LLM-generated candidate configurations that violated implemented

variability constraints, including incompatibilities between the selected backend and the required quantum algorithm. The deterministic guardrails intercepted these candidates and prevented the corresponding invalid configurations from being applied.

The available audit records do not provide a complete per-rule or per-guardrail classification of all rejected candidates. Consequently, the 10 blocked candidates and the resulting aggregate ICBR cannot be decomposed into independent effectiveness values for G1–G6.

Accordingly, all candidate configurations identified as invalid in the controlled campaign were blocked: $ICBR = 10/10 = 100\%$. No candidate identified as invalid by the implemented guardrails was enacted: $ICE = 0$.

ICBR and ICE are derived metrics introduced in this study rather than metrics explicitly named in the original prototype evaluation. Their values follow from the reported audit evidence: 10 invalid LLM-generated candidates were identified, and all were intercepted before application. This interpretation is also consistent with the reported structural-integrity result that the resulting configurations satisfied the implemented feature-model constraints throughout the evaluated campaign.

Table 13 summarizes these results using terminology aligned with the QUASAR validation boundary.

Table 13: Aggregate guardrail-related results from the controlled prototype campaign.

Observed result	Count	Rate
Controlled executions	42	100%
Non-blocked execution outcomes	32	76.2%
Executions containing an invalid candidate blocked by guardrails	10	23.8%
Identified invalid candidates blocked	10/10	100%
Invalid configurations enacted	0	0%

These results provide direct evidence for the architectural separation between probabilistic candidate generation and deterministic validity enforcement. The occurrence of invalid LLM-generated candidates shows that contextual reasoning alone was insufficient to ensure conformance with the implemented variability model, while deterministic validation prevented the identified invalid candidates from reaching enactment.

The 100% ICBR must not be interpreted as complete or universal guardrail coverage. It applies only to the invalid candidates observed during the 42 controlled executions and to the subset of constraints implemented by the prototype. The result does not establish that every possible invalid QUASAR configuration would be detected, nor that the same blocking behavior would necessarily be obtained in other HSS domains, providers, or variability models.

Answer to EQ2.

All ten candidates identified as invalid by the implemented checks were blocked before enactment. This supports the separation between LLM-supported proposals and deterministic admission to execution within the evaluated subset.

7.4.1 Runtime overhead and traceability

The prototype reports an adaptation latency of 1.5–3 s per decision cycle. LLM inference is considered a major contributor to this latency, although the available timing evidence does not allow its contribution to be isolated quantitatively. This latency is compatible with coarse-grained orchestration and runtime configuration decisions such as backend selection, capability

activation, or parameter adjustment, where adaptation operates at software-management timescales.

The observed latency does not support the use of the current approach for low-level real-time quantum control, pulse-level adaptation, or quantum error correction mechanisms requiring reactions on substantially shorter timescales. The empirical scope of QUASAR should therefore be understood as runtime software orchestration and configuration adaptation rather than physical-layer quantum control.

Answer to EQ3.

The available traces support adaptation-level reconstruction of candidate validation and execution outcomes. However, incomplete per-phase timestamps and per-rule validation records prevent fine-grained reconstruction of every internal operation. Together with the adaptation latency reported above, these observations delimit the prototype's current overhead and traceability evidence.

These findings are limited to the implemented variability, the constraint subset, the controlled simulation environment, and the available evidence for C1–C3.

8 Threats to Validity

This section examines threats to the validity of the empirical findings reported in Section 7. It considers construct, internal, external, and conclusion validity following empirical software engineering guidelines (Wohlin et al. 2012), along with reliability and study documentation concerns (Runeson and Höst 2009). Given the architectural contribution, artifact evaluation also informs the assessment (Wieringa 2014). For QUASAR, this includes implementation fidelity, scenario selection, guardrail coverage, and evaluation scope.

8.1 Construct Validity

Construct validity concerns whether the evaluation measures adequately represent the phenomena under study. The operationalization of adaptation feasibility (EQ1) relies on observable adaptation behavior rather than direct measurements of configuration optimality, execution-quality improvements, or application-level utility. Consequently, the available evidence supports a preliminary feasibility assessment but cannot establish the operational benefits of the resulting adaptations.

Aggregate execution outcomes must also be interpreted carefully. The observed blocked and non-blocked executions are neither conventional failure/success rates nor candidate rejection/acceptance rates because a one-to-one mapping between executions and generated candidates has not been established. ICBR and ICE, therefore, characterize only whether candidates identified as invalid crossed the deterministic validation boundary.

LLM usefulness and configuration validity are likewise distinct constructs. Candidate acceptance does not directly measure the quality of LLM reasoning, while guardrail acceptance establishes conformance only with the implemented constraints and not the overall functional correctness of the resulting computation.

The runtime conditions used in C1–C3 are controlled proxies: injected latency, simulated noise, and execution-quality indicators, as well as problem complexity, which represent infrastructure, execution, and context-driven changes. They enable controlled evaluation of the corresponding mechanisms but do not capture the full behavior of production quantum environments. In particular, C2 does not establish that changing the number of shots causally improves quantum-result quality. Finally, binding classes, decision–information relationships,

and runtime treatments involve modeling judgment. The paper defines these categories explicitly and applies them consistently to C1–C3, but alternative HSS architectures may classify some concerns differently.

8.2 Internal Validity

The QUASAR artifacts were derived from prior SPL-HQC and DSPL work, the literature synthesis, and the prototype scenarios. Their selection and classification, therefore, retain a degree of researcher judgment. Moreover, the prototype operationalizes only the subset required by C1–C3, and several architectural responsibilities are distributed across configuration structures, policies, control logic, and runtime state rather than being implemented as one-to-one modules. The evidence consequently demonstrates partial operationalization of QUASAR rather than implementation completeness.

The scenarios and runtime triggers were deliberately selected or injected to exercise known adaptation mechanisms. This improves repeatability and observability, but it may favor conditions for which the prototype was already prepared; unexpected or more heterogeneous disturbances could produce different behavior.

LLM-generated candidates may vary with prompt formulation, context, model version, temperature, and other inference parameters. Guardrail behavior also depends on the subset of constraints manually encoded in the prototype: missing or incorrectly represented rules could leave an invalid configuration undetected. Deterministic validation limits the consequences of LLM variability but does not eliminate either source of uncertainty. In addition, the heterogeneous realization of G1–G6 prevents individual blocked outcomes from being attributed unambiguously to independent guardrail classes.

C1 contains an additional implementation confound: the Qiskit and Cirq adapters expose a common backend abstraction but encapsulate heterogeneous optimization workloads. The observed migration, therefore, supports software-level rebinding and execution redirection, but computational differences cannot be attributed solely to a backend change. Accordingly, C1 is not used to claim semantic-preserving migration or comparative backend performance.

8.3 Conclusion Validity

The aggregate evaluation comprises 42 controlled executions, which are sufficient to characterize prototype behavior descriptively but not to support broad statistical claims about QUASAR, LLM behavior, or guardrail effectiveness across HSS populations. We therefore avoid inferential statistical conclusions.

The original campaign was organized into four operational scenario groups, whereas this paper structures the architectural analysis around C1–C3. Because these classifications are related but not equivalent, aggregate campaign results are retained at the prototype level, and representative mechanism-specific evidence is used for the three QUASAR cases. The resulting C1–C3 evidence is heterogeneous and should not be interpreted as three symmetric experiments with equivalent sample sizes, treatments, or statistical power.

The strongest quantitative result concerns the observed validation boundary. ICBR = 100% and ICE = 0 apply only to candidates identified as invalid in the controlled campaign and to the implemented constraint subset. The reported 23.8% is the proportion of executions containing such a candidate, not a candidate rejection rate over 42 confirmed candidates. Because the audit data lack complete per-rule attribution, the aggregate result also cannot be decomposed into independent effectiveness estimates for G1–G6.

The reported adaptation latency is descriptive. Uniform per-phase timestamps, timing distributions, confidence intervals, and comparative measurements are unavailable; therefore,

the evidence supports only the interpretation that the prototype operates at a coarse-grained orchestration timescale.

Finally, the absence of comparative baselines limits causal attribution. Because the prototype is not compared with static configurations, rules-only adaptation, unguarded LLM planning, or alternative self-adaptation architectures, the evaluation supports the feasibility and observed behavior of the integrated approach but does not isolate the incremental contribution of the LLM, MAPE-K, variability model, or guardrails.

8.4 External Validity

The evaluated PoC instantiates only backend selection, shot adjustment, and HQC activation or bypass; additional decisions in the broader QUASAR architecture remain unevaluated. The prototype has also not been subjected to production-scale workloads, long-running operation, concurrent adaptation requests, large variability models, or industrial provider failures. Consequently, C1–C3 should not be interpreted as validation of the complete QUASAR adaptation space or its scalability under production conditions.

The evaluation uses Qiskit Aer and Cirq Simulator rather than physical QPUs. Controlled simulation supports repeatable injection of latency- and noise-related conditions but does not reproduce provider queue dynamics, calibration drift, device noise, connectivity changes, quotas, transient failures, or other characteristics of operational quantum-cloud environments.

Generalization is also affected by provider and SDK heterogeneity. Although the Quantum-Backend abstraction isolates part of the execution-specific logic, quantum platforms expose different APIs, execution models, metadata, queue mechanisms, and calibration information. Further evaluation is required before assuming equivalent adaptation and validation behavior across additional providers and software stacks.

The application setting also represents a limited class of HSS. Other domains may introduce different quality attributes, policies, timescales, workflow structures, or adaptation concerns, such as algorithm substitution, transpilation changes, circuit partitioning, or multi-provider orchestration.

The observed adaptation timescale further limits generalization to coarse-grained software orchestration and does not support conclusions about pulse-level control or real-time quantum error correction.

8.5 Reliability and Reproducibility

Reproducibility is affected by both the evolving quantum software stack and the external LLM service. Changes in Qiskit, Cirq, dependencies, APIs, model versions, prompts, inference settings, or service-side behavior may alter execution or candidate generation. Exact repetition of the LLM step, therefore, cannot be assumed, even when the remaining prototype configuration is held constant.

The replication package preserves the source code, dependency information, scenario definitions, variability and policy artifacts, available prompts and model settings, execution evidence, and adaptation traces used in this study (Candia et al. 2026). The package is archived as an immutable release to support the inspection and replication of the reported PoC evidence.

Adaptation traces mitigate this risk by preserving the available runtime context, candidate, validation outcome, enacted configuration, and execution result. They allow the variability introduced by candidate generation to be distinguished from deterministic validation and enactment behaviors.

Trace completeness is nevertheless uneven: not all executions expose equally detailed timestamps, prompt metadata, or per-rule validation records. Reproduction is therefore stronger at the adaptation-cycle level than for every internal operation.

Taken together, these threats restrict the conclusions regarding the feasibility of the implemented QUASAR subset and the observed behavior of its deterministic validation boundary. They do not support claims of complete artifact or guardrail coverage, semantic backend equivalence, industrial readiness, or general effectiveness across HSS domains and execution environments.

9 Discussion

This section interprets the PoC findings in light of the threats to validity discussed in the preceding section. It examines their architectural implications, the role of LLM-supported reasoning, and the trade-offs involved in engineering runtime adaptation for HSS.

9.1 Model-Based and Bounded Runtime Adaptation

The case-based results support the interpretation of QUASAR as a model-based adaptation architecture rather than as an isolated scheduler, backend selector, or collection of runtime heuristics. Adaptation explicitly evaluates modeled runtime-controllable decisions against observed evidence and explicit constraints and enacts an admissible configuration change when warranted.

A central implication is that this adaptation is intentionally bounded: runtime observations do not authorize arbitrary modification of the HSS but may trigger adaptation assessments only for variation points previously identified as runtime-controllable. The contribution, therefore, lies less in maximizing reconfiguration freedom than in making explicit *what* may change, *under which evidence*, and *subject to which constraints*. This also clarifies the relationship between design-time and runtime variability: structural validity defines the admissible configuration space, whereas runtime evidence determines whether a currently valid binding remains operationally suitable.

The corresponding trade-off is additional modeling and maintenance effort. Explicit runtime decisions, information dependencies, constraints, policies, and validation rules require more engineering than provider-specific scripts or embedded heuristics. For stable adaptation tasks with few alternatives and fixed thresholds, a simpler rule-based controller may be sufficient. The architectural rationale for QUASAR is to make variability, decision context, validation, and provenance explicit across adaptation concerns; the present evaluation does not establish when these benefits outweigh the additional engineering effort.

9.2 LLM-Supported Reasoning and Deterministic Guardrails

The observed invalid candidates reinforce one of QUASAR's central design decisions: contextual plausibility is not equivalent to configuration validity. LLM-supported reasoning can produce apparently reasonable adaptations that remain inconsistent with modeled variability or execution constraints, which justifies an independent deterministic validation boundary before enactment. From this perspective, rejection is not necessarily a failure of adaptation: preventing an inadmissible candidate from becoming active is itself a correct outcome of the guarded workflow.

The LLM is investigated as a mechanism for contextual candidate generation and explanation within an explicitly constrained adaptation process. The present cases establish the

feasibility of this integration but do not demonstrate that an LLM is necessary or preferable to deterministic candidate-selection rules. In particular, blocking invalid LLM-generated candidates supports the validation boundary, not the added value of the LLM itself. Whether contextual reasoning provides benefits that justify its latency, variability, and integration effort remains an empirical question.

The heterogeneous coverage of G1–G6 means that the observed blocking results support the implemented validation boundary without establishing independent effectiveness for every guardrail class or a general safety guarantee.

9.3 Traceability and Auditability of Runtime Adaptation

The findings suggest that recording both enacted configurations and rejected LLM-generated candidates allows runtime adaptation to be examined as an auditable engineering process rather than merely as a sequence of configuration changes.

Such provenance can support debugging, post-hoc explanation, reproducibility, and the evolution of adaptation assets. Recurrent rejected candidates, for example, may expose missing context, obsolete variability constraints, or repeated incompatibilities that should be represented more explicitly in the runtime knowledge. The current evidence demonstrates adaptation-level auditability rather than complete event-level provenance; finer-grained instrumentation remains future work.

9.4 Scalability, Evolution, and Maintainability of QUASAR Artifacts

The explicit QUASAR artifacts improve control and traceability, but they also introduce an evolution problem: runtime decisions, information mappings, variability constraints, policies, guardrails, and provenance structures must remain consistent as providers, SDKs, execution capabilities, and application requirements change. This challenge is amplified because conceptual responsibilities do not necessarily map one-to-one to software modules; related knowledge may be distributed across configuration structures, policies, control logic, runtime state, and audit records. Maintainability, therefore, concerns consistency across artifacts as much as the size of any individual model or catalog. The consistency problem becomes more significant as the adaptation space and the number of dependencies among variation points, observations, policies, and guardrails increase.

Future implementations should therefore support consistency management and semi-automated evolution of these assets; for example, by deriving runtime variation points from SPL-HQC models, importing provider capabilities and metadata, detecting platform changes, and exploiting adaptation traces to identify missing or obsolete mappings and guardrails.

9.5 Implications for Quantum DevOps and HSS Engineering

The results suggest that runtime adaptation in HSS should be addressed as a software engineering concern spanning variability management, observability, decision control, validation, execution, and operational feedback. From a Quantum DevOps perspective, runtime evidence is therefore more than telemetry; it becomes explicit Knowledge used to inform runtime configuration decisions in the hybrid system.

The evaluated cases further indicate that this concern spans infrastructure, execution-configuration, and application-capability levels; runtime HSS adaptation is therefore broader than quantum-device selection alone. The simulator-based evaluation supports preliminary feasibility at the software-adaptation level, but does not establish operational effectiveness

under real quantum-cloud conditions, including provider queue dynamics and physical-QPU variability.

More broadly, the prototype illustrates how DSPL principles, MAPE-K control, runtime variability knowledge, LLM-supported reasoning, and deterministic validation can be combined in an HSS engineering process in which adaptation decisions remain explicit, constrained, inspectable, and auditable. The evidence is preliminary, but it provides a software-engineering alternative to delegating runtime adaptation entirely to provider-specific logic or unconstrained reasoning mechanisms.

10 Conclusions

This paper presented QUASAR, a Dynamic Software Product Line architecture for model-based runtime self-adaptation in HSS. QUASAR extends design-time variability management toward runtime adaptation by combining an explicit runtime variability space, decision and information assets, MAPE-K control, bounded LLM-supported candidate reasoning, deterministic constraint-based guardrails, and adaptation traces. A central design principle is that runtime reconfiguration remains bounded by previously modeled variation points and constraints, with candidate adaptations subject to deterministic validation before enactment.

A proof-of-concept implementation operationalized a bounded subset of the architecture through three representative adaptation cases: backend rebinding, execution-parameter adjustment, and HQC activation or bypass. The controlled evaluation provides preliminary evidence that these adaptation concerns can be processed and recorded as auditable adaptation traces. Some controlled executions contained an LLM-generated candidate configuration that violated implemented variability constraints, and all the identified invalid candidates were intercepted before enactment. The reported adaptation latency places the current realization at the level of coarse-grained HSS orchestration. These results are restricted to the implemented constraint subset, the controlled simulation environment, and proof-of-concept scenarios.

For the research community, QUASAR provides an explicit architectural basis for investigating HSS adaptation while keeping candidate generation, configuration validation, and execution outcomes distinguishable and traceable. For practitioners, it offers a reference for structuring adaptation logic around existing hybrid workflows, with explicit constraints on runtime reconfiguration.

Future work will extend the catalogs towards capacity-aware classical–quantum execution choices, problem partitioning, and adaptation of variational parameters or structures, along with the required observations and validation rules. Changes to objectives or weights will require explicit conditions that preserve the intended application semantics. We will also investigate catalog evolution and consultative LLM support in Monitor and Analyze while retaining explicit trigger criteria and deterministic pre-enactment validation. Further evaluation will combine interacting adaptation concerns and compare candidate-generation strategies under equivalent observations, constraints, and validation checks, particularly LLM-supported and rules-only planning, along with larger workloads and physical-QPU execution.

Statements and Declarations

Funding. Samuel Sepúlveda acknowledges support from the Agencia Nacional de Investigación y Desarrollo (ANID), Chile, through FONDECYT de Iniciación en Investigación (Project No. 11240702).

Ricardo Pérez-Castillo acknowledges partial support from the Smooth project (PID2022-137944NB-I00), funded by MICIU/AEI/10.13039/501100011033/PRTR, European Union,

and the SINERGIA project (2025-GRIN-38310), under the UCLM Research Plan, co-funded by the European Regional Development Fund (ERDF).

Competing Interests. The authors declare that they have no competing financial or non-financial interests relevant to the content of this article.

Data and Code Availability. The source code, configuration, and policy artifacts, scenario definitions, and available experimental evidence supporting this study are archived in the QUASAR Replication Artifact, available at https://doi.org/10.5281/zenodo.22926163. Additional architectural documentation and literature evidence are available in the QUASAR Architecture and Technical Companion (https://doi.org/10.5281/zenodo.22925733) and the QUASAR Literature Evidence Report (https://doi.org/10.5281/zenodo.22925564).

Author Contributions (CRediT taxonomy).

Samuel Sepúlveda: Conceptualization, Methodology, Formal analysis, Investigation, Writing - original draft, Writing - review & editing, Visualization, Supervision, Project administration, Funding acquisition.

Ricardo Pérez-Castillo: Conceptualization, Methodology, Validation, Writing - review & editing.

Jesús Tapia-Martin: Writing - review & editing, Visualization.

Sebastián Candia: Writing - original draft, Data curation, Software.

Claudio Navarro: Conceptualization, Formal analysis, Writing - original draft, Writing - review & editing.

Ania Cravero: Formal analysis, Writing - review & editing.

All authors reviewed and approved the final version of the manuscript.

Generative AI Assistance in Manuscript Preparation. During the preparation and revision of this manuscript, ChatGPT (OpenAI, GPT-5.6 and GPT-6 models) was used as an editorial support tool to improve academic writing, review manuscript structure, identify redundancies and potential inconsistencies, and support the consolidation of revisions across sections. AI-assisted reviews also supported the assessment of internal consistency among the research objectives, stated contributions, evaluation questions, results, discussions, and conclusions.

The original research idea, conceptual framework, architectural proposal, methodological design, prototype implementation, experimental work, and scientific contributions were conceived and developed by the authors independently of ChatGPT. Generative AI assistance was limited to reviewing and improving the presentation, organization, and internal consistency of author-developed research content.

All AI-assisted suggestions were critically reviewed and verified by the authors, who retain full responsibility for the originality, scientific validity, interpretation of results, and final content of the manuscript.

Ethics Approval and Consent to Participate. Not applicable. This study did not involve human participants or animals.

Acknowledgements

The authors acknowledge the recognition of the Ibero-American Programme for Science and Technology for Development (CYTED) through RIPAISC, the Ibero-American Network for the Advancement of Quantum Software Engineering (525RT0174). Ricardo Pérez-Castillo is supported by the SMOOTH (PID2022-137944NB-I00) project, funded by

MICIU/AEI/10.13039/501100011033/PRTR, European Union; by the 2025-GRIN-38310 (SINERGIA) project, under the UCLM Research Plan, co-funded (85%) by the European Regional Development Fund (ERDF).

References

Aguayo, O., Sepúlveda, S.: Variability management in dynamic software product lines for self-adaptive systems—a systematic mapping. Applied Sciences **12**(20), 10240 (2022) https://doi.org/10.3390/app122010240

Aguayo, O., Escobar, F., Vasquez, F., Sepúlveda, S., Mazo, R.: Fmweb-k: A framework for variability management in dynamic software product lines. In: Rocha, Á., Vaseashta, A. (eds.) Developments and Advances in Defense and Security, pp. 357–367. Springer, Singapore (2025)

Alves, L., De Oliveira, G.L.B., Chagas, M., Bandeira, A., Maia, P.H., Rocha, C.: Maper: Extending mape-k with llm-based reasoning to manage unanticipated situations in self-adaptive systems. In: Proceedings of the 21st International Conference on Software Engineering for Adaptive and Self-Managing Systems. SEAMS '26, pp. 239–250. Association for Computing Machinery, New York, NY, USA (2026). https://doi.org/10.1145/3788550.3794886

Benavides, D., Segura, S., Ruiz-Cortés, A.: Automated analysis of feature models 20 years later: A literature review. Information systems **35**(6), 615–636 (2010) https://doi.org/10.1016/j.is.2010.01.001

Benavides, D., Sundermann, C., Feichtinger, K., Galindo, J.A., Rabiser, R., Thüm, T.: Uvl: Feature modelling with the universal variability language. Journal of systems and software **225**, 112326 (2025) https://doi.org/10.1016/j.jss.2024.112326

Benitez, F., Galindo, J., Romero, D., Benavides, D.: Uvl web-based editing and analysis with flamapy.ide. In: Procs. of 19th Int. Working Conference on Variability Modelling of Software-Intensive Systems (VaMoS'25), pp. 121–125. ACM, NY, USA (2025)

Burgholzer, L., Echavarria, J., Hopf, P., Stade, Y., Rovara, D., Schmid, L., Kaya, E., Mete, B., Farooqi, M.N., Chung, M., *et al.*: The munich quantum software stack: Connecting end users, integrating diverse quantum technologies, accelerating hpc. In: Proceedings of the Supercomputing Asia and International Conference on High Performance Computing in Asia Pacific Region, pp. 55–67 (2026). https://doi.org/10.1145/3773656.3773669

Candia, S., Sepúlveda, S., Tapia-Martin, J.: QuASAr: Replication Artifact for Runtime Self- Adaptation in Hybrid Quantum–Classical Systems. Zenodo (2026). https://doi.org/10.5281/zenodo.22926163

Cheng, B.H.C., Lemos, R., Giese, H., Inverardi, P., Magee, J., *et al.*: Software engineering for self-adaptive systems: A research roadmap. In: Software Engineering

for Self-Adaptive Systems. Lecture Notes in Computer Science, vol. 5525, pp. 1–26. Springer, Berlin, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02161-9_1

Chundury, S., Kim, S., Gopalakrishnan Meena, M., Lu, C., Suh, I.-S., Mueller, F.: SeleQtor: Intelligent quantum backend selection and runtime estimation. In: 2026 IEEE International Conference on Quantum Computing and Engineering (QCE). IEEE, Toronto, ON, Canada (2026). QSYS Technical Paper, Paper 654. https://qce.quantum.ieee.org/2026/qce26-schedule/qsys-schedule/

Clements, P.C., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley, NJ, USA (2001)

Cranganore, S.S., De Maio, V., Brandic, I., Deelman, E.: Paving the way to hybrid quantum–classical scientific workflows. Future Generation Computer Systems **158**, 346–366 (2024) https://doi.org/10.1016/j.future.2024.04.030

Czarnecki, K., Wasowski, A.: Feature diagrams and logics: There and back again. In: 11th International Software Product Line Conference (SPLC 2007), pp. 23–34. IEEE, Kyoto, Japan (2007). https://doi.org/10.1109/SPLINE.2007.24

De Stefano, M., Pecorelli, F., Di Nucci, D., Palomba, F., De Lucia, A.: Software engineering for quantum programming: How far are we? Journal of Systems and Software **190**, 111326 (2022) https://doi.org/10.1016/j.jss.2022.111326

Donakanti, R., Jain, P., Kulkarni, S., Vaidhyanathan, K.: Reimagining Self-Adaptation in the Age of Large Language Models . In: 2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C), pp. 171–174. IEEE Computer Society, Los Alamitos, CA, USA (2024). https://doi.org/10.1109/ICSA-C63560.2024.00036

Gheorghe-Pop, I.D., Tcholtchev, N., Ritter, T., Hauswirth, M.: Quantum devops: Towards reliable and applicable nisq quantum computing. In: 2020 IEEE Globecom Workshops (GC Wkshps), pp. 1–6. IEEE, Taipei, Taiwan (2020). https://doi.org/10.1109/gcwkshps50303.2020.9367411 . IEEE

Giortamis, E., Romao, F., Tornow, N., Lugovoy, D., Bhatotia, P.: Qonductor: A cloud orchestrator for quantum computing. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. SC'25, pp. 728–745. Association for Computing Machinery, New York, NY, USA (2025). https://doi.org/10.1145/3712285.3759785

Hachicha, N., Ghezala, M., Trabelsi, Z.: Formal modeling and verification of mape-k feedback loops in self-adaptive systems. Science of Computer Programming **121**, 82–104 (2016) https://doi.org/10.1016/j.scico.2015.10.007

Hallsteinsen, S., Hinchey, M., Park, S., Schmid, K.: Dynamic software product lines. Computer **41**(4), 93–95 (2008) https://doi.org/10.1109/mc.2008.117

Hopf, P., Quetschlich, N., Wille, R.: Efficient priority-aware scheduling for parallel circuit execution in distributed multi-qpu systems. In: 2026 IEEE International Conference on Quantum Computing and Engineering (QCE). IEEE, Toronto, ON, Canada (2026). DQC-SI Workshop. https://www.cda.cit.tum.de/files/eda/2026_qce_efficient_scheduling_for_multi_qpu_systems.pdf

Huenchuleo, N., Sepúlveda, S., Fernández, A.: QC-MM: A metadata and schema model for traceable quantum-circuit experiments. Applied Sciences **16**(13), 6346 (2026) https://doi.org/10.3390/app16136346

IBM Corporation: An Architectural Blueprint for Autonomic Computing. White paper, IBM White Paper (2006). V7. https://ibm.com

Jaleel, K.A., Sharon, R.G., Nivi, V.: Hybrid quantum computing workflow optimization. In: 2026 International Conference on Innovative Trends in Information Technology (ICITIIT), pp. 1–6 (2026). https://doi.org/10.1109/ICITIIT68860.2026.11499461 . IEEE

Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-oriented domain analysis (foda) feasibility study. Technical Report CMU/SEI-90-TR-21, Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst (1990)

Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer **36**(1), 41–50 (2003) https://doi.org/10.1109/mc.2003.1160055

Kim, Y., Eddins, A., Anand, S., *et al.*: Evidence for the utility of quantum computing before fault tolerance. Nature **618**, 500–505 (2023) https://doi.org/10.1038/s41586-023-06096-3

Leymann, F.: Towards a pattern language for quantum algorithms. In: International Workshop on Quantum Technology and Optimization Problems, pp. 218–230 (2019). https://doi.org/10.1007/978-3-030-14082-3_19 . Springer

Leymann, F., Barzen, J., Falkenthal, M., Vietz, D., Weder, B., Wild, K.: Quantum in the Cloud: Application Potentials and Research Opportunities. arXiv:2003.06256 (2020). https://arxiv.org/abs/2003.06256

Luo, W., Zhao, J., Zhan, T., Guan, Q.: Adaptive job scheduling in quantum clouds using reinforcement learning. In: Proceedings of the 54th International Conference on Parallel Processing, pp. 658–667 (2025). https://doi.org/10.1145/3754598.3754641

Murali, P., Abdo, P., Alexandrescu, A., Arroyo, F.R., Barkoutsos, P.K., Cross, A., Gambetta, J.M., Mohseni, M., Temme, K., Wood, C.J.: Qiskit Runtime: A Quantum-Classical Execution Platform for Cloud-Accessible Quantum Computers. APS March Meeting 2022, Abstract B47.00010. IBM Research Technical Report (2022). https://research.ibm.com/publications/qiskit-runtime-a-quantum-classical-execution-platform-for-cloud-accessible-quantum-computers

Murali, P., Baker, J.M., Javadi-Abhari, A., Chong, F.T., Martonosi, M.: Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers. In: Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '19), pp. 1015–1029 (2019). https://doi.org/10.1145/3297858.3304075

Murillo, J.M., Garcia-Alonso, J., Moguel, E., Barzen, J., Leymann, F., Ali, S., Yue, T., Arcaini, P., Pérez-Castillo, R., Guzmán, I., Piattini, M., Ruiz-Cortés, A., Brogi, A., Zhao, J., Miranskyy, A., Wimmer, M.: Quantum software engineering: Roadmap and challenges ahead. ACM Trans. Softw. Eng. Methodol. **34**(5) (2025) https://doi.org/10.1145/3712002

Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information, 10th anniversary edn. Cambridge University Press, Cambridge (2010)

Oh, C., Kim, S., Park, I.: A review of mape-k loop applications in cloud, iot, and cyber-physical systems. In: Proceedings of the 18th International Conference on Autonomic Computing (ICAC '23), pp. 69–78 (2023). https://doi.org/10.1145/3589619.3595092

Piattini, M., Peterssen, G., Perez-Castillo, R., Hevia, J.L., Serrano, M.A., Hernández, G., Guzmán, I., Paradela, C.A., Polo, M., Murina, E., Jiménez, L., Marqueño, J.C., Gallego, R., Tura, J., Phillipson, F., Murillo, J.M., Niño, A., Rodríguez, M.: The talavera manifesto for quantum software engineering and programming. In: QANSWER 2020 QuANtum SoftWare Engineering & pRogramming. Proceedings of the 1st International Workshop on the QuANtum SoftWare Engineering & pRogramming, Talavera de la Reina, Spain, pp. 1–5 (2020). http://ceur-ws.org/Vol-2561/paper0.pdf

Pohl, K., Böckle, G., Der Linden, F.J.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer, Berlin, Germany (2005)

Preskill, J.: Quantum computing in the nisq era and beyond. Quantum **2**, 79 (2018) https://doi.org/10.22331/q-2018-08-06-79

Päßler, J., ter Beek, M.H., Damiani, F., Johnsen, E.B., Tapia Tarifa, S.L.: Analysing self-adaptive systems as software product lines. Journal of Systems and Software **222**, 112324 (2025) https://doi.org/10.1016/j.jss.2024.112324

Ramesh, V., Glass, R.L., Vessey, I.: Research in computer science: An empirical study. Journal of Systems and Software **70**(1–2), 165–176 (2004) https://doi.org/10.1016/S0164-1212(03)00015-3

Ravi, G.S., Smith, K.N., Murali, P., Chong, F.T.: Adaptive job and resource management for the growing quantum cloud. In: 2021 IEEE International Conference on Quantum Computing and Engineering (QCE), pp. 301–312 (2021). https://doi.org/10.1109/QCE52317.2021.00047 . IEEE

Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research

in software engineering. Empirical Software Engineering **14**(2), 131–164 (2009) https://doi.org/10.1007/s10664-008-9102-8

Seitz, P., Geiger, M., Ufrecht, C., Plinge, A., Mutschler, C., Scherer, D.D., Mendl, C.B.: Scim milq: An hpc quantum scheduler. In: 2024 IEEE International Conference on Quantum Computing and Engineering (QCE), vol. 2, pp. 292–298 (2024). https://doi.org/10.1109/QCE60285.2024.10294 . IEEE

Sepúlveda, S., Pérez-Castillo, R., Piattini, M.: Dynamic software product lines for run-time adaptations in quantum software. In: 2025 IEEE International Conference on Quantum Computing and Engineering (QCE), vol. 2, pp. 156–161 (2025a). https://doi.org/10.1109/QCE65121.2025.10312 . IEEE

Sepúlveda, S., Pérez-Castillo, R., Piattini, M.: A software product line approach for developing hybrid software systems. Information and Software Technology **178**, 107625 (2025b) https://doi.org/10.1016/j.infsof.2024.107625

Sepúlveda, S., Piattini, M., Pérez Del Castillo, R.: Developing hybrid quantum-classical software: a software product line approach. In: Proceedings of the 5th ACM/IEEE International Workshop on Quantum Software Engineering (Q-SE), pp. 37–40. ACM, New York, USA (2024). https://doi.org/10.1145/3643667.3648224

Sepúlveda, S., Tapia-Martin, J., Cravero, A.: QuASAr Literature Evidence Report: Runtime Self-Adaptation in Hybrid Quantum–Classical Software Systems. Zenodo (2026a). https://doi.org/10.5281/zenodo.22925564

Sepúlveda, S., Perez-Castillo, R., Tapia-Martin, J., Candia, S., Navarro, C., Cravero, A.: QuASAr Architecture and Technical Companion. Zenodo (2026b). https://doi.org/10.5281/zenodo.22925733

Serrano, M.A., Pérez-Castillo, R., Piattini, M.: Quantum Software Engineering. Springer, Switzerland (2022). https://doi.org/10.1007/978-3-031-05324-5 . https://link.springer.com/book/10.1007/978-3-031-05324-5

Shinde, S., Szefer, J.: Quantum fidelity-per-cost: A metric for evaluation of quantum computing systems. In: 2026 IEEE International Conference on Quantum Computing and Engineering (QCE). IEEE, Toronto, ON, Canada (2026). https://arxiv.org/abs/2607.28572

Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., *et al.*: A survey on large language model based autonomous agents. Frontiers of Computer Science **18**(6), 186345 (2024) https://doi.org/10.1007/s11704-024-40231-1

Weder, B., Barzen, J., Leymann, F., Salm, M.: Automated quantum hardware selection for quantum workflows. Electronics **10**(8), 984 (2021a)

Weder, B., Barzen, J., Leymann, F., Zimmermann, M.: Hybrid quantum applications need two orchestrations in superposition: A software architecture perspective. In: Proceedings of the 18th IEEE International Conference on Web Services (ICWS 2021), pp. 1–13. IEEE, Chicago, IL, USA (2021b). https://doi.org/10.1109/ICWS53863.2021.00015

Weder, B., Barzen, J., Leymann, F., Vietz, D.: Quantum software development lifecycle. In: Serrano, M.A., Pérez-Castillo, R., Piattini, M. (eds.) Quantum Software Engineering, pp. 61–83. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-05324-5_4

Wieringa, R.: Design Science Methodology for Information Systems and Software Engineering. Springer, Berlin, Heidelberg (2014)

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering. Springer, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29044-2

Xiao, Z., Tapia, J., Díaz, J., Sepúlveda, S., Ancán, O., Cares, C.: Engineering control in LLM-based recommender systems. In: Proceedings of the 21st International Conference on Evaluation of Novel Approaches to Software Engineering, Benidorm, Spain. ENASE 2026, vol. 1, pp. 402–409. SCITEPRESS, Setúbal, Portugal (2026). https://doi.org/10.5220/0014829400004015

Ye, B., Khan, A.A., Pihkakoski, T., Liang, P., Azeem Akbar, M., Silveri, M., Malmi, L.: C2|Q⟩: A robust framework for bridging classical and quantum software development. ACM Trans. Softw. Eng. Methodol. (2026) https://doi.org/10.1145/3803018

Zhao, J.: Quantum Software Engineering: Landscapes and Horizons. arXiv:2007.07047 [cs.SE] (2020). https://arxiv.org/abs/2007.07047